# TOWARDS THE ADAPTABILITY OF SERVICE-BASED SYSTEMS:

## MEASUREMENT, REQUIREMENTS VARIABILITY, AND CONTEXT-AWARE RECOMMENDATION

A THESIS SUBMITTED TO AUCKLAND UNIVERSITY OF TECHNOLOGY
IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Primary Supervisor: Assoc Prof Jian Yu
Secondary Supervisor: Assoc Prof Quan Bai

June 2020

By
Khavee Agustus Botangen
School of Engineering, Computer and Mathematical Sciences

# Abstract

The recent advances in technologies supporting ambient, ubiquitous, and mobile computing have motivated the integration of service-based software systems into people's everyday lives. This integration requires software adaptability, to meet the ever-changing needs and desires of users in a volatile environment. However, achieving software adaptability is inherently complex, with extremely diverse concerns. A framework to establish the basis of organising and conceptualising adaptability solutions has become necessary. The thesis has explored an existing adaptability conceptual framework that envisions the integration of adaptability processes with the service composition life cycle. Such a framework provides an abstract of a holistic approach for adaptability spanning throughout the service composition life cycle phases. In particular, this thesis aims to contribute to the envisioned adaptability framework that demands coherent approaches to fulfil the multitude of adaptability needs at every phase of the service composition life-cycle. The thesis has addressed three adaptability challenges within the requirements engineering & design, and construction phases of the service composition life-cycle.

The first challenge is the need to quantify adaptability. A set of adaptability metrics has been defined based on two underpinning concepts: the *structure variability* and *binding variability* of composite services. A case study has demonstrated the practical use of the metrics both to evaluate the adaptability of compositions specified from various Business Process Execution Language (BPEL-based) frameworks, regardless

of the implemented adaptability mechanism, and to compare adaptability of the composition specification frameworks. The case study has indicated that the metrics can aid developers and designers in their decisions toward creating adaptable composite services. Hence, the inclusion of adaptability metrics to the service composition adaptability framework facilitates the design and evaluation of adaptability in the service composition life cycle.

The second challenge is the need for a context-aware requirements variability analysis. An approach for requirements variability analysis has been proposed. The approach utilises a *contextual goal model* that specifies relationships between goals and context variations, and a *contextual preference model* that enhances preferences with contextual information to capture the changing nature of preferences caused by context variation. The approach integrates the contextual preferences and contextual goals, in which the combined influence of context variability and preference variability is realised when deriving requirements variants. The result of a case study has indicated the effectiveness of the approach in identifying requirements modelling errors and inconsistencies, thus facilitating the refinement of requirements variability models. The approach expands the realisation of adaptability from the early phase of the service composition life cycle, that is, from the stakeholders' level of requirements engineering.

The third challenge is the need for a context-aware service selection. A geographic-aware collaborative filtering service recommendation approach that deals with implicit user-service invocation data has been proposed. The approach examines the mashup-API invocation scenario where the user is a service composition (i.e., the mashup) invoking a Web service (i.e., the API). The approach has a method that maps geographical location information into geographical relevance scores. The geographical relevance, combined with functional relevance, is integrated into a probabilistic matrix factorisation recommendation model. The resulting recommendation model utilises

both the geographical and functional relevance to infer the preference degrees underlying the implicit mashup-API invocation data. The experiment results have shown that augmenting the implicit invocation data with geographical location information, in addition to functional descriptions, increases the precision of API recommendation for mashup services. The approach can be significant to the adaptability of service selection tasks in the service composition life cycle.

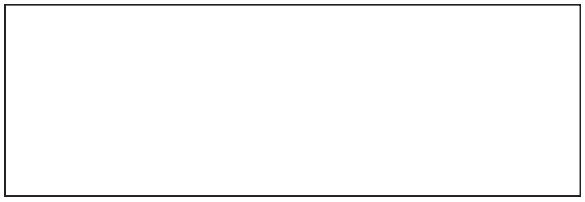
# Contents

# List of Tables

# List of Figures

# Attestation of Authorship

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent has been accepted for the qualification of any other degree or diploma of a university or other institution of higher learning.



Signature of candidate

# Publications

1. Botangen, K. A., Yu, J., & Sheng, M. (2017, January). Towards measuring the adaptability of an AO4BPEL process. In Proceedings of the Australasian Computer Science Week Multiconference (p. 6). ACM.

2. Botangen, K. A., Yu, J., Yongchareon, S., Yang, L. H., & Bai, Q. (2018, January). Specifying and reasoning about contextual preferences in the goal-oriented requirements modelling. In Proceedings of the Australasian Computer Science Week Multiconference (p. 47). ACM.

3. Botangen, K. A., Yu, J., Yongchareon, S., Yang, L. H., & Sheng, Q.Z. (2019, October). Integrating Geographical and Functional Relevance to Implicit Data for Web Service Recommendation. In Proceedings of the International Conference on Service-Oriented Computing (pp. 53-57). Springer, Cham.

4. Botangen, K. A., Yu, J., Han, Y., Sheng, Q. Z., & Han, J. (2020, January). Quantifying the Adaptability of Workflow-based Service Compositions. *Future Generation Computer Systems*, *102*, 95-111.

5. Botangen, K. A., Yu, J., Yeap, W.K., & Sheng, Q. Z. (2020, January). Integrating Context to Preferences and Goals for Goal-oriented Adaptability of Software Systems. *The Computer Journal*, https://doi.org/10.1093/comjnl/bxz167.

6. Botangen, K. A., Yu, J., Sheng, Q. Z., Han, Y., & Yongchareon, S. (2020, August). Geographic-aware Collaborative Filtering for Web Service Recommendation. *Expert Systems With Applications*, *151*, 113347.

# Acknowledgements

I am extremely grateful to my primary supervisor Associate Professor Jian Yu, whose guidance, encouragement, and enthusiasm have motivated my research over the last four years. It was an exceptional experience to have such a dedicated supervisor. It was his belief in me, his constant push and tireless support, that enabled me to strive and reach this point in my research journey.

This work would not be possible without the full-support and generous provisions of the New Zealand Scholarships programme of the Ministry of Foreign Affairs and Trade. It was a great priviledge to be a recipient of this scholarship grant. Likewise, the pro-active and endless support of the AUT New Zealand Scholarships team members Sacha Pointon, Margaret Leniston, Ruth Sullivan, Petrina Hibben, Prerna Taneja, Sandelyn Su Kuan Lua, and George Kimani, has been helpful throughout my research journey.

I am thankful to my secondary supervisor Associate Professor Quan Bai for the always-available support. I am also thankful to my co-authors, Professor Quan Z. Sheng, Professor Wai Kiang Yeap, Dr Sira Yongchareon, and Professor Jun Han, for their contribution in various portions of my research.

I thank all my colleagues in WT 404 Research Lab and the Software Engineering Research Lab, especially Aminu Usman, Jing Ma, Chamari Kithulgoda, Herman Wandabwa, Naga Kunchala, Sarita Pais, David Airehrour, Emmanuel Ndashimye, Olayinka Adeleye, Maojun Wang, Reza Mohseni, Bikash Pokhrel, Jehan Badshah, Kan Ngamakeur, and Anita Rabieyan, for the friendship and snap chats.

I also thank all my New Zealand-based friends – Filipino cohorts and *kababayans*, and especially Cordilleran *kakailians* – for the catchups, travels, eat & drink gatherings, and fun-times that made my research journey memorable and colourful.

Finally, I thank my family for supporting me throughout my PhD journey. I am especially grateful to my wife Edz for being always there as a source of inspiration, and to my parents, Khaloy and Viola, for their incredible support and sacrifice.

To all of you, many thanks, *maraming salamat*, *dakel ni iyam-man*!

# Dedication

I dedicate this work to these wonderful human beings:
To my parents Khaloy and Viola, who wilfully brought me into this existence;
To my uncle Artas, whose goodness helped me finish college;
To my wife Edz, who has been constantly annoying and photobombing me;
And to them foolish and hungry, who don't want to give up.

# Chapter 1

# Introduction

Adaptability is the ability of a software system to fit its behaviour to the changes in its environment (Tekinerdogan & Aksit, 1996; Andresen & Gronau, 2005; Evans & Marciniak, 1987). Adaptability is a notion rooted from nature, particularly in the life sciences (Conrad, 1983; Fisher & Ford, 1926; Doncaster, 1917), and has transcended into the realms of computer science. Natural occurrences that actualise adaptability, such as evolution, natural selection, variation, and adaptation, have been inspiring various computing disciplines, as exemplified by a plethora of evolutionary and bio-inspired computation methods (Coello, Lamont & Van Veldhuizen, 2007; L. Wang, Shen & Yong, 2012; Mantere & Alander, 2005; Eiben, 2004). Referring to Charles Darwin's evolution of species, the overall evolutionary process can be a consequence of three natural occurrences: *adaptation*, *variation*, and *selection*. Adaptation is a dynamic process that fits organisms to their environment, enhancing their evolutionary fitness. Variation produces new variants of organisms as a result of reproduction among the population. Selection is the process in which the best variants (i.e., those that have withstood environment pressures) survive, while the poor ones are left out. These processes define the "survival of the fittest" phenomenon which thoroughly necessitates two properties: *adaptability* and *variability*. On the one hand, an individual organism's ability to adapt

raises its chances to survive. On the other hand, the variability of organisms within a population (i.e., having a large number of variant organisms), increases the chances of survival of the species (Lloyd & Gould, 1993).

In software systems, adaptability makes a system behaviour respond appropriately to the changing requirements and contexts in order to "survive" (i.e., to continue to offer its required functionality and quality of service) (De Lemos et al., 2013). The idea of a software system that adapts had been considerably set aside for many years because the complexity of creating such a system hardly justified its value (Garlan, 2017). That is, early systems were typically designed for predictable, stable environments, where a static and uniform set of requirements and fault models was sufficient to fulfil their goals. However, modern contexts of software systems that have emerged with the contemporary computing developments, such as mobility and pervasiveness, service orientation, and cloud computing, seek the reconsideration of adaptability as a vital concern. Today's systems must function in a complex unpredictable environment where direct control of infrastructure, system components, and services are becoming unlikely. The traditional solution of deploying more system administrators to handle the needed changes and occurrences of unexpected faults (e.g., re-engineering the system through updates, or re-configuring the system), has become increasingly cost-prohibitive. Besides, such human intervention cannot scale with the demand for immediate adaptation or the growing complexity of systems. Consequently, there has been a recent surge of interests on innovative ways to make systems variable, flexible, and adaptable, without compromising quality and development cost, and significantly, without taking systems offline (De Lemos et al., 2013; Galster, Weyns, Tofan, Michalik & Avgeriou, 2014; Mistrik, Ali, Kazman, Grundy & Schmerl, 2017).

New technological forces and the shifting technology use landscape are motivating efforts to integrate adaptive systems in today's technologically rich ecosystems. For example, cyber-physical and internet of things (IoT) systems have been facilitating

the control systems' perspective of adaptability. Such perspective leverages the use of sensors to monitor situations, actuators to perform runtime changes, and reasoning mechanisms to decide when it is appropriate to adapt, and how best to do so (Garlan, 2017; Brun et al., 2009). Recent advances in sensor systems and mobile devices have boosted realisations of the needed adaptability in software systems. These technologies can facilitate the capture and provision of context, which has been considered a core element for adaptability. Consequently, software systems have increased their reliance on contextual information that weaves the software into its volatile operating environment. This is necessary to better understand and enhance the usability of a functionality in as many situations as possible. As such, these emerging technologies are providing vast opportunities and facilities to realise software adaptability (Bennaceur et al., 2016).

Adaptability has become a consequence of the need for continuous and effective operation. Vital properties, such as criticality (e.g., in software that runs power grids), accessibility and availability (e.g., of software that handles international banking), require systems to run continuously. Software systems are becoming increasingly interconnected, one being a part of another, or (particularly for multiple heterogeneous systems) becoming components of large civil and business infrastructures. Such complexity requires a software that can operate in a highly dynamic environment, in which user requirements constantly change, and system interactions and business goals evolve.

Moreover, due to the ever-increasing integration of software systems into the fabrics of society, adaptability has become increasingly necessary. The growing necessity of personalised information systems (Gaß, Ortbach, Kretzer, Maedche & Niehaves, 2015) to support individual activities of everyday living (i.e., both personal and business activities) has demanded a shift in software adaptability efforts, from the traditional norm of users adopting a software, to the paradigm of software that adapts to a particular user (Liaskos, Lapouchnian, Yu, Yu & Mylopoulos, 2006a). Several societal, economic, and technological trends have been motivating personalisation efforts. For instance, the

global trend of ageing populations has increased the interest in technologies that are inclusive of the challenged user groups (Kostavelis, Giakoumis, Malasiotis & Tzovaras, 2015). This entails systems that adapt to individual needs and the capabilities of elderly who experience physical or cognitive limitations. Likewise, the aforementioned technological advances in the various computing and communication domains (e.g., mobile and ubiquitous computing, ambient intelligence, smart homes, and autonomic computing) have inspired a wider-range integration of information systems into everyday life. Mobile computing, for instance, gave rise to the trend of information technology (IT) consumerisation (Bauer et al., 2013), which has shifted the access to organisational services through arbitrary mobile devices. Mobile devices such as smart phones have the potential to serve as gateways to all the personal or business services an individual needs. Such services could be further personalised by taking advantage of the relevant context, as sensed through the device. Designing adaptable software systems that provide context-dependent functionality from recognising individual users' goals, characteristics, and circumstances, has become a key challenge.

Software systems need to demonstrate properties like versatility, flexibility, resiliency, dependability, energy-efficiency, recoverability, customisability, configurability, and self-optimisability, to meet the ever-changing needs and desires of users. Such desirable properties can be achieved by adapting to the changes that are occurring in both operational contexts and system requirements (De Lemos et al., 2013).

## 1.1 Service Composition for Modern Software Systems Development

Service composition (SC) is a modern software development technique that utilises the service-oriented computing (SOC) paradigm (Dustdar & Schreiner, 2005; Papazoglou

& Van Den Heuvel, 2007; Bouguettaya, Sheng & Daniel, 2014b; Lemos, Daniel & Benatallah, 2016) in creating service-based software systems. The general idea of service composition is recursively aggregating existing (atomic) services to create a value-added service, called *composite service* or *business process*, to realise a business goal that cannot be achieved by a standalone service. Most existing service composition methods are built on the traditional Web service technologies, e.g., an enterprise system composed of Web services in which each component Web service encapsulates a complete business functionality. The following are the basic concepts and issues surrounding the typical Web service composition (see Sheng et al., 2014):

- *Orchestration vs choreography*: orchestration and choreography (Peltz, 2003) are concepts describing the control of integration and coordination of Web services that make up a business process. On the one hand, orchestration uses a centralised perspective where there is a single entity (i.e., the composite service) that coordinates the interaction among the involved services. A widely supported standard for Web services orchestration is WS-BPEL[1] (Web Services Business Process Execution Language). On the other hand, choreography follows the decentralised approach, which is more collaborative and allows each participating Web service to describe its role in the interaction. A standard specification for Web services choreography is defined by the WS-CDL[2] (Web Services Choreography Description Language).
- *Static vs dynamic composition*: static (services to be composed are decided at design time) and dynamic (services to be composed are decided at runtime) composition (Dustdar & Schreiner, 2005) regard the time when the Web services are integrated. In a static composition, component services are selected and bound at design time. This may work well when business partners in the composition

[1] http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html
[2] https://www.w3.org/TR/ws-cdl-10/

are fixed, and service functionalities or composition requirements do not or rarely change (Sheng et al., 2014). In contrast, dynamic composition enables selection and binding of component services at runtime. This is ideal in a volatile operational environment that needs an adaptable composite service suitable for frequent runtime changes in requirements or services.

- *Manual vs semi-automatic or automatic composition*: manual (user-driven) and automatic (no-user intervention) composition define the manner of aggregating services. In the manual composition, a human designer creates an abstract composite process[3] then manually selects and binds Web services to the abstract process. WS-BPEL can be used to specify manual compositions. However, Web service composition has become a complex task and the human capability to manually deal with it is insufficient. Several issues contribute to this complexity, such as the dramatic increase in the number of Web services available across the Web, the need to compose on the fly based on the current context, and the heterogeneity of Web services. The automatic composition typically aims for the automated discovery, selection, invocation, composition, and interoperation of Web services. Automatic service compositions have been utilising Artificial Intelligence (AI) planning (Rao & Su, 2004), semantic Web (McIlraith, Son & Zeng, 2001), and dynamic workflow (Casati & Shan, 2001) approaches. A standardisation effort aimed to facilitate automatic service composition is the OWL-S[4]. It is a Web Ontology Language (OWL)-based Web service ontology that defines a core set of constructs for describing the properties and capabilities of Web services in unambiguous, computer-interpretable form. Meanwhile, semi-automatic (hybrid) composition enhances the manual approach with some automatic approaches. A model-driven service composition (Gronmo, Skogan,

[3]The term *process* is commonly used to refer to a composite service.
[4]https://www.w3.org/Submission/OWL-S/

Solheim & Oldevik, 2004) is a typical example, which automatically generates a service composition specification (i.e., expressed in BPEL) from the abstract composition models (e.g. UML) that represent the requirements, workflow, and component services.

However, technological developments that come with the evolution of the Internet (Perera, Zaslavsky, Christen & Georgakopoulos, 2014), particularly connecting mobile devices, people (i.e., social networks), and everyday objects to the Internet, expand the scope of a service to notions like "everything as a service" (Duan et al., 2015). The Internet of Things phenomenon, for instance, is bringing the massive rise of smart objects (e.g., smartphones, smart-TVs, smart-appliances, smart public displays, robots, smart-cars, weather sensors, and traffic sensors). These objects are connected via standard protocols, such that they become accessible through the Internet or an ad-hoc network. Since each "thing" in the world could be encapsulated and defined as a service to mediate its use, e.g., there are efforts to integrate heterogeneous physical objects with the traditional Web services (Guinard, Trifa & Wilde, 2010; Han & Crespi, 2017; Meyer, Ruppen & Magerkurth, 2013), a composition of diverse smart objects and Web services can be performed to meet even the most complex user goal (Ko, Ko, Molina & Kwon, 2016).

Service composition is driving applications, and both software and hardware resources, to become loosely coupled, standardised, inter-operable, reusable, and dynamically integrated components of software systems. Service composition is strongly recognised to propel online, service-enabled business transformation, which creates an opportunity for seamless delivery of services over the Internet, minimising costs and maximising the potential market. Likewise, SC opens channels for alliances, enables outsourcing of functionalities, and provides a one-stop shop for business clients. Service

composition has become a promising development approach for modern software systems, given the enormously growing number of Web services, and taking into account the rising IoT and cloud services (Miorandi, Sicari, De Pellegrini & Chlamtac, 2012; Wei & Blake, 2010; Tsai, Sun & Balasooriya, 2010). For example, a survey in 2008 discovered 5,077 traditional (WSDL-described) Web services (Al-Masri & Mahmoud, 2008); a survey in 2013 discovered 6,686 cloud services (Noor, Sheng, Ngu & Dustdar, 2014); and, by the second quarter of 2019, there were already over 21,600 Web services in one of the largest service repository[5]. This emerging *service Web* – "a web that allows billions of parties to expose and consume billions of services" (Domingue, Fensel, Davies, González-Cabero & Pedrinaci, 2009, p.204), motivates SC to be a promising efficient, on-demand modern development approach, since a component service can be delivered wherever, however, and whenever it is needed.

The following are some other prevailing technology trends that have brought opportunities for service composition as an ideal development approach for modern software systems.

- Cyber-physical Systems – this phenomenon is the consequence of the rise of interconnected physical objects (e.g., IoT) that have their own computational and physical capabilities and can interact with humans. The capabilities of these entities can be wrapped and provided as services (e.g., just-in-time integration of networked physical entities into desired capabilities). Enabled by these interconnected "smart" objects, the vast number of new *always-responsive services* will shift the current vision of *always-on services*, typical of the Web era, to *always-responsive situated services* which are dynamically composed and tailored to the specific requirements and context of use (Miorandi et al., 2012).
- Mobility – comScore[6] reported that from early 2014 in the United States (U.S.)

[5] www.programmableweb.com
[6] https://www.comscore.com/

alone, the Internet usage on mobile devices exceeded personal computer (PC) usage. It is said that the landscape in which businesses operate has changed forever. Different business aspects like management, marketing, sales, search engine optimisation (SEO), social media, advertising, app development, or customer relationship management, have embraced mobile devices as a preferred tool for work and communication. This has driven the integration of mobile-friendly services for all mission-critical business assets (e.g., applications, data, and customer service). Another forecast by the International Data Corporation (IDC)[7] in 2015, predicted that the U.S. mobile worker population will grow to 105 million in 2020, this would account for almost three quarters of the total U.S. workforce.

- "IT Consumerisation" – refers to the blending of personal and business (e.g., work) use of technology devices and applications. This blurs the lines between which tools are for work and which are for personal use. Today's mobile devices (e.g., smartphones) have the potential to serve as gateway to all personal services an individual needs (Bauer et al., 2013). The devices are personalised and have access to all relevant personal context data, like calendars or address books, but also to the sensors of the devices, like global positioning systems (GPS) or wireless technologies. Likewise, these could utilise other (volatile) smart devices in the vicinity to provide context-adequate user experiences, such as connecting to a smart TV or entertainment device at home, car sensors, or to company resources at work.

Accordingly, services will continue to play a key role as the building blocks of modern software systems. Services provide abstractions and encapsulations, not just of processes, applications, data storage, and networks, but of anything like devices, physical goods, and human-provided services (Schall, Truong & Dustdar, 2008), e.g.,

[7] http://www.idc.com/

BPEL4People[8], and WS-HumanTask[9]. Other abstractions include work as a service (Oppenheim, Varshney & Chee, 2014), services encapsulating content and the associated intellectual property, and services abstracting physical presence (Kindberg et al., 2002).

## 1.2 Adaptability in Service Composition

The service composition (SC) adaptability framework, which encompasses the works in this thesis, is an integrated SC life cycle and adaptation life cycle (Bucchiarone et al., 2009). It envisions a holistic and comprehensive framework for adaptability that spans all SC life cycle phases, thus enabling the integration of different, isolated, and fragmented adaptability solutions.

### 1.2.1 The service composition life cycle

The main phases of a service composition life cycle are *requirements engineering & design*, *construction*, *deployment*, and *execution* (Sheng et al., 2014). These phases are represented as rectangles with rounded corners in Figure 1.1 on page 28. Figure 1.1 also shows a rectangle with broken lines surrounding the phases which are the focus in this thesis. In the requirements engineering & design phase, the composition requirements are specified (i.e., by using a requirements model) to provide information about the user demands and preferences sufficient for the composition. The requirements include the *functional* ones, which specify a set of business activities that include control and data flow among them, the *non-functional* ones, which cover quality of service (QoS) requirements, and the exception-handling behaviours. An *abstract composite service* model is either semi-automatically or automatically generated to represent the composition requirements. The construction phase includes composition modelling,

[8]http://docs.oasis-open.org/bpel4people/bpel4people-1.1-spec-cs-01.pdf
[9]http://docs.oasis-open.org/bpel4people/ws-humantask-1.1-spec-cs-01.pdf

service discovery, and service selection and contracting. It creates a *constructed service composition*, wherein the activities are matched with suitable Web services that satisfy the requirements. Component services are commonly searched from a service registry that usually contains the description about a service (e.g., a provider's advertised service information). The best candidate among the located concrete services should be selected. Since there are several services offering the same functionality, a typical service selection relies on services that best fulfil the non-functional requirements.

In the deployment phase, the constructed service composition is deployed as an *executable composition* to allow instantiation and invocation by end-users. In the execution phase, a *service composition instance* is created and executed by the execution engine, which is also responsible for invoking the concrete partner services. The execution phase activities include dynamic service binding (i.e., late binding), maintenance measures, and monitoring tasks (e.g., logging, execution tracking, performance measuring, and exception handling). Service discovery can continue at the execution phase, and dynamically updates the list of candidate services.

### 1.2.2 Adaptability integration to the service composition life cycle

Service composition offers a huge opportunity for an open, distributed, and on-demand approach of software development, but requires fundamental changes to the way traditional software is developed, deployed, and maintained. In particular, the highly dynamic setting in which composite services need to operate necessitates adaptability as a key capability to consider. Adaptability enables adjustments to the composite service in response to either optimisation needs or changes in the execution environment. Such changes include variations in the functionality offered by component services, unavailability of component services, decreasing performances, availability of better services, arising of new business requirements, changing needs of specific users, and

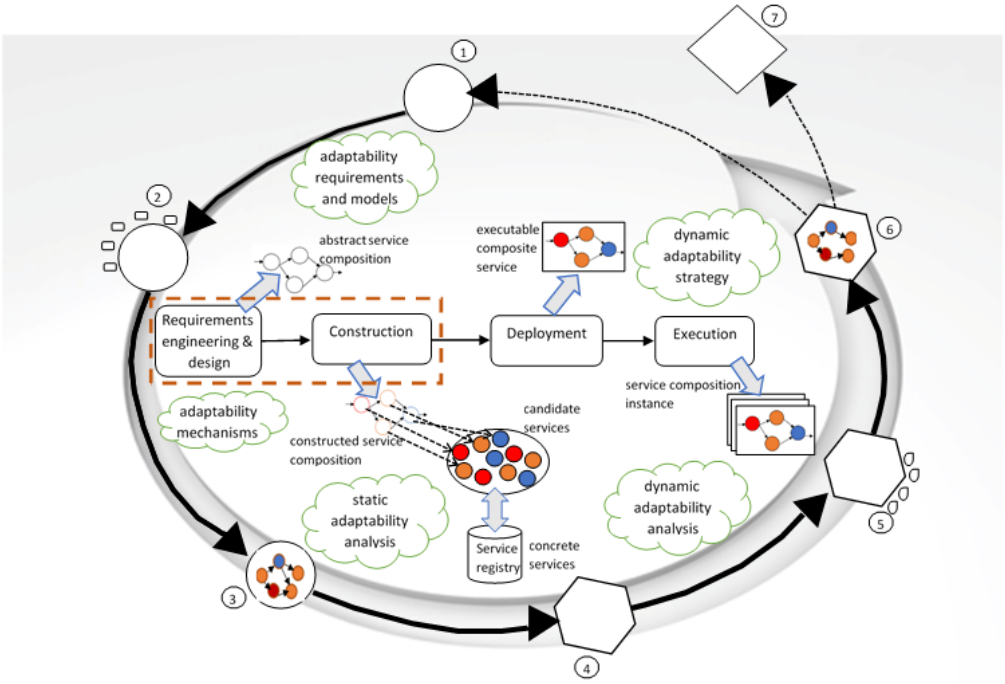



Figure 1.1: The adaptation life cycle of a service composition

changing situations and context of use. The service composition life cycle has to incorporate the facilities for adaptability from the very early phase (i.e., requirements engineering) up to the (post) execution phase. The abundance of efforts addressing the challenges of handling various adaptability needs, such as inter-operability amongst services (Williams, Battle & Cuadrado, 2006), customisation (Sam, Boucelma & Hacid, 2006), failure recovery and repair (Pernici & Rosati, 2007), self-optimisation (Di Nitto, Di Penta, Gambi, Ripa & Villani, 2007), and context-aware adaptation (Kapitsaki, Prezerakos, Tselikas & Venieris, 2009), demands a comprehensive framework that consolidates the adaptability mechanisms from all SC phases into a coherent set of solutions. The initial efforts towards such an SC adaptability framework (Bucchiarone et al., 2009; Kazhamiakin et al., 2010; Lane, Gu, Lago & Richardson, 2014), have led to the identification of competencies and gaps in the SC adaptability research, and consequently, have motivated conceptualisation of the work bundled in this thesis.

Figure 1.1 shows an impression of the SC adaptability framework integrating the SC

life cycle with the adaptation life cycle. It implies how the adaptation should be realised as a consequence of holistic and coherent adaptability mechanisms assimilated at every phase throughout the SC life cycle. The cloud entities represent abstractions of artefacts generated in the adaptability activities of the SC phases. A detailed discussion of the SC adaptability framework is found in Chapter 2, Sec. 2.2.6. Meanwhile, the adaptation life cycle is represented by the outer elements arranged in a circular form, showing a particular composition that adapts to changing contexts. For instance, a composition at a certain stable state and configuration executes while observing its context (1). The information obtained by its aware parts, such as sensors, indicates a change in the context. These changes are symbolised by the small rectangles surrounding the composition (2). In order to fulfil its intended goals, the composition may use these stimuli to activate adaptability mechanisms, e.g., change its structure or reconfigure (3). In this case, the composition is now adapted to a new usage context, and the effects of the adaptation become visible (4). In every change in context detected (5), the composition uses its adaptability mechanisms (6) to return to a previous configuration (1) or change into a new configuration (7).

It is no longer reasonable for service-based systems[10] to be engineered for a static set of requirements on a stable, fixed environment. Treating adaptability as a first-class concern is inevitable for three reasons. *First*, compositions rely on component services that are not in the direct control of the system integrator (e.g., systems developer). Hence, a composition-based software system is no longer owned by a single organisation, but is distributed and shared amongst many parties. This results to unpredictability such as: unforeseen changes in the provided functionality or behaviour of the component services, a decrease in quality of a component service, or a component service becomes unavailable for a certain period. For example, considering the smart objects in the IoT as

[10]In this thesis, the term *service-based systems* refer to software systems developed via service composition.

component services, their availability can be a challenge because of limited bandwidth or having many users compete for the same set of objects. *Second*, service compositions are designed to address a wide range of service consumers having their own preferences, constraints, service-level agreements, and requirements. The variations and frequent changes in these properties require customisation and personalisation to adapt the composition behaviour to a particular user. *Third*, a composition's execution context and usage context (e.g., location, device-used, time, user, and environment) constantly change. This has become widespread with the introduction of mobility. A change in context may result in changes to the requirements.

There are two cornerstones of the adaptability mechanisms in the SC adaptability framework (Mens, Capilla, Cardozo & Dumas, 2016; S. H. Chang & Kim, 2007; T. Nguyen, Colman, Talib & Han, 2011): *variability* and *context-awareness*. On the one hand, variability prompts different variants of a composite service that can be adapted for different users and changing situations. Such variability includes: logic variability – the variation of the algorithm or logical procedure in the operations of a component service providing a certain functionality, binding variability – the variation of the selection of a component service among candidate services, and structure variability – the variation in the workflow of a composition[11]. On the other hand, context-awareness enables compositions to be 'smart' software systems, utilising context to provide relevant services to users (Alegre, Augusto & Clark, 2016). A smart composition can understand and react to situational circumstances, producing dynamic results based on who, what, when, where, and why it was utilised (Kazhamiakin et al., 2010; Manes, 2001).

[11]The term *composition* similarly refers to a composite service.

## 1.3 Research Questions

The SC adaptability framework demands for techniques and modelling approaches to fulfil the adaptability mechanisms at every phase of the SC life-cycle. The work presented in this thesis focuses on addressing three challenges, which belong to the adaptability needs within the first two phases of the SC life cycle (i.e., requirements engineering & design, and construction).

1. **Quantifying adaptability.** Software quality attributes that are quantifiable will significantly support the analysis of the software design, to ensure it has the qualities the designer requires. In this case, quality attributes can be compared among alternative designs. A lot of work has been done in designing software systems prioritising qualities such as reliability, performance, security, and maintainability. The styles, analysis, and patterns are well understood for such properties, and standardisations of their evaluations are already in place, e.g., SQuaRE (ISO, 2011). However, incorporating adaptability as a design quality attribute and how it affects other properties has received little attention (Schmerl, Kazman, Ali, Grundy & Mistrik, 2017). Some perspectives of adaptability have long been recognised as quality characteristics and been part of some standardised software quality models (e.g., the ISO/IEC standard [ISO, 2001] defines adaptability as a sub-characteristic of portability). Such perspectives need to vary to accommodate adaptation aspects of higher relevance to service-based systems.

   Addressing this first challenge raises the research question: *(RQ1) How do we quantify the adaptability of service compositions?*

2. **Context-aware requirements variability**. While it has been argued that requirements engineering (RE) is an important phase in software systems development (Castro, Kolp & Mylopoulos, 2002), gap in the RE phase of service composition on the approaches and techniques for handling adaptability (Ralyte, 2012). For

instance, it was revealed in the literature surveys on software variability (Galster et al., 2014), quality attribute variability (Mahdavi-Hezavehi, Galster & Avgeriou, 2013), and context-aware systems (Alegre et al., 2016) that only a few studies address adaptability in the RE phase. It has been observed that most studies on SC adaptability are solution-space-oriented: technically-focused in the architecture, design, and implementation phases (Sheng et al., 2014; Lemos et al., 2016). In another survey (Guinea, Nain & Le Traon, 2016) on engineering software for ubiquitous systems, where explicit handling of context adaptation is necessary, it was also revealed that little attention has been given to the RE phase. There is a need to explicitly represent the underlying requirements (i.e., problem-space) that lead to the technical adaptability approaches. Furthermore, RE has been identified as a challenge in a research roadmap for adaptive systems, particularly on modelling of requirements variability and managing it, which sets the design space for adaptability (De Lemos et al., 2013).

Requirements variability, which is typically regarded as the variations in the intentions of stakeholders for the intended use of a software product, has long been advocated in RE to establish multiple subsets of requirements (Metzger & Pohl, 2014; Galster et al., 2014). For instance, in the software product line engineering (SPLE), each subset (i.e., requirements variant) is matched to achieve the needs of a specific group of users. This allows a single software to have multiple design specifications tailored to different groups of users with differences in requirements. At the RE phase, traditional requirements models (e.g., feature models and goal models) are used to specify the variability of system features to address the requirements variability. Recently, however, with the phenomena of ubiquitous computing, IoT, and smart devices, there is a demand for dynamic and context-aware capabilities that require significant utilisation of contextual information. Many software systems must dynamically adapt to suit their behaviour to the

volatile context in which they operate. This emergent need for context-aware software systems is seeking appropriate techniques of modelling and handling context–aware software variability. Hence, the requirements models should incorporate context variability to sufficiently represent the adaptation needs of systems in which contextual changes are commonplace. A recent survey in requirements modelling and analysis for adaptive software systems (Yang, Li, Jin & Chen, 2014) revealed that context-aware requirements modelling in RE are rarely studied. Traditional requirements engineering approaches either ignore context or presume it to be constant. Most existing requirements variability analysis approaches have not explored context-awareness, individualised contexts, and preferences. Such a gap has also been explicitly amplified in several software engineering research agenda (Galster et al., 2017; Bosch, Capilla & Hilliard, 2015; Alebrahim et al., 2016; Cheng et al., 2009).

Addressing this second challenge raises the research question: *(RQ2) How do we model an explicit representation of context-aware and preference-based requirements variability for service compositions?*

3. **Context-aware service selection**. Discovering and selecting the most suitable services to be invoked for a composition, given the composition's requirements and context, has long been a challenge in the SC life cycle. The traditional service discovery methods have two categories (Sheng et al., 2014): the *syntactic* and *semantic* matching methods. The syntactic methods offer search facilities based on service names and identifiers, while the semantic matching methods utilise semantic descriptions from several aspects such as operations, inputs, outputs, and even the capabilities and behaviours of services. The latter category provides better match-making accuracy and an increased level of automation support. Since there can be multiple services providing the same functionality, effective selection methods need to be in place. Typical selection methods are

based on price, execution duration, and other quality of service (QoS) (Y. Wang & Vassileva, 2007; Sheng et al., 2014; Kritikos & Plexousakis, 2009). Although a set of generic service QoS has been proposed by the World Wide Web Consortium (W3C) working group, such as performance, reliability, scalability, integrity, and accessibility (W. D. Yu, Radhakrishna, Pingali & Kolluri, 2007), such generic criteria might not be sufficient. Hence, specifying domain-specific QoS to augment the generic ones became necessary such as privacy, trust and reputation, and usability (Y. Liu, Ngu & Zeng, 2004).

However, obtaining accurate QoS values has been difficult to achieve, thus constraining the reliability of a QoS-based service selection. On the one hand, a provider usually advertises its service QoS information, or provides an interface to access the QoS values that are monitored from the provider side. This approach is vulnerable to inaccuracies, since QoS values can be subject to manipulation favouring the provider. On the other hand, a monitor or trusted party can be used to capture or assess a service QoS, through active monitoring or collecting consumers' QoS feedback (P. Zhang et al., 2018; Jurca, Faltings & Binder, 2007). A typical monitoring technique intercepts messages exchanged between a consumer and provider, in order to estimate a QoS value. This approach poses problems on scalability, since the monitor needs to intercept all service invocations, acting as a central proxy, which becomes prone to a performance bottleneck. Likewise, the use of a 'trusted' third party to periodically probe services of their QoS values, can be expensive.

Recently, recommender systems techniques have been explored to facilitate a more effective service selection considering modern service repositories (Zheng, Ma, Lyu & King, 2010; Tang, Zhang, Liu & Chen, 2015; Shao et al., 2007; Q. Zhou, Wu, Yue & Hsu, 2019). Still, most of these recommender-based service selection approaches are dependent on those elusive QoS values. Moreover, with

the dynamic and distributed nature of service compositions, the influence of context to the service selection has to be considered, e.g., the different kinds of environmental factors such as the location of the consumer, location of the provider, and time. Besides, it has been observed that these contexts can extensively affect the monitored QoS of services (P. Zhang et al., 2018).

Addressing this third challenge raises the research question: *(RQ3) How do we model a context-aware service recommendation to facilitate the selection of component services of service compositions?*

## 1.4 Research Methodology and Objectives

Each research question raised in Section 1.3 is answered in the key chapters of this thesis: *RQ1* in Chapter 3, *RQ2* in Chapter 4, and *RQ3* in Chapter 5. Guided by the analytical steps in engineering research (see Wieringa, 2005), the strategy to answer each research question comprises three main phases: *problem analysis*, *solution analysis*, and *implementation analysis*. The problem analysis phase investigates and specifies in detail the problem to solve, as described by the research question. This phase handles the thorough investigation of related literature and the setting of objectives. Then, in the solution analysis phase, a proposed solution to the problem is specified. Likewise, the solution analysis phase includes the investigation of the properties and variants of the proposed solution, as well as the evaluation of the solution's ability to address the problem. Finally, in the implementation analysis phase, the proposed solution is implemented and further evaluated through case studies and experiments.

The analyses of the problems associated with the research questions enable the identification of sets of specific objectives to achieve. Table 1.1 presents those objectives.

Table 1.1: Research objectives in relation to the research questions

| **Research question** | **Specific objectives** |
|---|---|
| RQ1 | — Investigate various adaptability mechanisms implemented in the workflow-based service compositions<br>— Define the dimensions of composition adaptability to quantify<br>— Develop the adaptability metrics<br>— Evaluate the use of the metrics in measuring and comparing adaptability of compositions<br>— Evaluate the applicability of the metrics on various adaptability frameworks<br>— Develop a support tool for measuring adaptability |
| RQ2 | — Investigate both requirements variability and the existing variability models being used in requirements engineering<br>— Develop a variability specification model that captures, analyses, and represents context variability in both requirements and preferences<br>— Develop a reasoning approach that will automatically derive the most suitable set of requirements needed to achieve a goal in a given context, especially when multiple sets of requirements are applicable<br>— Evaluate the applicability of the approach in the requirements modelling tasks<br>— Develop a support tool to facilitate the automated derivation of requirements variants<br>— Evaluate the use of the tool in the requirements modelling tasks |
| RQ3 | — Investigate the recommender systems approaches applied in service recommendation<br>— Investigate the existing repositories of Web services<br>— Examine the impact of context (i.e., geographic location) on both invocation behaviour and QoS of real-world services<br>— Develop a service recommendation model that considers the context of both composition and potential component service<br>— Evaluate the performance of the recommendation model over a real-world service invocation data |

## 1.5 Research Contributions

This thesis aims to contribute towards the realisation of the service composition adaptability framework introduced in Section 1.2. Answering each research question has produced techniques that can fill the existing gaps among the adaptability mechanisms of the framework. More precisely, the major contributions of the thesis are as follow:

1. **Metrics to quantify the adaptability of work-flow based service compositions**. The thesis provides a set of completely defined metrics to quantify and compare the adaptability of BPEL processes (i.e., the service compositions specified using the Business Process Execution Language[12]). The metrics definitions are based on two underlying adaptability dimensions: *structure variability* and *binding variability*. The thesis also demonstrates the applicability of the metrics to the wide spectrum of BPEL-based service composition frameworks. The metrics are used to evaluate the adaptability of compositions specified from various BPEL-based frameworks regardless of the implemented adaptability mechanism. The use of the metrics is also expanded for comparing not just compositions, but also specification frameworks. The frameworks are compared based on the maximal adaptability they can provide for a certain composition. Moreover, a support tool is developed to automate the calculation of the metrics.

   The inclusion of adaptability metrics to the SC adaptability framework facilitates the design and evaluation of adaptability in the SC life cycle. Quantifying adaptability can raise developers and designers' awareness, and aid their decisions towards composing adaptable systems.

2. **A context-based requirements variability analysis for goal-oriented requirements modelling**. The thesis proposes an approach for requirements variability analysis utilising *i*) a *contextual goal model* that specifies relationships between

[12]http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html

goals and context variations, and *ii*) a *contextual preference model* that enhances preferences with contextual information, to capture the changing nature of preferences caused by context variation. The proposed approach integrates contextual preferences and contextual goals so that the combined influence of context variability and preference variability is realised in the derivation of requirements variants. The exercise of the approach helps designers identify requirements modelling errors and inconsistencies. Moreover, the thesis provides a complete transformation semantics of a goal model, including the contextual goals and contextual preferences, into disjunctive datalog (DLV) specifications. Such semantics are useful in the development of an automated support tool.

This contribution expands the realisation of adaptability to the early phase of the SC life cycle, from the stakeholders' level of requirements engineering, thus facilitating the problem-space-oriented approach to systems adaptability. The effectiveness of an SC adaptability design depends on how well the variability in the problem space is captured and explored.

3. **A service recommender that considers geographic location information in the derivation of preferences underlying service invocations**. The thesis proposes a geographic-aware collaborative filtering service recommendation approach that deals with implicit user-service invocation data. This approach concentrates on the mashup-API invocation scenario in which the user is a service composition (i.e., the mashup) invoking a Web service (i.e., the Application Programming Interface – API). The approach includes a method that maps geographical location information into geographical relevance scores. The geographical relevance, combined with functional relevance, is integrated into a matrix factorisation recommendation model, in which the model utilises both geographical and functional relevance to infer the preference degrees underlying the implicit mashup-API invocation data.

This contribution can be significant to the adaptability of service selection tasks in the SC life cycle. Adaptability is realised through a service recommendation that finds the most suitable component service for an SC at a particular context (i.e., considering the geographical locations of the composition and services).

## 1.6 Thesis Structure

The remainder of the thesis is structured as follows:

- **Chapter 2** provides an extensive survey of the state of the art. The beginning of the chapter provides a discussion about the underlying concepts and existing implementing technologies of service computing. Elaborate discussions regarding the conceptual perspectives of adaptability and the integration of adaptability with the SC life cycle are presented. This chapter also renders lists of existing works on various SC adaptability concerns that are defined in an adaptability taxonomy, as well as comprehensive surveys of literature relevant to quantifying adaptability, requirements variability, and service selection.
- **Chapter 3** presents the proposed approach on quantifying the adaptability of service compositions. The chapter's main focus is defining the adaptability metrics, as well as demonstrating and evaluating the use of the metrics. The chapter also includes discussions about the rationale of measurable dimensions of service composition adaptability, the specification constructs of BPEL, and the various adaptability mechanisms and frameworks that have extended BPEL for dynamic adaptability. This chapter is based on the works published in Botangen, Yu and Sheng (2017); and Botangen, Yu, Han, Sheng and Han (2020).
- **Chapter 4** presents the proposed approach for context-aware requirements variability modelling and analysis. This chapter provides definitions and specifications

for context, contextual goals, and contextual preferences. It also presents the utilisation of contextual goals and contextual preferences in the requirements variability analysis, and the evaluation of using the approach in actual requirements modelling. This chapter is based on the works published in Botangen, Yu, Yongchareon, Yang and Bai (2018); and Botangen, Yu, Yeap and Sheng (2020).

- **Chapter 5** presents the proposed approach for a geographic-aware service recommendation. This chapter provides discussions about the geographical and functional relevance modelling, the integration of geographical and functional relevance into the matrix factorisation model, and the extensive experiments undertaken to evaluate the proposed approach. The chapter also presents a preliminary study made on a real-world service invocation QoS dataset, to observe the impact of geographic locations on the QoS of service invocations. This chapter is based on the works published in Botangen, Yu, Yongchareon, Yang and Sheng (2019); and Botangen, Yu, Sheng, Han and Yongchareon (2020).
- **Chapter 6** concludes the thesis and presents challenges that will set directions for future work.

# Chapter 2

# Literature Review

Leveraging the Internet to facilitate a modern platform for openness, scalability, and heterogeneity has brought a dynamic environment that requires adaptability of the inhabitant software systems. In the past two decades, the adaptability of software systems has been massively studied in the software engineering community. Adaptability, particularly for service-based systems, has likewise attracted significant attention. Remarkable innovations have emerged through a myriad of adaptation mechanisms, techniques, and tools, such as: the design methodologies and implementation patterns for adaptability, runtime dynamism in services, automated server monitoring and repair of Internet-based systems, adaptive performance in the cloud computing platforms, middleware to support development of adaptive applications, and special-purpose micro-services for the deployment of functional enhancements. However, there is still much work left to be done, in addition to facing the numerous new challenges arising. The exceptional complexity of adaptability, the emergence of novel computing paradigms, and the continuously shifting landscape of technology use have been moulding software adaptability as an exciting area to work in.

This chapter presents the background concepts related to service computing and looks through the state of research and practice on the adaptability of service-based

systems. Section 2.1 elaborates on the fundamental elements of the service-oriented computing paradigm. Section 2.2 gives detailed discussions about the adaptability conceptual framework that is referenced in this thesis, and the various adaptability viewpoints that are relative offshoots of that framework. Likewise, the section presents existing works that are mapped to the widely-encompassing taxonomy of service composition adaptability concerns. Sections 2.3, 2.4, and 2.5 present existing approaches related to quantifying adaptability, requirements variability, and service selection, respectively. Finally, Section 2.6 provides a summary of the chapter.

## 2.1 Basic Concepts in Service Computing

This section discusses the underlying concepts and elements of service computing: software-as-a-service, the service-oriented architecture, the notion of services, and the typical Web service technologies.

### 2.1.1 The service-oriented computing paradigm

The service-oriented computing (SOC), alternatively termed as *service computing*, has emerged as a paradigm for a world of loosely-coupled cooperating services, to realise distributed solutions that transcend diverse organisations and technology platforms (Papazoglou, 2003; Sheng et al., 2014). It aims to transform computing entities (e.g., hardware and software assets) into *services*, in order to promote an open integration of functionalities. Such integration leads to the rapid, low-cost development and deployment of distributed software applications. SOC provides an abstraction layer that shifts the focus from infrastructure and operations, to services distributed on the Web (Bouguettaya et al., 2017). SOC is continuously attempting to establish comprehensive, interdisciplinary computational abstractions, software development architecture, approaches, tools, and technologies; all of which are geared towards a

service-centric digital society that deprioritises ownership in favour of paying only for what you need.

### 2.1.2 Software-as-a-service

SOC promotes the notion of offering software *as-a-service* to end-users. This concept of centrally managing and providing a standard application package dates back to the 1960s, when the aim was to overcome the increasing demand for IT applications and the shortage of IT personnel (Lee, Huynh, Kwok & Pi, 2003). At that time, organisations relied on contract programming, which became the main form of outsourcing. Starting in the 1990s, with the evolution of the Internet, software as-a-service was reinforced by the Application Service Provider (ASP) model (Papazoglou, 2003). The ASP is a third party (vendor) that owns a software application that is provided to end-users (clients). The vendor acts as a central provider that hosts, deploys, and manages an application in the vendor's own system while clients across the network remotely access the application on a subscription or rental basis. Basically, companies are able to outsource their information technology needs, while the ASPs operate and maintain the applications, the related infrastructure, and the clients' data. However, the inherent limitations of the ASP model resulted in applications with a monolithic architecture, and with highly fragile, customer-specific, and tightly-coupled components (Papazoglou, 2003). Several different acronyms referring to this and similar business models came out, such as AIP (Application Infrastructure Provider), IBS (Internet Business Service), BSP (Business Service Provider), and SSP (Solution Service Provider). In the early 2000, the term SaaS (Software-as-a-Service) was coined by the Software & Information Industry Association[1] with the aim to unify the various terminologies that were used to refer to software services. Some major IT vendors, such as Salesforce.com, Oracle,

[1] https://www.siia.net/

and SAP, have ventured to push, and still are pushing, the SaaS model. Meanwhile, the SOC paradigm has been revolutionising the SaaS towards a loosely-coupled interaction among applications over the Internet. It has expanded through delivering complex business processes and transactions as a service, while allowing applications to be created by re-using services anytime, anywhere, and by anyone.

### 2.1.3 Service-oriented architecture

Service-Oriented Architecture (SOA) (MacKenzie et al., 2006; Papazoglou & Van Den Heuvel, 2007) is the core implementing model of SOC to represent and manage distributed capabilities of software applications, resources, and infrastructure into a set of loosely-coupled, standard-based, interoperable, and location-transparent services. It allows multiple applications to be implemented in varying technologies, protocols, and platforms to communicate and interconnect, and these applications to become integration-ready services (Papazoglou, 2003). For instance, a client from any device, using any computing platform, any operating system, or any programming language, can use a SOA service to create a new application.

SOA enables a cost-effective design, development, and management of software systems by defining a logical way of providing services to implement functionalities of either end-user applications or other services (Papazoglou & Van Den Heuvel, 2007). It enables remote and distributed services to be discovered and consumed across the network through their published interfaces. Figure 2.1 shows a typical SOA, defining the roles of and interactions between the three main participants: the *service provider*, the *service registry*, and the *service client* (requester). The interactions involve the *publish*, *find*, and *bind* operations. The provider and the requester are both software agents representing entities (i.e., people or organisations), the former providing a service and latter requesting a service execution. A service provider hosts a service and publishes a

service description into a service registry or repository. The service client then seeks for a service description and uses it to bind with the service provider to invoke a service.

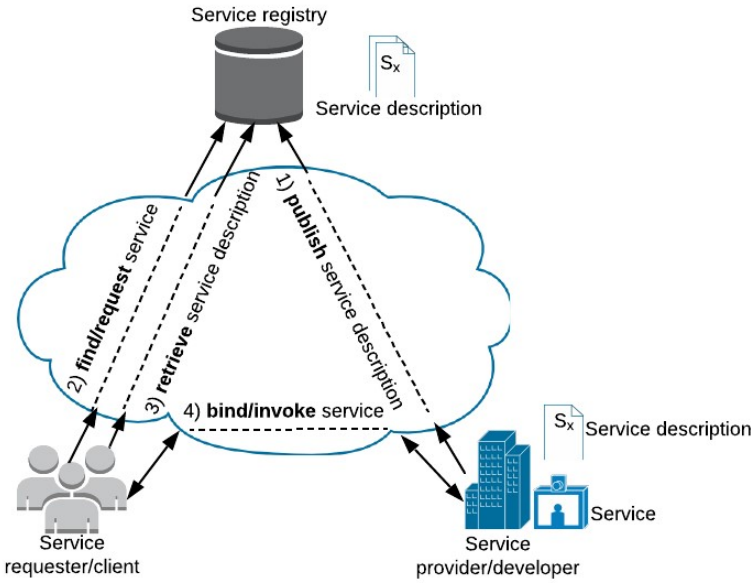



Figure 2.1: The fundamental service-oriented architecture

The SOA service registry defines the role of a service broker (Papazoglou & Van Den Heuvel, 2007). Service brokers are trusted software agents that drive service providers to comply with laws and regulations or industry best practices. A service broker maintains a list of available service providers and is able to "add value" to its registry by providing additional information about a service, such as trust and quality ratings. However, the basic SOA may not address some overarching concerns such as management, service composition, service transaction management and coordination, security, and other concerns that apply to the overall components of services architecture. Some early enhancements to the conventional SOA and implementing technologies were proposed.

- xSOA (extended SOA) (Papazoglou & Van Den Heuvel, 2007) defines a three-layered model that adds *service composition* and *service management* layers to

the foundation layer (i.e., the basic SOA construct). The service composition layer contains the roles and mechanisms for the integration of multiple services into a composite service. This layer defines the role of a service aggregator – a service provider that creates a value-added service from services that are provided by other service providers. Accordingly, a service aggregator acts as a service client at the same time as a service provider. The service management layer encompasses the operations management for supporting critical applications, enabling enterprises to manage the deployment and platform of services. This layer defines the role of a service operator that is responsible for performing the service operations management such as checking the correctness of service compositions. In addition, this layer aims to support the management of grid services and open service marketplaces, and defines the role of a market maker that is necessary to create and maintain the marketplace.

- ESB (Enterprise Service Bus) (Schmidt, Hutchison, Lambros & Phippen, 2005) is an open standards-based inter-operability message bus for the implementation, deployment, and management of SOA-based solutions, particularly between large-grained heterogeneous enterprise systems.
- OSGA (Open Grid Services Architecture) (Foster, Kesselman, Nick & Tuecke, 2002) aims to combine the advantages of a platform-independent description, discovery, and invocation of web services with the dynamic discovery and efficient use of distributed computational resources of grid services.

### 2.1.4 Services

A service is generally understood as the capability to perform work for another, or the offer to perform work for another (MacKenzie et al., 2006). It is a non-material equivalent of a good. A similar notion is applied to the ‘service’ in SOA – a "semantically

well-defined abstraction of a set of computational or physical activities involving a number of resources, intended to fulfil a client need or a business requirement" (Sheng et al., 2014). A SOA service is typically a software that encapsulates a complete business functionality that can be invoked to fulfil a needed functionality by another software system. In SOA, all business functions (i.e., from simple requests to complex business processes), are exposed as software resources which are packaged as services with the following characteristics (Papazoglou & Van Den Heuvel, 2007; Channabasavaiah, Holley & Tuggle, 2004; Papazoglou, 2003):

- Self-contained – a service is a well-defined module with its own state that is independent of the state and context of other services. Likewise, a service is designed irrespective of the kind, the purpose, and the context of use in the client's system.
- Platform-independent – a service is technology-neutral. It can be used regardless of the technological platform of the client. Hence, protocols, descriptions, and discovery mechanisms should comply with widely accepted standards.
- Loosely-coupled – a service does not need to know the implementation details of the client. It can be (re)used by different clients for different purposes. Loose-coupling has three basic aspects (Pautasso, Zimmermann & Leymann, 2008): *time/availability aspect* is the ability of service consumers to interact with a service provider even when the latter is not available, *location aspect* allows clients to discover the actual location of service providers at runtime (dynamic late binding), and *evolution aspect* is the ability to make modifications to a service without affecting its clients.
- Autonomous – a service is perceived as a blackbox by external entities. For instance, a client merely anticipates the expected result of invoking a service, but it is unnecessary to know or care how the service performs its function.

- Transparent – a service description is accessible from a known repository so that clients, regardless of their location, can locate and use the services.
- Invocable – in general sense, a service (interface) can be invoked regardless of: whether it is local or remote, the interconnect scheme or protocol of invocation, or the infrastructure components needed for the connection.

A fundamental concept within the loose-coupling principle of SOA is for a service to be used by any client by relying on the prescribed service interface specified by the service description, without needing to know the service implementation details (Papazoglou & Van Den Heuvel, 2007). Hence, SOA maintains two separate but interrelated artefacts of a service:

- Service description – describes the service capability, interface, behaviour, and quality, all of which provide the necessary information for the discovery, selection, binding, and composition of services. It includes information needed to interact with the service such as service inputs, outputs, and associated semantics. Likewise, it describes what is accomplished when the service is used and the conditions of use.
- Service implementation – defines the implementing technology and execution details to fulfil the functionality of a service.

Figure 2.2 presents the evolution of computing value chain, in which services are presently considered the highest form of abstraction in the value chain (Bouguettaya et al., 2017). At the outset of computing, the focus was to represent information into machine-readable format as bits and bytes, called data. Later on, the need for complementing data with meaning led to the challenge of transforming data into information. Further advances in computing have provided opportunities to reason about information, prompting the formulation of knowledge. Then recently, the challenge of responding to

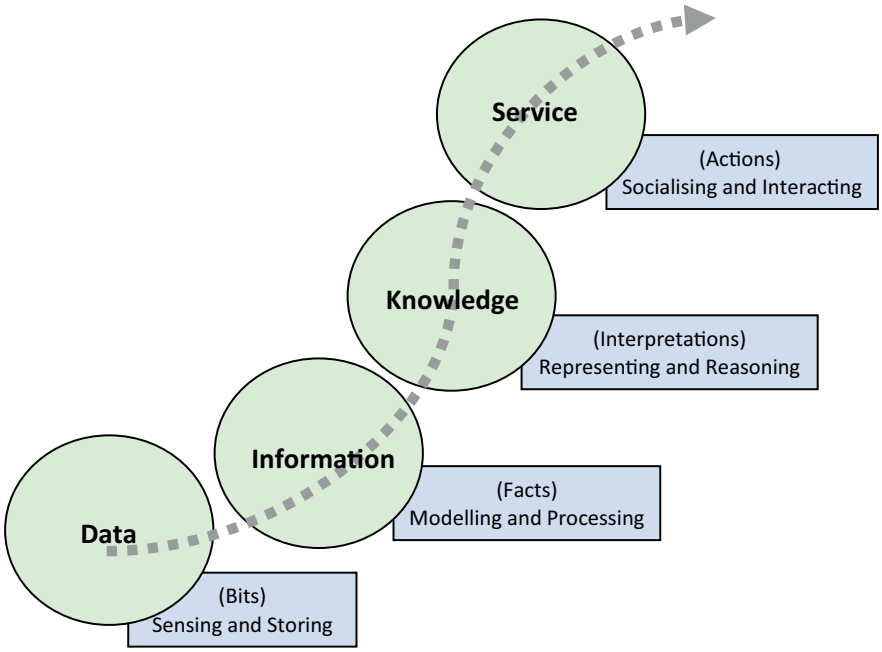



Figure 2.2: Abstractions along the evolution of computing value chain

such knowledge has resulted in services. The abstract definition of a service comes to fulfil the need to act and deliver on the knowledge, thus making the knowledge useful.

### 2.1.5 Web services

The Web services (WS) technologies and standards, have become a major enabler for the SOA. Web services have, so far, been the key technology to implementing typical SOA objectives, such as service sharing, just-in-time or on-demand integration, and interoperability among new and legacy software resources (Bouguettaya et al., 2017; Sheng et al., 2014; Papazoglou & Van Den Heuvel, 2007). To facilitate smooth interoperations, standardisation bodies such as the World Wide Web Consortium (W3C) and the Organisation for the Advancement of Structured Information Standards (OASIS), have led specification and standardisation efforts for implementing Web services.

Several definitions have been coined for the term *Web service* in the literature, ranging from the generic to the more specific. A very loose definition is “any unit of

business, application, or system functionality accessible over the Web" (Sheng et al., 2014; Manes, 2001), which means that anything that has a Uniform Resource Locator (URL) is a Web service. Some technical and detailed definitions are the following:

- Definition by the Dagstuhl SOC working group (Ludwig & Petrie, 2006): "a possibly remote procedure with an invocation that is described in a standard (preferably XML-based) machine-readable syntax reachable via standard Internet protocols with a description, including at a minimum the allowed input and output messages, as well as a possible semantic annotation of the service function and data meaning".
- Definition by W3C (World Wide Web Consortium)[2]: "is a software system identified by a Universal Resource Identifier (URI) designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialisation in conjunction with other Web-related standards".

Both definitions are geared towards the representation and integration of services through the use of standard Internet protocols and Web service technologies. The typical standard Web services interaction protocols and technologies include the following:

- SOAP[3] (Simple Object Access Protocol) – is an XML-based messaging protocol that allows the exchange of information between peers in a decentralised, distributed environment via Hypertext Transfer Protocol (HTTP) and Remote Procedure Call (RPC).
- REST (Representational State Transfer) (Fielding, 2000) – is an architectural style

[2]www.w3.org/TR/ws-arch/
[3]https://www.w3.org/TR/soap/

for building large-scale distributed hypermedia systems, exposing data, resources, and functionality through Web services identified by URIs.

- OSGi[4] (Open Services Gateway Initiative) – is a specification that defines a common, open, and dynamic architecture to develop, deploy, and manage services for Java.
- WSDL[5] (Web Services Description Language) – is an XML-based specification for a Web service description. It describes the operations in a Web service, messages exchanged by the operations, the parts that make up the message, and the protocol bindings.
- UDDI[6] (Universal Description Discovery and Integration) – is a universal business registry standard for indexing Web services, where service providers can register their service descriptions so that services can be located by developers and applications.
- ebXML[7] (Electronic Business XML) – is an initiative for a B2B XML framework that enables the collaboration of Web-based business services.

There are two dominant types of Web services based on the choice of technology used in its development (Pautasso et al., 2008; Sheng et al., 2014). The first is the *SOAP-based Web services*, also called "Big" or WS-* Web services, which rely on the three standardisation initiatives: SOAP, WSDL, and UDDI. Service registration, discovery, and invocation are represented in a single standardised messaging format: SOAP. SOAP-based Web services are stateful – demanding more computation resources, and transport-independent – allowing SOAP messages to be exchanged in a variety of transport protocols. The second type is the *RESTful Web services* which leverage the URI standard as the naming mechanism to address resources. URIs encapsulate

[4]https://www.osgi.org/developer/architecture/
[5]https://www.w3.org/TR/wsdl
[6]http://www.uddi.org/pubs/uddi_v3.htm
[7]http://www.ebxml.org/

all information required to identify and locate a resource on a global addressing space, without a need for a centralised registry. RESTful Web services interact using the Web's HTTP protocol, which comprises a fixed set of operations: GET, PUT, DELETE, and POST. RESTful Web services are *lightweight* – enabling the use of smaller message formats rather than XML, and *stateless*, both are suitable for tactical, ad hoc integration over the Web. An attempt towards this easy-to-accomplish end-user Web services integration to create situational Web applications is services mashups (Benslimane, Dustdar & Sheth, 2008). The SOAP-based Web services are preferred in enterprise application integration scenarios with a longer lifespan and advanced QoS requirements (Pautasso et al., 2008).

Web services espouse four SOC principles (Petrie, 2016). A web service can be consumed – by anyone without requiring any change to the service (the *democratic principle*), and at anytime without prior arrangement (the *just-in-time principle*). Moreover, software system robustness is achieved by – the runtime choice among competing services (the *free market principle*), and the ability to automatically construct and adjust processes to requirements in order to achieve goals (the *adaptability principle*).

In terms of complexity, Web services are distinguished between *atomic* and *composite* (Sheng et al., 2014). An atomic Web service, also called simple or elementary, does not rely on other Web services to fulfil a user's request. Monolithic legacy applications can use wrappers or adapters to be exposed as atomic Web services. A composite service is the aggregation of atomic and other composite services to implement a set of operations to achieve a business function. With regards to functionality, a Web service can be a *self-contained business task* (e.g., checking account balance), a *complete business process* (e.g., travel booking process), an *application* (e.g., online store), or a *service-enabled resource* (e.g., access to a sensor data). Accordingly, Web services can perform simple tasks to the more complex business functions.

Web services have continuously established an essential role in the development

of modern service-based distributed software systems (Bouguettaya, Sheng & Daniel, 2014a), expanding from the conventional WS, to unleashing the data and function of real-world things as both provider and consumer of services in the Internet (G. Huang, He & Zhang, 2014; Mathew, Atif, Sheng & Maamar, 2013; Christophe, Boussard, Lu, Pastor & Toubiana, 2011). Technology giants such as Amazon, Microsoft, IBM, Google, and Facebook have been providing utility to their resources through Web services, enabling third parties simple access and (re)use. Expanding the existing Web service standards and technologies is an ongoing objective in service computing. That will support the computing needs of key emerging areas such as cloud computing (i.e., including other technologies sharing similar aspects such as grid computing, utility computing, and virtualisation) (Q. Zhang, Cheng & Boutaba, 2010; Wei & Blake, 2010; Tsai et al., 2010), Web of things (Guinard et al., 2010), Internet of Things (Miorandi et al., 2012), mobile computing (Dinh, Lee, Niyato & Wang, 2013), and social computing.

### 2.1.6 Discussion

Service computing, a paradigm of service-oriented abstractions and on-demand interactions, has certainly brought a new perspective on the engineering of software systems. With the technology of Web services and APIs (i.e., Application Programming Interface, a Web API or simply API is basically a counterpart for a Web service), "anything" can be inevitably wrapped into a service. Services are becoming ubiquitous in today's everyday living environment. Most of the traditionally provided (i.e., brick and mortar) societal and economic services, like healthcare, finance, governance, tourism, education, and entertainment, can now be offered as online services. These advances in online service technologies have been transforming the Internet into a global workplace, a means to manage one's individual and social affairs, an entertainment platform, and a business platform for services (Bouguettaya et al., 2017). In most cases, a single service is not

sufficient to perform the functionality requested by a user. Hence, multiple services should be composed to achieve the ultimate user goal. The composition of services (i.e., *service composition*) has become a challenge in both service computing and software engineering communities. Service composition has then become a thriving area of research since it introduces new perspectives and requirements in the development of software systems (Sheng et al., 2014).

Service-based systems are mainly distributed in nature, and they are expected to operate in a highly dynamic environment. In this situation, adaptability becomes a necessary requirement. For instance, a new (better) component service may be available anytime, while an existing component could become temporarily unavailable and should be replaced; or the functionality or quality of service (QoS) of a partner service may change. A composition system should support dynamic service binding or an automatic replacement of undesirable component services at runtime. Moreover, user and business environments constantly change. Therefore, a composition system should facilitate the means of how compositions can dynamically adjust their behaviours to meet new user requirements, as well as to cope with new business policies.

## 2.2 The Notion for Adaptability in Service Composition

A software system that adapts has been a long-standing idea, as old as the notion of computing. Theories, laws, and hypotheses regarding software evolution have been formulated as early as the 1970s, for actively used software systems to continuously change to satisfy its stakeholders (Lehman, 1980). In an early work by Boyle (1979), he presented techniques for dynamic software adaptation and transformation, and defined three software adaptation classifications: adaptation to the hardware environment,

adaptation to the software environment, and adaptation to the problem environment. However, in those early days of computing, creating a software that adapts was not a priority, because traditional software systems were typically run with fixed requirements in stable environments. Adaptability had been barely significant, and it did not justify the inherent cost and complexity of its implementation.

Later on, revolutions in the computing field, such as the emergence of ubiquitous computing (Weiser, 1993, 1999) and autonomic computing (Kephart & Chess, 2003), have stimulated the resurgence of the need for software systems that adapt to their environment. Ubiquitous computing aims to integrate the (invisible) availability of computers throughout the physical environment, removing traditional boundaries for how, when, and where humans and computers interact, while autonomic computing focuses on developing systems that can automatically manage themselves given only high-level human guidance. The dynamic nature of today's operational environment and the opportunities brought by various computing developments have driven immense interest in software system adaptability.

Two general approaches for implementing software systems adaptability is defined by McKinley, Sadjadi, Kasten and Cheng (2004): *parameter adaptation* and *compositional adaptation*. Parameter adaptation modifies program variables that determine behaviour. However, it does not allow new algorithms and components to be added to an application after the original design and construction. In contrast, compositional adaptation allows changes in the algorithmic or structural components to improve an application's fit to its current environment. In addition to tuning program variables, compositional adaptation allows dynamic recomposition of the application during execution; e.g., switching system components, or adding new behaviour to deployed systems.

To augment the aforementioned McKinley et al.'s view on adaptability (2004), this section provides a detailed discussion about various conceptual perspectives of integrating adaptability in the service composition process. Likewise, existing works

about the adaptation of service-based systems are also presented and classified according to their specific adaptation concerns.

### 2.2.1 Defining adaptation and adaptability

The concept of *adaptation* originated from the natural sciences and has been adopted in the computing disciplines. Generally, adaptation defines a process of maintaining the relationship between an entity and its environment, particularly to deal with changes and adjustments. A variant concept – *adaptability* – is the capability to enact adaptation. Merriam-Webster[8] defines adaptation as "the process of changing to fit some purpose or situation" or "the process of adjustment to environmental conditions". Meanwhile, adaptation has been used and defined by researchers and organisations in various computing domains, such as:

- Definition by ISO 9241-210 (Human-centred design for interactive systems)[9]: "a process in which an interactive system adapts its behaviour to individual users based on information acquired about its user(s) and its environment".
- Definition by ISO/IEC 25000: "a software product is adapted for different specified environments without applying actions or means other than those provided for this purpose for the software considered".
- Definition by IEEE[10]: "a process by which a system or component can be modified for use in applications or environments other than those for which it was specifically designed".
- Definition by Kell (2008): "any process which modifies or extends the implementation or behaviour of a subsystem to enable or improve its interactions with its environment".

[8] http://www.merriam-webster.com/dictionary/
[9] http://www.iso.org/
[10] IEEE Standard Glossary of Software Engineering Terminology

- Definition by Subramanian and Chung (2001b): "a change in the system to accommodate a change in its environment; it is caused by a change from an old environment to a new environment, and results in a new system that ideally meets the needs of its new environment".

### 2.2.2 A conceptual framework for adaptability

The commonly used conceptual framework for the adaptability of software systems originated from the "figure 8" model proposed by Oreizy et al. (1999), which is a model for an architecture-based adaptation. The model recognises the dichotomy and tight interconnection between the software system and its architecture (Medvidovic, 2017). The model has two major components: the *adaptation management* which was intended to capture the lifecycle of adaptive software systems, and the *evolution management* which was intended to capture the software mechanisms employed to adapt the system.

Subsequent software adaptability models have adopted and expounded Oreizy et al.'s model (1999), such as in the adaptability framework for service-based systems described in the deliverables of the S-Cube[11] project. S-Cube is a European research consortium for service-based software systems. The S-Cube framework defines a set of high-level tasks for the adaptation process: *i*) collect relevant information through the monitoring mechanisms; *ii*) recognise critical events from the monitored information; *iii*) determine the need for adaptation based on the adaptation requirements; *iv*) identify and select an appropriate strategy to perform adaptation; and *v*) explore available adaptation mechanisms to implement the strategy. Adaptability strongly relies on the presence of monitoring mechanisms and facilities. The monitoring process allows one to identify, detect, and even foresee the critical events and situations occurring in a composition's environment that would require changes in the composition aspects such

[11] http://www.s-cube-network.eu

as configuration, behaviour, and presentation.

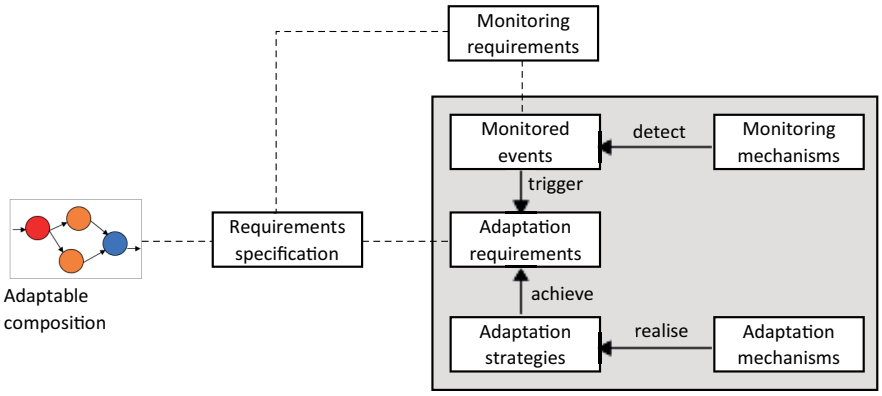



Figure 2.3: The key elements for adaptation and monitoring

The S-Cube adaptability framework exploits an adaptation-by-monitoring process which describes the standard sensing-planning-actuating control chain (Nagrath, 2006). This process is also shared by autonomic computing where the adaptation activities are described by a MAPE-K cycle (Monitor, Analyse, Plan, Execute) (Rutten, Marchand & Simon, 2017). Considering the aforementioned adaptation and monitoring tasks, the framework comprises five key elements (Kazhamiakin et al., 2010): *monitoring mechanisms*, *monitored events*, *adaptation requirements*, *adaptation strategies*, and *adaptation mechanisms* (Kazhamiakin et al., 2010). Figure 2.3 shows the relationships between the elements. An adaptable composite service needs to satisfy the *requirements specification* that also defines the adaptation and monitoring requirements. *Monitoring requirements* capture the need for detecting those situations that would require adaptation. From these monitoring requirements, designers derive the properties to be observed at runtime. A monitoring engine usually performs the monitoring task, emitting the *monitored events* (e.g., the relevant information about the service execution, users, and context). *Adaptation requirements* represent the need to change the composition behaviour in response to a monitored event. *Adaptation strategies* define the possible ways to achieve those adaptation requirements in a given situation, and these strategies are realised by the *adaptation mechanisms*. The adaptation mechanisms

Table 2.1: Examples of the adaptation and monitoring tasks

| Adaptation-by-monitoring framework | | General examples |
|---|---|---|
| Process | Key elements | |
| Monitoring process | Monitoring mechanism (the *How?* of the monitoring process) | Runtime monitoring tools, QoS monitoring, process/data mining, online testing/analysis, post-mortem analysis, service composition verification/validation, log analysis, context monitoring |
| | Monitored event (the *What?* of the monitoring process) | Faults/failures, deviations of QoS parameters, changes in business requirements/rules, violations of SLAs, change in context |
| Adaptation process | Adaptation requirements (the *What?* of the adaptation process) | Identifying the aspects of the composition that are subject to change and specifying the expected outcome of adaptation like optimising QoS, completing failed actions, avoiding unavailable service, and correcting behaviour |
| | Adaptation strategies (the *How?* of the adaptation process) | Replacement of services, change providers, re-compose workflow, compensate a previous action, re-configure, re-execute, re-plan, re-bind |
| | Adaptation mechanism (the *How?* of the adaptation process) | Dynamic service discovery/binding, context-aware service selection, automatic/manual (re)composition, application re-design/re-engineering, tools/frameworks to execute adaptation strategies |

are techniques and facilities that can be provided by the underlying composition, or by the operation and management platform in different functional layers (see Section 2.2.8 on page 73). Table 2.1 presents some examples of each key element.

Being represented at high levels of abstraction, the key elements of the adaptability framework exemplify overarching concepts, allowing the framework to accommodate the integration of a wide range of adaptability mechanisms, techniques, and methodologies for realising the adaptability of composite services.

### 2.2.3 Adaptation strategies and triggers

The adaptation strategies are distinguished by Bucchiarone et al. (2009) into two types: the *domain-independent*, which are applicable to any application context, and the *domain-dependent*, which are limited to specific execution environments. Likewise, there are common adaptation strategies used in the compositions (see Table 2.2).

Table 2.2: The typical domain-independent adaptation strategies
*Source: Adapted from Bucchiarone et al., 2009, Table 1, pp. 471*

| Adaptation strategies | Description |
|---|---|
| Substitution | Reconfiguring a composition by changing a component service with an alternative one. |
| Re-execution | Execution goes back to a safe point and redo the same set of tasks or perform an alternative path. |
| (Re)-negotiation | It can be a simple termination of the service used on the requester side to re-negotiate SLA properties (e.g., QoS), or complex management of reconfiguration activities on the provider side. |
| (Re)-composition | Re-organisation and re-arrangement of the control flow of components of the composition. |
| Compensation | Ability to undo the effects of a process that fails to complete. |
| Trigger evolution | Insertion of workflow exception, able to activate the application evolution. This applies when a change is needed for the whole service composition model. |
| Log/update adaptation information | Storage of all information regarding adaptation activities for various purposes (e.g., service reputation, QoS analysis, adaptation result) |
| Fail | The system reacts to changes by storing the system status and causing the failure of the service and re-executing it. |

Adaptation triggers (i.e., the monitored events) initiate the adaptation process. Typical triggers are concerned with the component services, which can be the changes

in the service functionality or service quality. Changes in service functionality include: variation of service interface (e.g., signatures, data types, and semantics), variation of service interaction protocol (e.g., messaging), and failures. Changes in service quality include: decreasing service reputation, degrading QoS parameters, and violation of SLA.

Other major activators for adaptation are contextual triggers. Considering the increasingly dynamic settings of both compositions and users, the role of *contexts* in the adaptability framework is becoming much more essential. Contextual triggers can be classified as: changes in the *business context*, such as new business regulations and rules, changes in the *computational context*, such as variations in devices, protocols, and networks, and changes in *user context*, such as different user groups and profiles, user preferences, various social environment and physical settings (e.g., time and location), and different user activities. With applications now often designed to target individual users, compositions should be able to capture and correctly exploit user context, in order to initiate suitable service customisation and personalisation. These contextual triggers are sometimes interrelated. For instance, a client travels to a new location (i.e., change in user context), there may be new services available (i.e., change in business context), and there may be a change in computational context (e.g., bandwidth).

Figure 2.4 shows a mapping of the triggers to a set of applicable strategies, leading to a question of which strategy is the most suitable. A multi-criteria decision making can be involved in the process, such as considering the scope of the change (i.e., whether the adaptation affects only a composition instance or the entire composition model), the impact of the change (i.e., the possibility of the composition to fulfil its task), the cost of adaptation, autonomy (i.e., whether human intervention is not needed), and performance (e.g., how fast an adaptation strategy is). Choosing the best adaptation strategy depends on these criteria. For instance, the trigger evolution strategy applies when the scope of adaptation concerns the whole application model. Likewise, when considering the

impact of change, re-execution and substitution may apply when the task on hand can still be accomplished. Otherwise, compensation, fail, or trigger evolution strategies are used when it is impossible to complete the task.

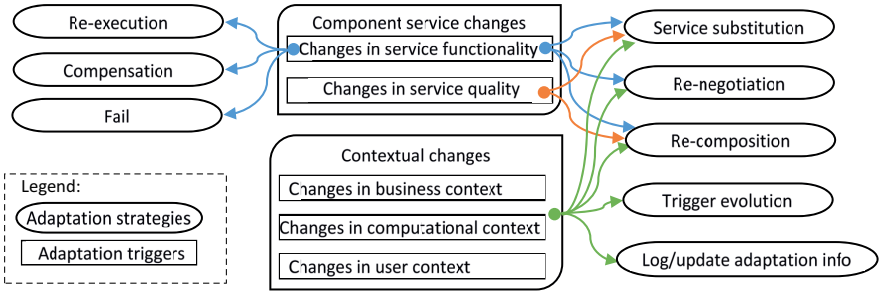



Figure 2.4: A mapping of adaptation triggers to the adaptation strategies

The S-Cube deliverable CD-JRA-1.1.4 presents three ways of defining triggers and strategies for adaptable service compositions: *i*) hard-coding the corresponding elements directly in the main logic of the composition, *ii*) hard-coding the elements in the composition infrastructure, *iii*) providing design patterns and tools for flexible and transparent modelling of the triggers, strategies, and their relationships. The first approach does not need any specific tool and mechanism on the design and execution infrastructure. However, this overloads the composition logic making it error-prone and difficult to maintain, and requires ad-hoc and non-reusable solutions when not supported by the composition language. For the second approach, although it follows the separation of concerns principle, there is no flexibility to deal with various adaptation needs or application domains. The third approach enables designers to focus solely on the adaptation aspect which would allow for the flexibility and reuse of the adaptation mechanisms. This approach drives the formulation of principles and guidelines for: *modelling adaptation triggers* (i.e., both the situation when the adaptation is needed and the specific adaptation need), *realising adaptation strategies* (i.e., includes modelling strategies, their properties, and their combination, and relating them to the underlying mechanisms and runtime infrastructure), and *associating adaptation strategies to*

*triggers*.

### 2.2.4 Adaptability specification

Adaptability specification represents the notations needed to specify the adaptation strategies and mechanisms (Kazhamiakin et al., 2010). It covers procedural approaches (i.e., specifying concrete actions to perform), declarative approaches (i.e., specifying a goal to be achieved), or hybrid approaches. From the design perspective, adaptability specification is crucial to building adaptable service compositions, and it may be defined *implicitly* or *explicitly* (see example approaches in Table 2.3 on the next page). In the case of implicit specification, the designer does not influence the adaptation process; the system decides and executes hard-coded (i.e., based on some predefined schemata by the adaptability framework) adaptation strategies and actions, which cannot be changed without modification of the adaptation mechanism. This situation can be observed from the dynamic service compositions where services are automatically selected and composed at runtime, or the case of self-healing systems where the recovery activities are hard-coded. Adaptation design activities, in the case of implicit specification approaches, aim to provide a rich schema and more complete descriptions of the services, in order to support and simplify automated runtime decisions.

In contrast, in the explicit specification, the designer guides or controls the adaptation process by explicitly providing adaptation instructions or requirements. Kazhamiakin et al. (2010) have introduced the following forms of the explicit specification:

- *Action-based* — defines situation-action rules which specify concrete actions to perform in certain situations or upon the occurrence of certain events such as a service failure or context change. This approach is typically based on procedural notations and languages. The situation part is associated with the variations, while the action part specifies concrete adaptation actions. These actions include

Table 2.3: Representative approaches for the various forms of adaptability specifications

| Approach | General description | Main purpose | Adaptability specification |
|---|---|---|---|
| Zeng et al. (2004), Canfora et al. (2005) | Dynamic service composition techniques based on multi-dimensional optimisation of QoS | Optimisation | Implicit |
| Bianculli et al. (2007), Mosincat and Binder (2011) | Dynamic maintenance techniques for performance degradation detection, diagnosis, and repair of composite services (i.e., based on monitored functional and non-functional properties) | Correction | Implicit |
| Friese et al. (2005), Dai et al. (2009) | Approaches for self-healing processes that support dynamic replacement of failed component services | Correction | Implicit |
| Benatallah et al. (2005), Brogi and Popescu (2006), Motahari Nezhad et al. (2007) | Approaches for dynamically resolving behavioural mismatches between the provided and required functionalities of partner services (i.e., through generating adapters for service integration) | Customisation | Implicit |
| Verma et al. (2005), Chafle et al. (2006), Y. Wu and Doshi (2007) | Adaptability approaches that enable the designer to define an abstract workflow model and a set of specific requirements that constrain the functional and non-functional properties | Optimisation, Correction | Goal-based |
| Lazovik et al. (2004), Ardagna and Pernici (2007) | Approaches where business processes accomplishing certain goals are predefined, while component services may vary according to user requirements and constraints | Customisation, Correction | Goal-based |
| Erradi et al. (2006), Vasilecas et al. (2016) | Utilise policies (i.e., event-condition-action rules) for handling specific adaptations, where action part of the policy represents the adaptation instructions which are enforced at run-time | Customisation, Correction | Action-based |
| Baresi et al. (2007), Boukhebouze et al. (2009) | Utilise rules for self-healing adaptation | Correction | Action-based |
| Siljee et al. (2005), Alférez and Pelechano (2017) | Adaptability is based on the explicitly represented variants of the composition | Correction , Customisation | Explicit variability |
| S. H. Chang and Kim (2007), Hallerbach et al. (2010) | Manage adaptability through variants that are selected and filtered according to contextual information | Customisation | Explicit variability |

management actions (e.g., notify and log) and adaptation strategies (e.g., re-execution of a service invocation, re-binding or substitution of a service, re-negotiation of Service Level Agreements (SLA), roll-back to previous stable execution point, re-composition of services, and halt). With the pre-defined adaptation strategies and actions, additional reasoning and decision-making are not required at runtime, which makes action-based approaches considerably more efficient. However, there is less flexibility since the adaptation specification defines restricted sets of scenarios.

- *Goal-based* — defines the adaptation activities in a high-level form of a behavioural specification to meet certain objectives (i.e., declaratively defining a goal), leaving the system or the middleware to determine the actions required to achieve those objectives. Although the main adaptation goal is often predefined, and the decisions and choice of actions are left to the platform, the designer may set additional requirements and constraints to guide the adaptation process.
- *Explicit variability* — specifies execution points in the application where the adaptation should take place (i.e., variation points) and associates to these points a set of alternative behaviours (i.e., variants) that may be used in certain cases. Since the variation point is explicitly defined, there is a lesser need for monitoring specific conditions or assertions. Instead, monitoring should be used to obtain relevant context information for use in selecting variants. Hence, the specification of context and contextual conditions is important in this approach.

### 2.2.5 Adaptability based on dynamicity

Three adaptability approaches are defined based on the dynamicity of specifying adaptation needs (Bucchiarone et al., 2009): *i*) *built-in adaptability*, *ii*) *abstraction-based adaptability*, and *iii*) *dynamic adaptability*. Built-in adaptability specifies at design time

the situations on which adaptation is triggered, and the concrete actions to be executed. Possible adaptation needs and configurations are fixed and known *a priori*. Typical of this approach are specifications that extend standard service composition notations (e.g., Business Process Execution Language (BPEL)) with adaptation-specific mechanisms using ECA-like (event-condition-action) rules (Colombo, Di Nitto & Mauri, 2006), variability modelling (Koning, Sun, Sinnema & Avgeriou, 2009; Sun, Rossing, Sinnema, Bulanov & Aiello, 2010), or aspect-oriented methods (Charfi & Mezini, 2007; Kongdenfha, Saint-Paul, Benatallah & Casati, 2006). The strategies suitable for this approach are service substitution (i.e., choosing from a pre-defined list of alternative services), re-execution, compensation, re-composition (i.e., using pre-defined variants), and fail.

Abstraction-based adaptability fits when adaptation needs are fixed but possible configurations to trigger adaptation are not known. Hence, concrete adaptation actions cannot be completely defined at design time. For example, an abstract service composition model, together with an abstract adaptation strategy, is defined at design time but the concrete services are discovered and bound at runtime based on the execution context (Verma et al., 2005; Alférez, Pelechano, Mazo, Salinesi & Diaz, 2014). Likewise, a goal or utility function can be specified at design time, then realised by service re-composition at runtime based on the execution environment and available services (Zeng, Benatallah, Dumas, Kalagnanam & Sheng, 2003). The design of the abstract models (i.e., including adaptation options), to specify abstract process models or composition goals, are important in this approach. Strategies that may be used are service substitution (i.e., through dynamic discovery), re-composition (i.e., based on predefined goal/utility function), and re-negotiation.

Dynamic adaptability is suitable when adaptation needs that may occur at runtime are not known during design time (Bucchiarone, Marconi, Mezzina, Pistore & Raik, 2013). At runtime, there have to be mechanisms that *i*) select and instantiate suitable

adaptation strategies according to specific triggers and concrete situations, *ii*) define concrete actions and parameters of those strategies, and *iii*) execute the strategies using the appropriate mechanisms. This adaptability approach is needed in situations such as: modifications or corrections of business process instances (e.g., ad-hoc actions or changes performed by business analysts), and changes in the user activities that require either modifying the composition or creating new ones. It can be built on top of the other adaptability approaches, but the focus is to define mechanisms to fulfil appropriate adaptation strategies and actions, to realise adaptation needs at runtime. The adaptation strategies (e.g., re-composition, service substitution, compensation, re-execution, evolution, and fail) still apply. However, there may be variations in the realisation of these strategies such as requiring active user involvement in decision making or in performing ad-hoc changes.

### 2.2.6 Adaptability in the service composition life cycle

Integrating the adaptation life cycle with the service composition (SC) life cycle creates two complementary views for the SC adaptability life cycle activities: *design-time* and *runtime* activities (Bucchiarone et al., 2009; Andrikopoulos et al., 2008). The two views co-exist and augment each other as counterparts during the lifetime of a composition (i.e., the design-time adaptation activities need to be carried out to facilitate runtime adaptation). Alternatively, Bucchiarone et al. (2009) refer to the runtime and design-time activities as *adaptation* and *evolution*, respectively. On the one hand, the runtime adaptation activities refer to the typical *adaptation* that involves some temporary modifications in response to changes in requirements or service composition context, or to faulty situations during the execution phase. Typical examples of these runtime adaptations include re-executing an unavailable service or replacing a poorly-performing service. On the other hand, the design-time adaptation activities, which are

associated with the design-time phases of service composition, allow the *evolution* of a composition. Evolution describes the introduction of changes that could require the re-design or re-engineering of the composition, modifying it permanently. Evolution becomes reasonable when a faulty situation that requires adaptation is recurrent. A permanent modification of the fault-prone composition logic could be more efficient than a frequent enactment of adaptation strategies.

Table 2.4 on the following page presents the SC adaptability life cycle framework that maps the adaptation activities into the main phases of the SC life cycle. The corresponding artefacts, either exploited or generated within each phase, are also presented. It should be noted that the adaptation activities defined within the SC life cycle phases are very high-level tasks that can encompass a variety of specific adaptability techniques and approaches. The requirements engineering (RE) & design phase is extended with (*i*) *eliciting adaptation and monitoring requirements* using any formal RE method or requirements elicitation technique. This initiates (*ii*) *specifying adaptation and monitoring models* together with the requirements models. Based on the composition requirements (e.g., functional or non-functional), it is important to define the adaptation goals, which should be satisfied when certain changes happen with respect to the expected service composition state, functionality, or context. Likewise, the monitoring requirements define what should be continuously observed and the need for detecting those events that may trigger the adaptation of a composition. From the monitoring requirements, the properties to be monitored are identified. An abstract composite service model and the adaptability requirements model are finally generated at the RE & design phase. Both models define the underlying design of the composition.

The means to achieve, operate, and manage adaptation at runtime must be taken into account during design decisions, and must therefore be explicitly described in the RE & design phase. Measurable requirements and design properties usually get involved in such design decisions, where designers perform trade-offs between adaptability

Table 2.4: Adaptation activities in the service composition life cycle

| Adaptation view | Service composition life cycle phase | Adaptation activities | Artefacts |
|---|---|---|---|
| Design-time adaptation | Requirements engineering & design | i) eliciting adaptation and monitoring requirements<br><br>ii) specifying adaptation and monitoring models | abstract composite service, adaptive requirements model, monitoring and adaptation architecture |
| | Construction | iii) construction of adaptation and monitoring mechanisms<br><br>iv) static adaptability analysis | constructed composite service, adaptation mechanisms (realisation mechanisms), monitoring mechanisms, decision mechanisms |
| | Deployment | v) deployment of adaptation and monitoring mechanisms | executable composite service |
| Runtime adaptation | Execution | vi) runtime monitoring (e.g., monitoring analysis)<br><br>vii) identify the adaptation needs (e.g., dynamic adaptability analysis)<br><br>viii) identify adaptation strategies<br><br>ix) enactment of the adaptation strategy | composite service instance, adaptation strategy, monitored properties |

and different design properties (Perez-Palacin, Mirandola & Merseguer, 2014). Each property has its respective metrics that can be used to specify concrete target values. These concrete values facilitate how designers ensure the satisfaction of requirements in the resulting composition design.

The construction phase integrates the (*iii*) *construction of adaptation and monitoring mechanisms*, to extend a typically constructed service composition. The mechanisms may be integrated directly into the composition code, or as a separate entity within the underlying execution platform. These mechanisms include monitoring frameworks, tools for performing actual adaptation actions (i.e., realisation mechanisms), and tools for important decision making regarding adaptation (i.e., decision mechanisms). In addition, the (*iv*) *static adaptability analysis*, which can be pure architectural level decision making, includes comparing the measurable adaptability requirements against some defined target values. For instance, during service discovery, static adaptability analysis uses the provided QoS values to help designers assess the composition and different service selections before service binding. The static analysis, using the estimated probability of failure values and behavioural descriptions of component services, can also provide a predictive analysis as feedback to composition design. Such static analysis helps detect unfavourable services. In the deployment phase, the deployment of a composition includes (*v*) *deployment of adaptation and monitoring mechanisms* (e.g., monitors and sensors). Likewise, deployment-time adaptation actions (e.g., binding), and testing and validation of operational contexts are performed.

At the execution phase or post-mortem (i.e., after the execution), the composition is expected to fulfil its requirements by monitoring itself and its context, observing changes, deciding how to react, and deciding on the actions to execute to adapt to the changes. A monitoring engine performs (*vi*) *runtime monitoring* relying on the monitoring mechanisms and the monitored properties that help detect relevant execution context and changes. The monitored data, which are collected and stored in a dynamic

data storage, include the raw or calculated adaptability data. Basically, an adaptation is performed because monitoring reveals a problem, or because the composition identifies some needed optimisations, or because the execution context changes. The evaluation of the results from the monitoring analysis against the adaptation requirements helps identify adaptation needs which leads to two different directions: executing either evolution or adaptation of the service composition. The first case restarts the design time adaptation process from the requirements engineering adaptation activities while the second case proceeds with the runtime adaptation process. In the second case, it continues to (*vii*) *identify the adaptation needs* that can be triggered from monitored events, adaptation requirements, or changes in context. An important role is played by the context which may include the set of component services available for the service composition, the computational resources available, protocols, user characteristics and preferences, execution properties (e.g., the conditions under which the composition and its component services execute), and composition runtime environment.

Part of assessing adaptation needs is the dynamic adaptability analysis which verifies the fulfilment of the quantitative adaptability requirements. By relying on the fulfilment of the adaptability requirements, it calculates the actual, dynamic adaptability values of the composition and its component services. For example, if we consider reliability as a requirement, we can calculate reliability for the entire composition based on the reliability values of the component services. At certain time intervals, we obtain reliability data from the dynamic data storage, perform reliability analysis, and trigger adaptation initiatives which could replace the unreliable services.

In order to address each of the adaptation needs, there is a need to (*viii*) *identify adaptation strategies*. Suitable strategies define the possible ways to achieve those adaptation needs given the current situation. Identifying the most suitable strategy is supported by the decision making unit that decides using multiple criteria such as current situational context, knowledge of previous adaptations and executions, and

user decisions or preferences. Then the (*ix*) *enactment of the adaptation strategy* (i.e., the selected strategy instance) is performed using available adaptation mechanisms. Upon successful implementation, a new version of the service composition is realised through a change to either the currently executed composition instance or the whole composition model. Finally, the modified service composition is re-activated in the current operational context. It may be through complete re-deployment, or the execution may continue from the previous state according to the new model.

### 2.2.7 The building blocks for adaptability

Another conceptual point of view for achieving dynamic adaptability is introduced by Alférez and Pelechano (2017), where adaptability has two main building blocks: *design block* and *runtime block*. This is significantly comparable to the adaptability perspective of Bucchiarone et al. (2009) presented in Section 2.2.6. The design block contains the modelling activities to support dynamic adaptability of composite services, while the runtime block utilises the models created at design time to adapt a composition.

The design-related building blocks include the following:

- *Service composition modelling*: this is the abstraction of underlying composite service, utilising composition modelling tools such as the Business Process Modelling Notation (BPMN) (von Rosing, White, Cummins & de Man, 2015).
- *Variability modelling*: this specifies the different adaptation variants of a composite service. Typical variability modelling tools include feature models (Alférez et al., 2014; Kang & Lee, 2013) and goal models (Song & Lee, 2013; E. S. Yu, 1997).
- *Context modelling*: this specifies the abstraction of the composition contexts, which can be used at runtime to reason about the adaptations. The commonly used context models include the ontology-based (Furno & Zimeo, 2014) and

graphical-based (e.g., Context Modelling Language [Henricksen & Indulska, 2006]).

- *Requirements modelling*: this specifies the requirements that must be preserved by the composition at runtime, basically categorised into functional and non-functional requirements. Requirements modelling is oftentimes integrated with variability modelling, hence, the variability models are similarly used to specify composition requirements (Gillain, Faulkner, Jureta & Snoeck, 2013; Morandini, Penserini, Perini & Marchetto, 2017).

Moreover, Alférez and Pelechano (2017) categorise the runtime-related building blocks into two: *closed-world* and *open-world* dynamic adaptations. The closed-world dynamic adaptation assumes all contextual situations are foreseen at design time, as specified in the design-related modelling activities. Common realisation components of the closed-world adaptation include the MAPE-K loop of autonomic computing (Rutten et al., 2017), the models at runtime (i.e., leveraging the design time models to analyse actual runtime context and explicitly construct suitable adaptation techniques [Morin, Barais, Jézéquel, Fleurey & Solberg, 2009]), and the Dynamic Software Product Lines (DSPL) approach for dynamic service recomposition (Hinchey, Park & Schmid, 2012; Baresi, Guinea & Pasquale, 2012). Meanwhile, the open-world dynamic adaptation, which Alférez and Pelechano (2017) call as dynamic evolution, recognises that unexpected situations (i.e., those contexts that are not foreseen at design time) can arise at runtime (C. K. Chang, Jiang, Ming & Oyama, 2009).

### 2.2.8 Adaptability at different service composition layers

According to Kazhamiakin et al. (2010), adaptability approaches can likewise be viewed based on their association to the three functional layers of service composition: *business process management* (BPM), *service composition and coordination* (SCC), and *service*

*infrastructure* (SI). At the BPM layer, the entire business process is described as a workflow composed of business activities. Here, the service composition activities, constraints, and requirements are described in a high-level business process model. The SCC layer associates suitable services to the workflow in order to construct the whole service composition. The SI layer provides the underlying runtime environment for the service composition, and manages available services in a service registry from where the SCC layer selects component services.

Each layer has elements that are critical to adaptation (Kazhamiakin, Pistore & Zengin, 2009). The BPM layer contains the *workflow*, which is the abstract model of the business process (i.e., the workflow can indicate the business activities and the governing business rules); the *key performance indicator* (KPI), a quantitative metrics of the business performance in achieving pre-defined business goals; and the *agile service network*, a model to illustrate cross-organisational interactions among collaborating companies. The SCC layer includes the *service composition*, which creates the composite service to realise the workflow; the *process performance metrics*, which measures the performance of a process or its parts; and the *service metrics*, for monitoring the non-functional properties of services. The SI layer constitutes *service realisation*, the runtime environment for executing services (e.g., grids, clusters, data servers, software, protocols, and network infrastructure); *service registry*, where service information are stored; and *service discovery and selection*, which provides SCC the basic functionalities for the selection of suitable services to realise a service composition.

Adaptation approaches at the BPM level include *i*) modifications at the business process model, control flow, and data flow; *ii*) addition or removal of a KPI, or adjustments of KPI values to the changing business goals; and *iii*) ad-hoc modifications and negotiations performed by the business analyst. BPM level adaptation is demonstrated in the works of Hallerbach et al. (2010); S. H. Chang and Kim (2007); Ly, Rinderle

and Dadam (2008); Casati, Ceri, Pernici and Pozzi (1998); and Hagen and Alonso (2000). SCC level adaptation may involve *i*) re-composition, process optimisation, and recovery to cope with changes in the business process model; and *ii*) individual service adaptations to deal with changes or failures of component services, optimisations, and SLA violations. SCC level adaptation is observed in the works of Verma et al. (2005); Bianculli et al. (2007); Colombo et al. (2006); and Ardagna and Pernici (2007). The SI level adaptation deals with *i*) changes in the service registry such as addition of new services and descriptions; *ii*) changes in the discovery and selection mechanisms; *iii*) and changes in service platforms and resources. Representative approaches for SI level adaptation include those works by Andreozzi et al. (2005); Ardagna and Pernici (2007); and Pernici and Rosati (2007).

It has been observed by Kazhamiakin et al. (2009) that most adaptation approaches are ad-hoc, isolated, or focused only on a specific layer without taking into account dependencies to other layers. This opens up further problems such as: *incoherence of monitored events* — if monitoring is provided in isolation at different layers, then the events are not aligned and critical information is not shared across layers, which may lead to misidentifying the problem source; *lack of adaptation effectiveness* — the failure to achieve the expected adaptation effect for not considering the properties of other layers; *lack of adaptation compatibility* — the adaptation performed in one layer is not compatible with the constraints posed by other layers; and *lack of adaptation integrity* — where an adaptation performed in a certain layer is not enough so that more actions at different layers should be performed. To address these problems, Kazhamiakin et al. (2009) have suggested that a cross-layer adaptation framework is necessary to explicitly relate the different adaptation elements across various functional layers.

### 2.2.9 The taxonomy of adaptability

The approaches for adaptability of composite services can be described using five dimensions, based on the *Why*, *What*, *How*, *Who*, *Where*, and *When* adaptation takes place (Kazhamiakin et al., 2010; McKinley et al., 2004). A taxonomy for composition adaptability, built upon those five dimensions is depicted in Figure 2.5.

A vast amount of work has been done in the effort to address the challenges of adaptability, each attempt focusing on some specific adaptation concerns in the taxonomy. A list of works, classified according to the elements of this taxonomy, is presented in Table 2.5 on page 79 and Table 2.6 on page 80. Additional works are also presented in the surveys made by Truong and Dustdar (2009); Krupitzer, Roth, VanSyckel, Schiele and Becker (2015); Murguzur, Intxausti, Urbieta, Trujillo and Sagardui (2014); and Cognini, Corradini, Gnesi, Polini and Re (2018).

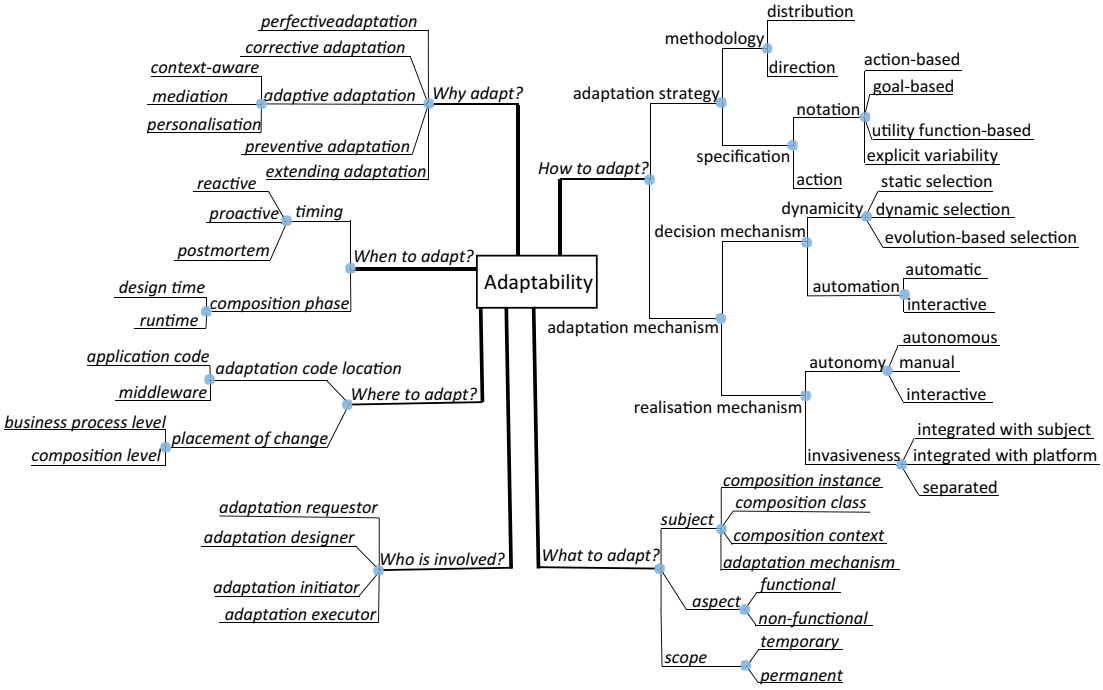



Figure 2.5: Taxonomy of composition adaptability

### The Why dimension

The Why dimension (i.e., Why adaptation is needed?) describes the purpose of adapting compositions. It is classified into: *perfective*, *corrective*, *adaptive*, *preventive*, and *extending* adaptations. These classifications are similarly used to describe the standard software engineering maintenance process (ISO, 2006; Lane, Gu, Lago & Richardson, 2010). Perfective adaptation aims to improve the composition even if it is running correctly (e.g., to optimise quality properties). Corrective adaptation aims to handle faults such as recovery from an undesired behaviour or change of application logic in order to eliminate the fault. Adaptive adaptation modifies the composition in response to changes in its environment. This kind of adaptation is typically necessary for various scenarios such as to accommodate changes in the composition context (e.g., execution context, user context, or physical context), to ensure interoperability between interacting parties (e.g., by providing appropriate adapters and mediators), and to personalise a composition to the requirements of particular users. Preventive adaptation aims to prevent faults or extra-functional issues to occur. Extending adaptation adds new needed functionalities, which significantly modifies (or evolves) the service composition.

### The What dimension

The What dimension describes the adaptation target and the expected result. This dimension is classified into three: *subject*, *aspect*, and *scope* of adaptation. The subject of adaptation is the entity to be modified by the adaptation process such as a *composition instance* (e.g., a business process instance, a service composition customised to a particular user, or a particular configuration of a component service), a *composition class* (i.e., the entire service composition model), and *composition context* (e.g., the environment in which the service composition executes). Adaptation and monitoring mechanisms could likewise be subjects of adaptation changing the way compositions

are adapted. Finer granularity subjects can be represented such as component services, rules, policies, and protocols. The adaptation aspect is a particular concern or focus for adaptation, such as functionality and QoS. Adaptation scope refers to the effect of the adaptation process whether *temporary* (i.e., applies only to a particular instance or in a particular context) or *permanent* (i.e., modifies the composition model that reflects to every instance).

### The How dimension

This dimension defines the *strategies* and *mechanisms* for adaptability. Adaptation strategies are classified according to the methodology used and the way a strategy is specified. On the one hand, adaptation methodology includes the *distribution* and *direction* of the changes. Distribution distinguishes between *centralised* adaptation where adaptation actions are executed on all affected components in a centrally controlled way, and *distributed* adaptation where adaptations are locally performed and propagated among components. The direction of adaptation can be either *forward*, where the strategy drives the composition to a new state to meet the adaptation requirements, or *backward*, where the composition reverts to a previously known state to meet the adaptation requirements. On the other hand, adaptation specification defines the *notations* needed to represent strategies and the particular *actions* performed to enact the strategies (see Section 2.2.4). Notations can be either *implicit* or *explicit*. Implicit strategies are hard-coded and cannot be changed without modification of the adaptation mechanism. Explicit adaptation specification allows the designer to explicitly state adaptation requirements or instructions in the following forms: *i*) *action-based* specification consists of situation-action rules that define actions to perform in certain situations; *ii*) *goal-based* specification establishes performance objectives leaving the composition to determine the actions required to achieve those objectives; *iii*) *utility function-based* specification exploits utility functions to qualify and quantify the desirability of various

Table 2.5: Classification of approaches based on the subject and aspect of adaptation

| Approach | Subject | | | | | | Aspect | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | PI | CS | IP | IF | CI | CC | WF | CO | FR | QS | FN | CX |
| Hallerbach et al. (2010) | ✓ | | | | | | ✓ | | | | | |
| Casati et al. (1998) | ✓ | | | | | | ✓ | | | | | |
| Ly et al. (2008) | ✓ | | | | | | ✓ | | | | | |
| Hagen and Alonso (2000) | ✓ | | | | | | | | ✓ | | | |
| Calinescu et al. (2010) | | ✓ | | | | | | | | ✓ | | |
| Verma et al. (2005) | | ✓ | | | | | | | | ✓ | ✓ | |
| Bianculli et al. (2007) | | ✓ | | | | | | | | ✓ | | |
| H. Wang et al. (2018) | | ✓ | | | | | | | ✓ | ✓ | | |
| Colombo et al. (2006) | | ✓ | | ✓ | | | | | | ✓ | ✓ | |
| Ardagna et al. (2007) | | ✓ | | | | | | | | ✓ | | |
| Rodriguez-Mier et al. (2015) | | ✓ | | ✓ | | | | ✓ | | | ✓ | |
| Chafle et al. (2006) | | ✓ | | | | ✓ | | | | ✓ | ✓ | |
| Ardagna and Pernici (2007) | | ✓ | | | | | | | | ✓ | | |
| Barakat et al. (2018) | | ✓ | | | | | | | ✓ | ✓ | | |
| Z.-Z. Liu et al. (2017) | | ✓ | | | | | | | | ✓ | | |
| Aschoff et al. (2019) | | ✓ | | | | | | | ✓ | | | |
| Brogi and Popescu (2006) | | | ✓ | | | | | ✓ | | | | |
| Benatallah et al. (2005) | | | ✓ | | | | | ✓ | | | | |
| Motahari Nezhad et al. (2007) | | | ✓ | ✓ | | | | ✓ | | | | |
| Kongdenfha et al. (2006) | | | ✓ | | | | | ✓ | | | | |
| Autili et al. (2018) | | | ✓ | | | | | ✓ | | | | |
| Williams et al. (2006) | | | ✓ | | | | | ✓ | | | | |
| Modafferi et al. (2006) | | ✓ | | | ✓ | | | | ✓ | | | |
| Pernici and Rosati (2007) | | ✓ | | | ✓ | | | | ✓ | | | |
| Baresi et al. (2007) | | ✓ | | | ✓ | | | | ✓ | | | |
| Boukhebouze et al. (2009) | | ✓ | | | ✓ | | | | ✓ | | | |
| Erradi et al. (2006) | | ✓ | | | ✓ | | | | ✓ | | | |
| Vasilecas et al. (2016) | | ✓ | | | ✓ | | | | ✓ | | | ✓ |
| Console and Fugini (2007) | | ✓ | | | ✓ | | | | ✓ | | | |
| Tripathy and Tripathy (2018) | | ✓ | | | ✓ | | | | | ✓ | | |
| Mousa et al. (2019) | | ✓ | | | ✓ | | | | ✓ | ✓ | | ✓ |
| Siljee et al. (2005) | | | | | ✓ | | | | | ✓ | | |
| J. Yu et al. (2015) | ✓ | | | | ✓ | | ✓ | | | | ✓ | |
| Alférez et al. (2014) | ✓ | | | | ✓ | | ✓ | | | | ✓ | ✓ |
| Gillain et al. (2013) | ✓ | | | | ✓ | | ✓ | | | | ✓ | |
| Cardellini et al. (2011) | | ✓ | | | | | | | | ✓ | | |
| Hallsteinsen et al. (2012) | | ✓ | | | | | | ✓ | | ✓ | | ✓ |
| Menasce et al. (2011) | | ✓ | | | ✓ | | | | | ✓ | | |

PI: Process Instance,
CS: Component Services,
IP: Interaction Protocol,
IF: Interface,
CI: Composition Instance,
CC: Composition Class

WF: Workflow,
CO: Compatibility,
FR: Failure/Fault Recovery,
QS: QoS,
FN: Functional Requirements,
CX: Context

Table 2.6: Classification of approaches based on the timing, methodology, decision mechanism, and realisation mechanism of adaptation

| Approach | Methodology | | | | | | Decision Mechanism | | | | Realisation Mechanism | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | RE | PR | FW | BW | CN | DS | DY | ST | AU | IT | AT | MA | IT |
| Hallerbach et al. (2010) | ✓ | | ✓ | | ✓ | | ✓ | | | ✓ | ✓ | | |
| Casati et al. (1998) | ✓ | | ✓ | | ✓ | | ✓ | | | ✓ | | ✓ | |
| Ly et al. (2008) | ✓ | | ✓ | | ✓ | | ✓ | | | ✓ | | ✓ | |
| Hagen and Alonso (2000) | ✓ | | ✓ | | ✓ | | | ✓ | ✓ | | ✓ | | |
| Calinescu et al. (2010) | ✓ | | ✓ | | ✓ | | ✓ | | ✓ | | ✓ | | |
| Verma et al. (2005) | ✓ | | ✓ | | ✓ | | ✓ | | ✓ | | ✓ | | |
| Bianculli et al. (2007) | | ✓ | ✓ | | ✓ | | ✓ | | ✓ | | ✓ | | |
| H. Wang et al. (2018) | | ✓ | ✓ | | ✓ | | ✓ | | ✓ | | ✓ | | |
| Colombo et al. (2006) | ✓ | | ✓ | | ✓ | | | ✓ | ✓ | | ✓ | | |
| Ardagna et al. (2007) | ✓ | | ✓ | | ✓ | | ✓ | | ✓ | | | | ✓ |
| Rodriguez-Mier et al. (2015) | ✓ | | ✓ | | ✓ | | ✓ | | ✓ | | ✓ | | |
| Chafle et al. (2006) | ✓ | | ✓ | | ✓ | | ✓ | | ✓ | | ✓ | | |
| Ardagna and Pernici (2007) | | ✓ | ✓ | | ✓ | | ✓ | | ✓ | | ✓ | | |
| Barakat et al. (2018) | ✓ | | ✓ | | ✓ | | ✓ | | ✓ | | ✓ | | |
| Z.-Z. Liu et al. (2017) | ✓ | | ✓ | | ✓ | | ✓ | | ✓ | | ✓ | | |
| Aschoff et al. (2019) | | ✓ | ✓ | | ✓ | | ✓ | | ✓ | | ✓ | | |
| Brogi and Popescu (2006) | | ✓ | ✓ | | | | ✓ | | ✓ | | | | ✓ |
| Benatallah et al. (2005) | | ✓ | ✓ | | | | ✓ | | ✓ | | | | ✓ |
| Motahari Nezhad et al. (2007) | | ✓ | ✓ | | | | ✓ | | ✓ | | | | ✓ |
| Kongdenfha et al. (2006) | | ✓ | ✓ | | | | | ✓ | ✓ | | | | ✓ |
| Autili et al. (2018) | | ✓ | ✓ | | | ✓ | | ✓ | ✓ | | | | ✓ |
| Williams et al. (2006) | | ✓ | ✓ | | | | ✓ | | ✓ | | ✓ | | |
| Modafferi et al. (2006) | ✓ | | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | | ✓ | | |
| Pernici and Rosati (2007) | ✓ | | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | | ✓ | | |
| Baresi et al. (2007) | ✓ | | ✓ | | ✓ | | | ✓ | ✓ | | ✓ | | |
| Boukhebouze et al. (2009) | ✓ | | ✓ | | ✓ | | | ✓ | ✓ | | ✓ | | |
| Erradi et al. (2006) | ✓ | | ✓ | | ✓ | | | ✓ | ✓ | | ✓ | | |
| Vasilecas et al. (2016) | ✓ | | ✓ | | ✓ | | | ✓ | ✓ | | ✓ | | |
| Console and Fugini (2007) | ✓ | | ✓ | | ✓ | | ✓ | | ✓ | | ✓ | | |
| Tripathy and Tripathy (2018) | ✓ | | ✓ | | ✓ | | ✓ | | ✓ | | ✓ | | |
| Mousa et al. (2019) | ✓ | | ✓ | | | ✓ | ✓ | | ✓ | | ✓ | | |
| Siljee et al. (2005) | ✓ | | ✓ | | ✓ | | | ✓ | ✓ | | ✓ | | |
| J. Yu et al. (2015) | ✓ | | ✓ | | ✓ | | | ✓ | ✓ | | | | ✓ |
| Alférez et al. (2014) | ✓ | | ✓ | | ✓ | | | ✓ | ✓ | | ✓ | | |
| Gillain et al. (2013) | ✓ | | ✓ | | ✓ | | | ✓ | ✓ | | | | ✓ |
| Cardellini et al. (2011) | ✓ | | ✓ | | ✓ | | ✓ | | ✓ | | ✓ | | |
| Hallsteinsen et al. (2012) | ✓ | | ✓ | | | ✓ | | ✓ | ✓ | | ✓ | | |
| Menasce et al. (2011) | ✓ | | ✓ | | ✓ | | ✓ | | ✓ | | ✓ | | |

RE: Reactive
PR: Proactive
FW: Forward
BW: Backward
CN: Centralised
DS: Decentralised

DY: Dynamic
ST: Static
AU: Automatic
IT: Interactive

AT: Autonomous
MA: Manual
IT: Interactive

alternatives in order to decide on the fly for the best adaptation option; and *iv*) *explicit variability* specification associates the situations or execution points where adaptation should take place with a set of variants that define different possible implementations of a corresponding composition function.

Adaptation action is an action to execute the strategies in order to fulfil the adaptation requirements. These actions are further classified according to the subject and scope of adaptation: *service instance* adaptation actions (e.g., retry, negotiate SLA, duplicate service, and substitute service), *flow instance* adaptation actions (e.g., substitute flow, undo, redo, and skip), *service class* actions (e.g., change SLA), and *flow class actions* (e.g., re-design, re-plan, change service selection logic, change service registry, and change platform).

The adaptation mechanisms are the techniques and facilities provided at various functional layers of the underlying service composition that enable corresponding adaptation strategies. These mechanisms include *decision mechanisms* through which adaptation approaches may decide on the strategy to better satisfy the adaptation requirements, and *realisation mechanisms* which implement the actual adaptation actions. On the one hand, the decision mechanisms are characterised by the *dynamicity* and *automation* of decision. Dynamicity refers to the flexibility of which the adaptation approach decides on the strategy to be applied: *static selection*, when the adaptation strategy is predefined and explicitly associated with a given adaptation requirement, situation or event; *dynamic selection*, when the adaptation strategy is chosen at runtime based on a concrete situation and context; and *evolution-based selection*, when the adaptation strategy is chosen taking into account not only the current situation but the history of past adaptation decisions. Automation describes the degree of human involvement in the decision process, from a totally automatic (i.e., without user intervention) to interactive (i.e., the user participates in the decision making). On the other hand, realisation mechanisms are characterised by *autonomy* and *invasiveness*.

Autonomy describes the degree of human involvement in the adaptation execution. It can be as a totally autonomous adaptation (i.e., self-adapt) or manual or interactive. Invasiveness distinguishes the integration of the adaptation framework with the subject of adaptation: when the adaptation facilities are integrated with the subject, when the adaptation facilities are integrated with the platform where the subject operates, and when the adaptation facilities are separated from the subject.

### The When dimension

This dimension defines the *timing* of adaptation and the service composition *life-cycle phase* when the adaptive behaviour is integrated with the business logic. Based on the timing of adaptation, *reactive* adaptation is a modification in response to (i.e., after the occurrence of) faults or problems, *proactive* adaptation modifies a composition before a deviation occurs, and *post-mortem* adaptation is evolving the composition through re-design or re-engineering. Based on the service composition life-cycle phase, static adaptation integrates adaptive behaviour during *design time* making it hardwired to the composition while dynamic adaptation is applied at *runtime*.

### The Who dimension

This dimension distinguishes the various adaptation actors and roles, those who are involved in the adaptation process. This dimension relates to the How dimension that introduces a range of approaches from completely autonomous (i.e., self-* approaches) to interactive and manual (i.e., human-in-the-loop approaches). The typical actors are identified as (Hielscher, Metzger & Kazhamiakin, 2009): the *adaptation requestor* (i.e., stakeholders) who defines the adaptation requirements, the *adaptation designer* who defines the adaptation strategies, the *adaptation initiator* who triggers modifications of the service composition in reaction to identified changes, and the *adaptation executor* who performs the adaptation actions defined by the selected strategy.

In the perspective of participation to the life cycle activities, the roles of *requirements engineers*, *designers*, and *adaptation engineers* are identified (Hielscher et al., 2009). A requirements engineer defines the composition requirements that include the adaptation and monitoring requirements. A designer designs the composition and may perform manual or semi-automatic design-time adaptation. An adaptation engineer performs specific activities for defining and specifying adaptation strategies and triggers, and possibly engineering novel adaptation and monitoring mechanisms.

Other roles include the composition *manager* and *end-users*. The manager observes how the composition is executed and evolved in order to make critical decisions such as triggering requests for re-design or re-engineering. The end users directly or indirectly influence the way adaptation and monitoring is performed (i.e., adaptation mechanisms), since a composition aims to adapt to the context of the user and the way the user interacts with the composition.

#### The Where dimension

This dimension describes the location where the adaptation code is placed in the various functional layers of a service composition. It can be in the service composition code itself or in the middleware. This dimension is likewise interpreted as the placement of change in the service composition architecture and environment: *horizontal* and *vertical* placement. Horizontal placement describes the scope of adaptation effect which can be either *local* (i.e., specified to a service instance or restricted to certain clients) or *global* (i.e., transcends the entire service value chain). Vertical placement defines the affected functional layers: adaptation at *service composition level* affects the behavioural protocols and the operational semantics of compositions, adaptation at *business process level* changes business rules, requirements, organisational models, clients, and even the entire value chain, and *cross-layer* adaptation affects multiple functional layers.

### 2.2.10 SDLC methodologies for service composition

Aside from the SC life cycle previously introduced in this thesis, there have been several systems development life cycle (SDLC) methodologies specifically utilised for service-oriented systems engineering (SOSE), and to some extent, have considered adaptability. A detailed survey of SOSE approaches is found in Gu and Lago (2011).

- Rational Unified Process (RUP) for SOA (Brown, Johnston, Larsen & Palistrant, 2005; Kruchten, 2004): is an iterative software development framework created by Rational Software Corporation, a division of IBM, which utilises component-based architectural paradigm and visual modelling of software. It aims to provide an environment for code-centric, visual, and model-driven development, acknowledging (re)use of existing architectural components. The RUP life cycle has four main phases: *inception*, *elaboration*, *construction*, and *transition*.
- Service-oriented Modelling and Architecture (SOMA) (Arsanjani et al., 2008): is a method proposed by IBM, which aims to facilitate a software engineering method for building end-to-end SOA solutions. It defines a fractal software development model that contains six main phases: *business modelling and transformation*, *identification*, *specification*, *realisation*, *implementation*, and *deployment, monitoring, and management*.
- Service-oriented Modelling Framework (SOMF) (M. Bell, 2008): is a SOA framework composed of four major pillars, each pillar defines the directions and corresponding units of work that make up a service-based system development scheme: *practices*, *environments*, *disciplines*, and *artefacts*.
- Service-Oriented Design and Development Methodology (SDDM) (Papazoglou & Van Den Heuvel, 2006): is a service development methodology that aims to provide principles and guidelines to specify, construct, refine, and customise compositions. The methodology comprises one preparatory phase (i.e., *planning*)

and five iterative and incremental main phases: *analysis & design*, *construction & testing*, *provisioning*, *deployment*, and *execution & monitoring*. This approach contains adaptation-specific activities such as service monitoring for continuous evaluation of service level objectives and performance, QoS monitoring based on metrics and SLAs, and alerts for compliance failures.

- BEA SOA Reference (Durvasula, 2006): presents three main stages of development: *requirements & analysis*, *design & development*, and *IT operations*. Each stage encompasses the list of development activities and concerns such as actors, tools, deliverables, and best practices. The BEA methodology relatively caters for adaptability since it includes runtime monitoring activities for the component services and operational management.
- Chang's Service-Oriented Analysis and Design (SOAD) (S. H. Chang, 2007): is a development methodology that contains five main processes: *analysing target services*, *defining unit services*, *acquiring service components*, *developing service adapters*, and *verifying the service composition*. Despite being a concise model, its processes span several adaptability concerns such as modelling variability, modelling mismatches, designing adaptation mechanisms and adapters, and enabling dynamic composition.
- The Service Centric System Engineering (SeCSE) methodology (Di Penta et al., 2009): is a development methodology adopted by the SeCSE project, a European Union-funded project. Engineering activities are contained in three functional areas: *service engineering* (i.e., for developing atomic services), *service-centric system engineering* (i.e., for service composition), and *service acquisition and provisioning* (i.e., the activities associated with the service marketplace). The service-centric system engineering area activities are distinguished into two: *design time* and *runtime*. Design time activities include business modelling, requirements definition, and service-centric architecture and composition design.

Runtime processes include (re-)binding, test, (re)planning/runtime service composition management, execution management, service monitoring, and recovery management.

- Lane et al.'s development framework for adaptive service-based systems (2014): is a development framework that integrates into the SC lifecycle a combination of adaptation activities extracted from the various service-oriented development methodologies (i.e., ASTRO [Trainotti et al., 2005], SDDM, BEA, SOAD, and SeCSE). With the aim to adequately facilitate adaptability, the core adaptation activities from those service-oriented development methodologies are augmented with support activities from the standard software maintenance process. This framework defines five main processes: *requirements engineering & design*, *construction* (includes deployment and provisioning of monitoring components), *operation & management* (includes identifying adaptation needs), *select adaptation strategy*, and *enact adaptation*.
- MUSIC methodology (Hallsteinsen et al., 2012): is a model-driven development methodology from the MUSIC project, a project for creating a development framework for adaptive context-aware applications in open, heterogeneous ubiquitous computing environments. The framework utilises a support middleware that automatically transforms adaptability models into the necessary source code to publish the application's adaptation capabilities, context dependencies, and variability features. The methodology consists of five main tasks: *analysis*, *modelling*, *model transformation & deployment*, *testing and validation*, and *operation*.
- De Sanctis and Marconi's design for adaptation framework (2018): is a recent development framework for context-aware service-based systems intended for multiple application domains. It comprises three major high-level components, namely: the *system models*, the *adaptation*, and the *interaction*. The system model level includes the specifications for the domain object and context. The domain

object defines the behavior of the service it models (i.e., *core process*), and the functionalities it provides. At the adaptation level, strategies and mechanisms already supported by the system models, are defined and configured. At the interaction level, adaptation enactment happens as applications are executed, where various execution and user contexts trigger the adaptation embodied with dynamic service composition.

### 2.2.11 Discussion

It is evident in the literature that adaptability in service-based systems has been broadly studied. It is also apparent how the study of such adaptability has become complex, as shown by the extremely diverse adaptation concerns among the examined works. The role of a high-level conceptual model has been significant in the organisation and conceptualisation of adaptability solutions (Medvidovic, 2017). Such a model presents the adaptability problem at a very high level and defines adaptation activities and mechanisms in a general way. Having that comprehensive scope, high level adaptability models can easily capture the *what*, *when*, *where*, and the *why* of adaptation, targeting broad classes of service-based systems in multiple domains and adaptation scenarios.

However, the most important details of *how* adaptation activities would be achieved are often left unspecified. It turns out that the actual details of *how* to adapt a composition comprises variants of specific and more focused, targeted solutions. These multitudes of specific solutions have diverse concerns for adaptability. Although this may seem intricate, it does not necessarily cause a serious problem since each solution can be typically mapped into the general-purpose adaptability models.

The extensive research into service-based systems has introduced a variety of state-of-the-art solutions on how adaptability would be accomplished. However, much certainly remains to be done. The study of the literature has revealed the three concerns

that are being particularly addressed in this thesis, as highlighted by the posed research questions (see Section 1.3 on page 31). The succeeding sections discuss existing works that strongly relate to such concerns.

## 2.3 On Quantifying Adaptability

The adaptability of service-based systems has attracted significant attention in the literature, but most of the studies are focused on approaches to engineer adaptive systems. This has left little attention on how to decide which form of adaptability or which instance of the adaptive system is suitable in certain cases. Likewise, how these various forms support or constrain other system properties, and how to conduct trade-offs to decide the best adaptability instance for the system, are less studied. Such concerns have also been highlighted in the compilation of recent works on managing design trade-offs in adaptable software systems (see Mistrik et al., 2017).

Designers oftentimes discover several alternatives to applicable adaptability strategies or adaptable design instances. Choosing among those alternatives needs performing trade-offs with other system properties. A trade-off analyses the impact of each adaptability design alternative to the other properties (i.e., QoS), particularly to the prioritised properties. An important step towards this trade-off analysis is to *quantify* the properties, so that these properties can be easily compared among the alternatives. There are already established metrics for some vital properties like reliability and performance; it is necessary to have such similar metrics for adaptability.

### 2.3.1 The perspectives of adaptability

In Table 2.7, we summarise the existing approaches for quantifying adaptability and show the similarities and differences of their characteristics. It is evident how the different measurement essentials and approaches reflect varying adaptability perspectives

Table 2.7: Characteristics of the adaptability metrics (Legend: S-Subjective, A-Architectural, B-Behavioural, AF-Adaptability framework, BP-Business process, GS-General software)

| | Adaptability viewpoints | | | Target of evaluation | | | No. of metrics defined | |
|---|---|---|---|---|---|---|---|---|
| | S | A | B | AF | BP | GS | Multiple | Single |
| ISO/IEC standard (2001) | ✓ | | ✓ | | | ✓ | ✓ | |
| Weibelzahl (2002) | ✓ | | ✓ | | | ✓ | ✓ | |
| Gilb (1988) | | ✓ | | | | ✓ | | ✓ |
| Fenton (1991) | ✓ | | ✓ | | | ✓ | ✓ | |
| Sadat and Ghorbani (2004) | ✓ | | ✓ | | | ✓ | ✓ | |
| Aldris et al. (2013) | ✓ | | ✓ | | | ✓ | ✓ | |
| Raibulet and Masciadri (2009) | ✓ | ✓ | ✓ | | | ✓ | ✓ | |
| Masciadri and Raibulet (2009) | ✓ | | | ✓ | | ✓ | ✓ | |
| Kaddoum et al. (2010) | | ✓ | | | | ✓ | ✓ | |
| Reinecke et al. (2010) | | | ✓ | | | ✓ | | ✓ |
| Dueñas et al. (1998) | | ✓ | | | | ✓ | | ✓ |
| Subramanian and Chung (2001a) | | ✓ | | | | ✓ | ✓ | |
| Perez-Palacin et al. (2014) | | ✓ | | | | ✓ | ✓ | |
| Lenhard (2014) | | ✓ | | | ✓ | | ✓ | |
| Mirandola et al. (2015) | | ✓ | | | ✓ | | ✓ | |
| Khoshkbarforoushha et al. (2016) | | ✓ | | | ✓ | | ✓ | |

and concerns. We classify three main adaptability perspectives considered in the approaches: *subjective, architectural, and behavioural*. The subjective perspective relies on user judgement of the adaptability aspect of interest (Weibelzahl, 2002; Sadat & Ghorbani, 2004; Aldris, Nugroho, Lago & Visser, 2013). The architectural perspective is concerned with the varying system components and structure, while the behavioural perspective focuses on the adaptability of a particular system property of concern that is observable during system execution (Reinecke, Wolter & Van Moorsel, 2010). Although most works are aimed at evaluating adaptability in the general software systems, some works are intended for evaluating the adaptability of specific software domains such as frameworks and processes. Moreover, there are works that propose multiple metrics considering various dimensions of adaptability (Raibulet & Masciadri, 2009; Masciadri & Raibulet, 2009; Kaddoum, Raibulet, Georgé, Picard & Gleizes, 2010). This is in contrast with works that define a single metric for evaluating a specific adaptability concern (Reinecke et al., 2010).

The common trait among the approaches is that each can be used to evaluate adaptability from either a certain perspective or an adaptability aspect of interest. For instance, the adaptability metrics defined in the ISO/IEC standard (ISO, 2001) concentrate on user behaviour (e.g., to assess if users can adapt easily to a given software environment). Similarly, a methodology for empirical evaluation of adaptive systems is presented by Weibelzahl (2002), where the objective of adaptation is reducing the complexity of interaction between users and software systems. Although such analyses of adaptability from the point of view of users are certainly useful, there are other adaptability concerns typically posed by service-based systems that need a different way of evaluation.

Among the earliest adaptability metrics are in Gilb (1988) – where adaptability is based on the proposed measures of software extendability, such as quantifying the number of additions or extensions to an existing system; and in Fenton (1991) – where adaptability is understood as the degree of maintainability, the proposed measurement scale is based on the time spent in doing adaptive maintenance. Furthermore, some approaches have to involve user contributions, such as in Sadat and Ghorbani (2004), where the features for the evaluation of adaptability of hypermedia systems are presented. In Sadat and Ghorbani's metrics, each feature is given a subjective value which is then normalised to compute a certain estimate of the system's adaptability, besides the assigned weights. Other subjective methods to evaluate adaptability are based on stakeholder surveys, e.g., in Aldris et al. (2013). However, there can be limitations to these metrics that are based on human judgement with regards to reproducibility and reliability.

Sets of complementary quality metrics that include the evaluation of adaptability, have been proposed. A set of metrics grouped into four categories to evaluate the quality of design and functionality of adaptive systems is defined in Raibulet and Masciadri (2009). These metrics are calculated statically, although they are intended

to evaluate dynamic adaptability. Likewise, in an attempt to evaluate frameworks for the development of adaptive systems, a set of metrics is proposed in Masciadri and Raibulet (2009). These metrics aim to measure both the personalisation effort necessary to exploit a framework to fit a certain case study, and the indications of a framework's usability. Another set of criteria for the description and evaluation of adaptive properties of self-* systems is proposed in Kaddoum et al. (2010). These criteria are categorised as methodological, architectural, intrinsic, and runtime evaluation measures of the adaptability properties of autonomic systems.

Deviating from the use of multiple metrics, an approach proposed in Reinecke et al. (2010) to evaluate system adaptability relies on a single quantitative metric. The evaluation is derived from repeated measurements of the system performance (i.e., on a system quality of interest), at some discrete points in time. This approach, which exhibits an evaluation of behavioural adaptability, assumes that measurement traces or simulation traces can be obtained from test-beds, from real systems or software tools for discrete-event simulation. The main goal of this metric is to evaluate and compare adaptability with respect to the observable system behaviour at runtime.

### 2.3.2 The architectural perspective of adaptability

Meanwhile, there are approaches that measure adaptability from a software architecture viewpoint. Such metrics focus on the system components, their structure, and their interrelationships that will control the overall design and system adaptation. Decisions about software architecture, which are made at the early design phase of software development, can have a significant impact on the final system quality. Such early decisions will impose constraints on the capabilities and quality of the final system. In addition, poor decisions made at this phase will have more costly consequences if propagated to later stages. Hence, a profound decision for the software architecture

especially considering quality requirements is important. Several works on evaluating adaptability at the software architecture level are discussed in this section.

The evaluation model in Dueñas et al. (1998) describes adaptability as system changeability. This is measured as a parameterisation ratio between the number of parameterised data components in a software architecture and the specified number of data components in a quality specification. In the work of Subramanian and Chung (2001b), software adaptability can be measured in a layered approach by first identifying the adaptability dimension in which to base the measurement. This dimension refers to any attribute external to the software system, and its changes influence the system (e.g., changes in syntax, character-set, or processing speed). An element adaptability index is assigned to every software component (i.e., 1 for adaptable, 0 otherwise) with regards to the defined adaptability dimension. The component indices are aggregated to derive architecture adaptability indices, which are further aggregated to get the overall software adaptability index. This layered approach inspired several other succeeding works such as in Perez-Palacin et al. (2014) and Lenhard (2014).

Perez-Palacin et al. (2014) have proposed metrics to measure adaptability considering the number of components that provide or can provide the functionalities comprising a software architecture. The weighted adaptability index of each functionality is aggregated to get the overall adaptability of the software. Moreover, they examined possible relationships between adaptability and other system quality attributes (e.g., availability and cost). If such relationships exist, improving adaptability can cause either improvement or decline on the other attributes. The work of Lenhard (2014), which is further expanded in Lenhard, Geiger and Wirtz (2015), has proposed metrics to quantify the degree of structural adaptability of codes written in business process languages. Lenhard's notion of adaptability is the portability of applications to various runtime platforms. The purpose is to measure the likelihood of an implementation code to be adapted to another execution platform. Hence, the primary dimension for

adaptability measurement is the number of alternative representations that a language offers for every element comprising the entire application specification. An alternative can be of different form but should have the same runtime semantics. It means the more alternatives exist for a given element, the easier to modify or replace such element when ported to a different runtime platform that does not support its current specification. Such metrics can be helpful for quality assessment during development or decision support during migration. With a higher degree of this code-level adaptability, an application specification can have better chances to be ported or adapted in different platforms. Otherwise, a lower degree can support the decision to rewrite the application code from scratch to fit the new platform.

In Mirandola et al. (2015), the authors have extended the metrics in Perez-Palacin et al. (2014) particularly for measuring the adaptability of a BPEL process in orchestrating distributed services, based on the number of known services associated with a process element. Moreover, they aim to determine relationships that exist between process adaptability and other quality attributes concerning the different configurations of a process. However, their metrics assume a static process structure that excludes the perspective of often highly dynamic context of processes, i.e., changes in the business workflow. Solely relying on this approach to evaluate process adaptability is limited. In particular, processes have been adopting the concept of late binding (see Colombo et al., 2006) where the associated services are unknown beforehand. A new service that provides a needed functionality may become available at runtime.

In Khoshkbarforoushha, Jamshidi, Nikravesh, Khoshnevis and Shams (2009), the adaptability of a process is perceived as the flexibility to adjust to new, different, or changing requirements. The authors argue that the degree of coupling of a process to its environment (e.g., partner Web services, clients, and resources) affects the adaptability of a composite service. Accordingly, adaptation should require a lower degree of dependency between a process and its environment. Thus, they have proposed metrics

to measure the context-independency (i.e., degree of loose-coupling) of a BPEL process. In Khoshkbarforoushha et al. (2016), metrics are introduced to evaluate a BPEL process on the extent to which it can be adapted with future requirements. The authors believe that a process has to be reused in other contexts, organisations, or solutions with minimal effort or change. The authors has focused on quantifying potential mismatches (i.e., description or logic mismatch) that may happen between the process and a potential context of use. For instance, it can be estimated for a certain composite service that it cannot be reused since it cannot be matched with the business rules of potential solutions. Hence, the approach is inclined to predict reusability of a BPEL process.

## 2.4 On Context-aware Requirements Variability

It has been observed that the SC life cycle phases are not equally supported by the existing adaptability approaches. There is a huge gap on adaptability approaches in the early requirements engineering phase of service composition. This echoes the observations revealed in the literature surveys conducted by Ralyte (2012); Mahdavi-Hezavehi et al. (2013); Galster et al. (2014); Alegre et al. (2016); and Guinea et al. (2016). Although there has been a vast amount of work in requirements variability to represent adaptability at the RE phase (Metzger & Pohl, 2014; Galster et al., 2017), such works got short on exploring the integration of requirements variability with context variability. This section presents the state-of-the-art in requirements variability modelling, as well as some existing works on context modelling and context-aware requirements variability.

### 2.4.1 Goal-based requirements modelling

Goal-oriented requirements engineering (GORE) is a powerful approach in the RE literature for eliciting, specifying, and analysing requirements. It captures the intentional

ontology behind software requirements, the why of the software system, and offers very intuitive ways to reason about requirements (E. Yu & Mylopoulos, 1998; Van Lamsweerde, 2001). GORE uses a goal model to represent the rationale of both humans and software systems, and its constructs enable the analysis of high-level goals and the ways to achieve them. Basically, the core of this model is the decomposition of goals into AND/OR constructs. GORE has some core frameworks namely: *i*) the NFR framework (Mylopoulos, Chung & Nixon, 1992; Chung & do Prado Leite, 2009) that focuses on modelling and analysing non-functional requirements; *ii*) the i* (E. S. Yu, 1997) or its variant TROPOS (Bresciani, Perini, Giorgini, Giunchiglia & Mylopoulos, 2004; Castro et al., 2002); *iii*) the KAOS (Darimont, Delor, Massonet & van Lamsweerde, 1997); and *iv*) GBRAM (Anton, 1996). Kolos et al. (2006) have assessed these goal-modelling core frameworks for their applicability in supporting service-based systems. Despite showing some potential, Kolos et al. have found that these approaches are inadequate to capture, analyse, and specify context-aware requirements.

Subsequent works have extended these core approaches to address the following primary GORE concerns (Lapouchnian, 2005):

- *Goal analysis* — this mainly includes goal refinement which is the core activity in GORE. Different approaches define a variety of notations and refinement patterns. To support each approach, some formal and semi-formal reasoning methods are introduced, e.g., informal heuristics, label propagation algorithms (i.e., qualitative or quantitative), temporal logic-based techniques, probabilistic methods, or the combination of methods.
- *Obstacle analysis* — obstacles are the obstructions to achieving goals due to the unanticipated behaviour of agents. Obstacle analysis is a pessimistic view of the goals, requirements, and assumptions, that enables identifying potential hindrances to the achievement of goals (Van Lamsweerde & Letier, 2000). It

aims to produce more complete requirements for a more robust system by systematically identifying *i*) the possible ways that a system might fail to meet its goals, and *ii*) the alternative ways of resolving such anticipated problems as early as possible during requirements engineering.

- *Conflict analysis* — the differences in objectives, needs, concerns, perceptions, knowledge, and skills of system stakeholders (e.g., clients, users, requirements engineers, etc.) produce different and conflicting goals. Capturing and resolving those different viewpoints (i.e., the inconsistencies) leads to a correct and complete requirements specification. However, the differences should be resolved appropriately as early as possible.
- *Variability analysis* — this specifies the many alternative ways of achieving a goal as well as making a choice among the alternatives. Variability can be driven by many forces such as changing users and user needs, dynamics in the availability of resources and external services, variation in hardware resources, stakeholder preferences, customer profiles, and contexts.

### 2.4.2 Contextual requirements

The underlying dynamicity of service-based systems (e.g., changes associated with locations, ambient conditions, available interfaces, bandwidth, heterogeneity of users, temporal and spatial conditions, and user activities) refers to the contexts that must be accounted for in requirements specifications. Context is the reification of the environment — this environment means whatever in the world that provides a surrounding in which a system operates (Finkelstein & Savigni, 2001). There have been various definitions of context bringing different interpretations. Such a lack of consensus may be caused by the differences in the domains being considered by researchers. This thesis dwells upon the definition provided by Abowd et al. (1999): "*Context is any*

*information that can be used to characterise the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and the application themselves*".

Alternatively, Henricksen (2003) provides a service-oriented viewpoint for context: "*The context of a task is the set of circumstances surrounding it that are potentially of relevance to its completion*". Henricksen however argues that context is a vague concept that is difficult to define or bound. Hence, Henricksen suggests to use two related concepts, *context models* and *context information*, which are defined as: "*A context model is a specification of identified concrete subset of the context that is realistically attainable from sensors, applications, and users and able to be exploited in the execution of the task*"; and "*A context information is a set of data, gathered from sensors and users, that conforms to a context model. This provides a snapshot that approximates the state, at a given point and time, of the subset of the context encompassed by the model*".

Moreover, the literature offers various taxonomies of context, i.e., from the general and comprehensive categorisations, to the more specific ones. That shows how contexts have been differently specified by researchers. Perera et al. (2014) present and integrate several of these taxonomies, suggesting that context taxonomies could be combined together, for creating a more comprehensive taxonomy. The "context" in this thesis, however can be sufficiently described by Krogstie's taxonomy (2001), which encompasses the following context types: *spatio-temporal*, *environment*, *personal*, *task*, *social*, and *information*. All of these contexts are relevant for satisfying, particularly the mobility and personalisation tendencies of service-based systems.

An important aspect of requirements variability is *how to specify contexts in the requirements*. Context modelling has been primarily focused on representing context-awareness in an infrastructure-centred view, which assumes that the complexity of engineering context-aware applications can be significantly reduced, only through the use of an efficient infrastructure that is responsible for gathering and managing context

information (Henricksen & Indulska, 2004). But to realise a context-aware requirements variability, it is important to identify and specify the application's usage context that may influence variability in the requirements level.

Specialised RE methodologies that integrate context-awareness have been proposed. Kolos-Mazuryk, Poulisse and van Eck (2005) present an RE approach for service-based systems, focusing on the properties of pervasive services (e.g., context-awareness, mobility, personalisation) that are not given attention in the traditional RE methods. Munoz, Valderas, Pelechano and Pastor (2006) pose a similar assumption that pervasive services need suitable methods to elicit and specify requirements. They utilise the ConcurTaskTree (CTT) technique to model the functional requirements of pervasive systems, particularly the tasks to be performed. Their requirements model captures the *i*) characterisation of the physical environment (i.e., a location model), *ii*) descriptions of tasks (i.e., a CTT model augmented with: location in the physical environment where the task must be achieved, preconditions for achieving a task, interaction interface, and temporal dependency), and *iii*) description of the system behaviour. This requirements model can be later mapped into the Pervasive Modelling Language (PervML) (Muñoz & Pelechano, 2006), a domain-specific language for the specification of pervasive systems in a technology-independent manner.

### 2.4.3 Survey of related work

Software variability modelling describes the various possible configurations of software functionalities to systematically provide a configuration customised to stakeholder needs and preferences. A variability model, especially when associated with context, determines the potential adaptability of a software system in a dynamic operational setting. Some popular research fields aiming for such software system adaptability

include self-adaptive systems (Weyns, 2019; Paucar, Bencomo & Yuen, 2019), autonomic systems (Abeywickrama & Ovaska, 2017), and dynamic software product lines (Bencomo, Hallsteinsen & De Almeida, 2012). Table 2.8 on the following page presents some RE approaches that are tagged according to *i*) the requirements model used, *ii*) the main concerns being addressed, and *iii*) whether the approach involves context modelling, context-aware requirements variability modelling, or personalisation.

### UML-based models

UML-based models have also been explored for context-aware modelling. Sheng and Benatallah (2005) propose the ContextUML, a UML-based modelling approach that is aimed for model-driven development of context-aware services. It utilises UML classes to separately model contexts from context-awareness mechanisms. However, the model is intended for services implementation which focuses mainly on the development phase. Another approach by Choi (2007) introduces utilising a context-aware use case diagram that specifies contexts affecting the functions of a system. This extension of the traditional use case model specifies the variations to fulfil a function that depends on context. The approach also models context variation through a context-switch diagram that combines the UML class and state diagram, in order to express transitions between context changes. Services of a system are classified based on the context roles. The context determines the variants of the use case. However, this approach does not cover non-functional requirements. Likewise, it does not include the analysis of multiple contexts that may affect the requirements. Al-alshuhai and Siewe (2015) propose another approach describing a context-aware extension of the use case diagram. To enable clear separation of concerns between context-awareness requirements and functional requirements, a separate model for context called *use context diagram* is proposed to capture the context (i.e., context information and sources) that will realise the application behaviour. This work by Al-alshuhai and Siewe focuses on modelling context sources

to realise a requirement, but does not specify variations in the requirements affected by contexts.

Table 2.8: A survey of works related to context-aware requirements variability

| | Requirements variability modelling | | | | | CM | CR | Primary concern | | | | PS |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | GM | RN | PF | UM | FM | | | GA | OA | CA | VA | |
| Henricksen and Indulska (2004) | | | | | | ✓ | | | | | | ✓ |
| Van Lamsweerde et al. (1998) | ✓ | | | | | | | ✓ | | ✓ | | |
| Van Lamsweerde and Letier (2000) | ✓ | | | | | | | ✓ | ✓ | ✓ | | |
| Robinson (1989, 1990) | ✓ | | | | | | | ✓ | | ✓ | | |
| Boehm et al. (1995) | ✓ | | | | | | | ✓ | | ✓ | | |
| Letier and Van Lamsweerde (2004) | ✓ | | | | | | | ✓ | | | ✓ | |
| Mylopoulos et al. (1992) | ✓ | | | | | | | ✓ | | | ✓ | |
| In et al. (2001) | ✓ | | | | | | | ✓ | | ✓ | | |
| Yen and Tiao (1997) | ✓ | | | | | | | ✓ | | ✓ | | |
| Giorgini et al. (2002) | ✓ | | | | | | | ✓ | | | ✓ | |
| Sebastiani et al. (2004) | ✓ | | | | | | | ✓ | | | ✓ | |
| Ernst et al. (2010) | ✓ | | | | | | | ✓ | | | ✓ | |
| Ali et al. (2010) | ✓ | | | | | ✓ | ✓ | | | | ✓ | |
| Ali et al. (2013, 2014) | ✓ | | | | | ✓ | ✓ | | | ✓ | ✓ | |
| Rodrigues et al. (2018) | ✓ | | | | | ✓ | ✓ | | | ✓ | ✓ | ✓ |
| Lapouchnian and Mylopoulos (2011, 2009) | ✓ | | | | | ✓ | ✓ | | | | ✓ | |
| Hui et al. (2003) | ✓ | | | | | ✓ | ✓ | | | | ✓ | ✓ |
| Sutcliffe et al. (2005) | ✓ | | | | | ✓ | ✓ | | | | ✓ | ✓ |
| Sutcliffe and Sawyer (2013) | ✓ | | | | | ✓ | ✓ | | | | ✓ | ✓ |
| Liaskos and Mylopoulos (2010) | ✓ | | | | | | | | | | ✓ | |
| Liaskos et al. (2011) | ✓ | | | | | | | ✓ | | | ✓ | |
| Liaskos et al. (2006a) | ✓ | | | | | ✓ | ✓ | | | ✓ | ✓ | |
| Jureta et al. (2010) | | ✓ | | | | | | | | ✓ | ✓ | |
| Oster et al. (2015) | | ✓ | | | | | | ✓ | | | ✓ | |
| C. M. Nguyen et al. (2018) | ✓ | | | | | | | ✓ | | | ✓ | |
| Salifu et al. (2007) | | | ✓ | | | ✓ | ✓ | | | | ✓ | |
| Choi (2007) | | | | ✓ | | ✓ | ✓ | | | | ✓ | |
| Al-alshuhai and Siewe (2015) | | | | ✓ | | ✓ | ✓ | | | | ✓ | |
| Sheng and Benatallah (2005) | | | | ✓ | | ✓ | | | | | | |
| Bosch et al. (2015) | | | | | ✓ | ✓ | ✓ | | | | ✓ | |
| Hartmann and Trew (2008) | | | | | ✓ | ✓ | ✓ | | | | ✓ | |
| Gamez et al. (2011) | | | | | ✓ | ✓ | ✓ | | | | ✓ | |
| Cetina et al. (2009) | | | | | ✓ | ✓ | ✓ | | | | ✓ | |
| Capilla, Ortiz and Hinchey (2014) | | | | | ✓ | ✓ | ✓ | | | | ✓ | |

GM: Goal model
RN: R-nets/ CI-nets
PF: Problem frames
UM: UML
FM: Feature model

CM: Context modelling
CR: Context-aware requirements variability
GA: Goal analysis
OA: Obstacle analysis
CA: Conflict analysis
VA: Variability analysis
PS: Personalisation

### Feature-based models

Feature models (Kang & Lee, 2013; Bosch et al., 2015), which have dominated variability modelling, possess properties similar to goal models such as the hierarchical structuring of elements with AND/OR decomposition. Model elements called features

represent different options for software functionalities. There have been attempts to extend feature models with contextual capabilities. The explicit definition of relations between contexts and features is introduced by Hartmann and Trew (2008). In addition to an existing feature model, a duplicate context variability model is defined to capture the common and variable aspects of a context. They aim to support a software product line engineering that strongly considers the derivation of a product that fits the environment of use. Their approach has played a key role in subsequent works, such as in the management of dynamic software product lines (Capilla, Bosch, Trinidad, Ruiz-Cortés & Hinchey, 2014), in which designers need to identify those features representing system options that may vary with context changes. An integrated modelling approach is proposed by Capilla, Bosch et al. (2014), where contextual features in the feature model are directly labelled with context, and separate supplementary information describe in detail those contextual features. In contrast to the one in Hartmann and Trew (2008), the approach in Capilla, Bosch et al. (2014) avoids duplicating a feature model. Moreover, feature models specifying contextual variabilities have also been exploited in autonomic computing such as in the wireless sensor network field where a feature model represents the potential runtime adaptation of sensor nodes (Gamez et al., 2011), or in a smart home system, where a feature model becomes the basis for runtime reconfiguration plans guiding the selection of system variants in response to environment changes (Cetina et al., 2009). Feature models represent bundles of functionality (functional or non-functional) that are transposed to software configuration options. However, prior requirements that entail these software functionalities are not represented.

#### Problem frames

The relationship between requirements and context is recognised by Salifu et al. (2007) as a fundamental element in designing the adaptability of systems. They apply the problem frames approach in defining a problem description to reason about different

specifications that can satisfy requirements in various contexts. For each different context, there could be a different problem description. This leads to identifying variant problems (variations of the original problem) that are adapted to specific contexts. They focus on monitoring contextual variability and looking for an alternative problem variant specification that fits a context. A context change that violates a requirement triggers a behaviour switch to an alternative specification to ensure the satisfaction of requirements. In this case, however, they treat requirements as invariant. They concentrate on making sure that the given requirements are met in every context. They assume requirements specifications as given, while the approach in this chapter tries to understand the stakeholder goals preceding those specifications.

**Goal-based models**

Although feature models have become popular, particularly in the software product lines (Capilla, Bosch et al., 2014), defining the relationship between each functionality and the context when that functionality should or can be adopted to realise the allowed variability remains a problem. Requirements variability, which is typically characterised by creating multiple subsets of requirements, promotes the problem-space-oriented approach to software adaptability (Dey & Lee, 2017; Metzger & Pohl, 2014; Galster et al., 2014). Such a problem-space-oriented approach highlights the importance of determining the scope of variations in the behaviour of adaptable systems to guarantee that such behavioural variations conform to the stakeholders' goals. Goal modelling has bridged the understanding of variability at an earlier stage (i.e., at the requirements level), it is seen to bridge the existing loose connection between system adaptation and goals. The runtime adaptation, which usually involves reconfiguration of system architecture and components, would manifest the goals, e.g., requirements-aware adaptability (Sawyer, Bencomo, Whittle, Letier & Finkelstein, 2010) and requirements reflection (Bencomo, Whittle, Sawyer, Finkelstein & Letier, 2010). Moreover, stakeholders may

have varying preferences over the functionalities, and defining those preferences and their contexts is another question. Goal modelling aims to address such contextual variability concerns at the requirements level, thus, focusing on requirements variability.

Since goal models have become a popular tool to capture and analyse requirements variability, two major influences to variability have emerged: context and preferences. Various studies show how either contexts or preferences affect variability analysis. Hui et al. (2003) propose a framework that models requirements variability parameters using a goal model that encompasses three components: user goals, skills (capabilities), and preferences. Variability in achieving user goals is influenced by user skills and preferences, which are considered to be the contexts and softgoals, respectively. A user with many strong capabilities would be better matched to an alternative that exercises more skills. In the same manner, a user with many preferences would be better matched to an alternative that has high contributions to those preferences. Similarly, the recent work of Sutcliffe and Sawyer (2013) considers the personal context and behaviour of individual users in the design of software systems. It provides a personalisation framework that aims to match requirements with user needs. It assumes that requirements evolve, such that individual needs change over time. Using a goal model, the analysis of the influence of personal contexts to system goals is approached at two levels: user characteristics and personal goals. For both works, context is solely focused on user capabilities and skills.

Lapouchnian and Mylopoulos (2009) propose an approach for modelling and analysing domain variability and its effects on requirements where a goal model explicitly shows context-dependence by labelling model elements with contextual tags. A Boolean value is assigned to each tag, indicating whether a tag is active or not. A goal model variant is generated based on active contexts wherein a goal model element becomes visible when its context is active. The contexts act as filters by removing irrelevant model fragments in a certain context. This approach is later applied in a framework

(Lapouchnian & Mylopoulos, 2011) extending the i* goal modelling notation (E. S. Yu, 1997) to represent and support variations in goal models. In that framework, the effects of domain variability on a system are captured using a single context-parameterized i* model and the variations of that model are automatically produced based on the currently active contexts.

Ali et al. (2010) propose a contextual goal modelling framework that extends the Tropos requirements modelling (Bresciani et al., 2004; Castro et al., 2002). They aim to derive goal model variants that meet the goals in a given context. Contexts are specified by tagging the goal model elements, and each context can be further classified based on the element they associate with. Accordingly, the context may influence the choice among the alternatives that satisfy a goal. Moreover, this framework integrates an approach to choose variants with minimal development costs. Their further work in Ali et al. (2013) integrates the detection of both the context specification inconsistencies and goal conflicts resulting from variabilities, and their work in Ali et al. (2014) extends their contextual goal model to consider the deployment environment. Meanwhile, a variant of Ali et al.'s contextual goal model is proposed in Rodrigues et al. (2018), where a *persona* information is utilised as contextual conditions for the goals. This approach is aimed towards personalisation of goal achievement.

Furthermore, the effects of preferences to variability have been extensively recognised in the literature. Given the potentially large space of solutions a goal model presents, stakeholder preferences have been used as selection criteria. Two main approaches for preference specification became apparent: the qualitative specification, e.g., Sebastiani et al. (2004); Jureta et al. (2010); Oster et al. (2015), and Ernst et al. (2010), and the quantitative specification, e.g., Liaskos et al. (2011) and C. M. Nguyen et al. (2018). In the qualitative preference valuation, preferences between two goals are directly specified as a binary relation, e.g., requirement $A$ is preferred over requirement

$B$. In a quantitative preference approach, however, scoring functions associated with numerical scores are used to indicate prioritisation. The qualitative ones are first discussed. Techne (Jureta et al., 2010) is a novel requirements modelling framework that uses an r-nets graph to represent requirements, preferences, and their relations. In the r-nets graph, a node represents a preference denoting a binary relation between two requirements. An arrow line is drawn from the more preferred requirement to the preference node, and from the preference node to the less preferred requirement. Techne supports the analysis of r-nets to find the set of candidate solutions to a requirements problem where preferences are used to *i*) compare candidate solutions *ii*) specify the criteria for comparison, and *iii*) specify how the criteria are represented in the requirements model. Meanwhile, the pQGM framework (Ernst et al., 2010) modifies the approach presented in Sebastiani et al. (2004) to derive solutions to a requirements problem utilising both the optional goals (i.e., the 'nice-to-have' requirements) and qualitative preferences among requirements. Recently, another qualitative preference specification technique (Oster et al., 2015), using the conditional importance networks (CI-nets), is introduced to capture complex binary preferences trade-offs between multiple optional goals and softgoals. This preference specification technique is intended for integration with the GORE approach to derive the best set of requirements while considering large sets of stakeholder preferences over goals.

For the quantitative preference specification, Liaskos et al. (2011) propose a goal modelling framework that derives a solution to the requirement problem considering a set of numerical preferences over optional goals. Here, the optional goals refer to softgoals, temporal preferences (the preferred order of achieving goals), or optional hardgoals (hardgoals that are not always mandatory). The preference-based HTN planner is utilised to derive a solution according to the specified preferences. C. M. Nguyen et al. (2018) propose the constrained goal model (CGM), which extends a traditional goal model into an expressive modelling language. The CGM integrates different goal

modelling constructs such that it allows users to express various types of preferences: *i*) preferences via penalties and rewards respectively for tasks and requirements to achieve, *ii*) preferences via multiple objectives to optimise (e.g., minimising or maximising cost, operational time); and *iii*) preferences via binary preference relations between pair of requirements, tasks, or domain assumptions.

**Section summary**

As observed in the examined works, goal models have been widely used for requirements variability modelling. However, other variability specification notations are being adopted, such as feature models, UML-based models, problem frames (Salifu et al., 2007), r-nets (Jureta et al., 2010), and CI-nets (Oster et al., 2015). Feature models possess properties similar to goal models such as the hierarchical structuring of elements with AND/OR refinements. Model elements called *features* represent different options for software functionalities. The elements can also be specified with constraints and properties. Attempts to extend feature models to include context-aware capabilities (Bosch et al., 2015; Capilla, Ortiz & Hinchey, 2014), can enable software designers to explicitly identify and represent the contextual system options that may vary according to changes in the context. Designers can then manage system configurations more efficiently, and associate specific policies and strategies for the activation and deactivation of contextual configurations (Capilla, Bosch et al., 2014). As such, feature models represent bundles of functionality (functional or non-functional) that are transposed to software configuration options. However, prior requirements that entail these software functionalities are not represented. Likewise, C. M. Nguyen et al. (2018) argue that reasoning capabilities in feature models are limited compared to what goal models provide.

## 2.5 On Context-aware Service Selection

Another major and long-standing challenge in the service composition lifecycle is the *discovery* and *selection* of suitable component services (Bouguettaya et al., 2017). The typical Web service discovery process is based on matching the service requester's query to the descriptions of available services in the repository. From the discovered candidate services (i.e., in case there are multiple services with similar functionality), the one with the best QoS is usually selected. According to Rong and Liu (2010), the effectiveness of the discovery process is dependent on the quality of both the user's request and the matchmaking model. Matchmaking models can be characterised as *syntactic*, *semantic*, *context-aware*, *agent-based*, and *recommender-based* (Sheng et al., 2014; Rong & Liu, 2010; Afify, Moawad, Badr & Tolba, 2014). Syntactic matching is the typical keywords-based similarity matching. It is based on matching the actual words in the query with the service descriptions, normally using term analysis methods such as the Term Frequency-Inverse Document Frequency (TF-IDF) (Salton & Buckley, 1988). Semantic matching is based on understanding the intent or meaning of queries and service descriptions. Semantic descriptions, which come from some ontology of domain concepts, are used to specify queries and annotate service descriptions. Eventually, it is the semantic descriptions that will be involved in the matchmaking. Various standards for defining semantic descriptions have been proposed, such as OWL-S (Martin et al., 2004), WSDL-S (Akkiraju et al., 2005), WSML (De Bruijn, Lausen, Polleres & Fensel, 2006), and DAML-S (Ankolekar et al., 2002). A context-aware matching considers context in the discovery of services. Rong and Liu (2010) mention the explicit contexts, which are directly provided by the user; and the implicit ones, which can be automatically or semi-automatically obtained during the matchmaking process.

Other approaches use software *agents* to perform the discovery and selection of

services that match the functional requirements (i.e., functions and tasks that a service should provide), technical requirements (e.g., hardware and operating systems), and budgetary requirements of users (Guo, Wang, Kang & Cao, 2015; Sim, 2011). Maximilien and Singh (2004), for instance, have proposed an agent-based service framework that extends the classical SOA architecture with agents. The agents act as broker implementations for service consumers and agencies (i.e., augmented service registries), and these agents utilise a QoS ontology and XML policy language in the matchmaking process. Software agents have also been utilised in enterprise systems, not only to facilitate functional integration among inter-organisational systems, but also to promote business intelligence and collaborative decision making (W. Shen, Hao, Wang, Li & Ghenniwa, 2007).

### 2.5.1 Recommender approaches

Meanwhile, recommender system techniques have been explored to handle the challenges of service discovery and selection. Early efforts in Web service recommendation and selection have been focused on traditional Web service discovery from standard repositories such as the formerly popular UDDI registries (Pastore, 2008). However, UDDI has become outdated since its use has been discontinued by giant technology companies such as Microsoft, IBM, and SAP. Keywords-based and ontology-based query techniques have been the promising approaches to discover Web services from these repositories. But due to the potential limitations of these early techniques on modern Web service repositories (Broens, Pokraev, Van Sinderen, Koolwaaij & Costa, 2004; Yao, Sheng, Ngu, Yu & Segev, 2015), service recommendation research has been actively looking for better techniques.

Collaborative filtering (CF) methods have become widely adopted in Web service

recommendation. Such methods mainly use the ratings (i.e., feedback) gathered during user-service invocation. The CF methods are typically classified into two types: *memory-based* and *model-based*. On the one hand, the memory-based CF methods (Herlocker, Konstan, Borchers & Riedl, 1999; Linden, Smith & York, 2003; Sarwar, Karypis, Konstan & Riedl, 2001; R. M. Bell & Koren, 2007) are centred on computing relationships, through the rating similarities between users, or alternatively, between Web services. On the other hand, the model-based CF methods use prediction models that are trained using the user-service rating matrix as input, in part or in whole. Collaborative filtering relies on different types of input data contained in a user-service matrix. A more convenient to use input data is the domain-independent *explicit feedback* (i.e., the QoS ratings), for which the majority of CF recommendations focus on. For instance, the work of Zheng et al. (2010) and Shao et al. (2007) recommend Web services based on the similarity of historical QoS ratings of users. The work of X. Chen, Liu, Huang and Sun (2010), which is later extended in X. Chen, Zheng, Liu, Huang and Sun (2013), enhances the memory-based CF method to predict the QoS of candidate Web services, by considering the historical QoS information among geographically clustered users. The geographical clustering is based on users' IP addresses. Likewise, Tang et al. (2015) has proposed a cloud service selection method which exploits users and services' locations to compute neighbourhoods for data smoothing, and then combines the user-based CF and service-based CF to make QoS predictions.

However, obtaining explicit feedback can be difficult for various reasons such as the oftentimes reluctance of users to give feedback, or system limitations in collecting user feedback. Although explicit feedback can generally give more accurate recommendations (Jawaheer, Szomszor & Kostkova, 2010), the better availability of *implicit feedback* has attracted attention in the service recommendation research. Such implicit feedback defines the binary invocation data which can tell whether or not a user has invoked a service.

Table 2.9 on the next page presents some existing service discovery and selection approaches that are marked according to the aforementioned characteristics. The QoS-based selection approaches have different prioritised QoS, such as, cost, reliability, credibility, response time, and throughput. Moreover, dealing with multiple QoS typically involves decision making that applies multi-criteria constraint satisfaction techniques. Such techniques include integer linear programming (Zeng et al., 2004), mixed integer linear programming (Anselmi, Ardagna & Cremonesi, 2007; T. Yu, Zhang & Lin, 2007), Bayesian network-based QoS assessment (G. Wu, Wei, Qiao & Li, 2007), global-aware utility measurement (Kurdija, Silic, Delac & Vladimir, 2019), Analytical Hierarchy Process (AHP) (Godse, Sonar & Mulik, 2008), and Multi-criteria Decision Making (MCDM) (Bagga, Joshi & Hans, 2019; A. F. Huang, Lan & Yang, 2009). A recent survey of methods for MCDM for cloud service evaluation is presented in Alabool, Kamil, Arshad and Alarabiat (2018). In some approaches, multiple QoS have been combined to define new quality models such as reputation (Limam & Boutaba, 2010; Wishart, Robinson, Indulska & Jøsang, 2005), trust (L. Li, Wang & Lim, 2009; Somu, MR, Kirthivasan & VS, 2018), and overall performance (F. Chen, Dou, Li & Wu, 2016).

Aside from the popular collaborative filtering, another fundamental technique is the content-based recommendation (Blake & Nowlan, 2007; Lécue & Delteil, 2007), which utilises the service descriptions as a profile of each Web service to characterise a service's nature or functionality. The profiles are used in the similarity analysis, allowing the association of a target user (i.e., in the context of this chapter the service composition) to the Web services. For instance, a target user is recommended with services of high similarity value to what the user prefers. However, most of the newly emerged Web services, such as APIs, lack the formal semantic information of traditional Web services (e.g., WSDL) typically expressed by a set of attributes, such as *i*) functional category (tags), *ii*) functional parameters (inputs and outputs), and *iii*) requirements

Table 2.9: A survey of works related to service discovery and selection

| | Characteristics of the approach | | | | | | |
|---|---|---|---|---|---|---|---|
| | Syntactic | Semantic | Context-aware | Agent-based | QoS-based | Implicit feedback | Recommender-based |
| Lemmens et al. (2006) | ✓ | ✓ | | | | | |
| Jiao et al. (2017) | | ✓ | | | ✓ | | |
| X. Dong et al. (2004) | ✓ | | | | | | |
| Zou et al. (2009) | ✓ | | | ✓ | ✓ | | |
| Adacal and Bener (2006) | ✓ | | | ✓ | | | |
| Maximilien and Singh (2004) | ✓ | | | ✓ | ✓ | | |
| Sim (2011) | | ✓ | | ✓ | | | |
| J. Wang et al. (2011) | ✓ | ✓ | | | | | |
| Tahamtan et al. (2012) | | ✓ | | | | | |
| Dastjerdi et al. (2010) | | ✓ | | | ✓ | | |
| H.-p. Chen and Li (2010) | | ✓ | | | ✓ | | |
| H. Dong et al. (2011) | | ✓ | | | | | |
| Limam and Boutaba (2010) | | | | | ✓ | | |
| Zeng et al. (2004) | | | | | ✓ | | |
| Anselmi et al. (2007) | | | | | ✓ | | |
| T. Yu et al. (2007) | | | | | ✓ | | |
| G. Wu et al. (2007) | | | | | ✓ | | |
| Kurdija et al. (2019) | | | | | ✓ | | |
| Parhi et al. (2018) | | ✓ | | ✓ | ✓ | | |
| L. Li et al. (2009) | | | | | ✓ | | |
| Somu et al. (2018) | | | | | ✓ | | |
| Bagga et al. (2019) | | | | | ✓ | | |
| Pastore (2008) | ✓ | | | | | | |
| Broens et al. (2004) | | ✓ | ✓ | | | | |
| Yao et al. (2015) | | ✓ | | | ✓ | | ✓ |
| Zheng et al. (2010) | | | | | ✓ | | ✓ |
| Shao et al. (2007) | | | ✓ | | ✓ | | ✓ |
| X. Chen et al. (2013) | | | ✓ | | ✓ | | ✓ |
| Tang et al. (2015) | | | ✓ | | ✓ | | ✓ |
| Q. Yu et al. (2013) | | | ✓ | | ✓ | | ✓ |
| Ryu et al. (2018) | | | ✓ | | ✓ | | ✓ |
| Xu et al. (2016) | | | ✓ | | ✓ | | ✓ |
| Zhu et al. (2018) | | | ✓ | | ✓ | | ✓ |
| Y. Zhang et al. (2019) | | | ✓ | | ✓ | | ✓ |
| H. Wu et al. (2018) | | | ✓ | | ✓ | | ✓ |
| H. Li et al. (2017) | | | | | | ✓ | ✓ |
| B. Cao et al. (2017) | | | | | | ✓ | ✓ |
| Yao et al. (2018) | | | | | | ✓ | ✓ |
| Zhong et al. (2015) | | | | | | ✓ | ✓ |

(preconditions and effects). Without the explicit input/output specifications or functional tags, interface matching between Web services is unattainable and the Web service's intended use is hard to discern. By combining content-based and collaborative filtering (i.e., hybrid approach), the individual weaknesses of either filtering approach can be partially resolved. It is evident in the literature that hybrid approaches have improved Web service recommendation performance, enabling more accurate recommendations (Yao et al., 2015).

### 2.5.2 Matrix Factorisation

Meanwhile, the matrix factorisation (MF) technique for a model-based CF has become dominant in Web service recommendation. Through matrix factorisation's ability to address aspects of data that are often elusive, latent impact factors can be uncovered to build a more effective recommendation. Q. Yu et al. (2013) utilise an MF-based approach of recommending Web services to users, based on clustered QoS representation. To achieve better performance, traditional MF models have been incorporating additional information into the QoS matrix. For example, Ryu et al. (2018) propose a location-based MF approach that is built for QoS prediction. This work analyses location information of users and services, and computes their degree of similarity based on shared invocations. Xu et al. (2016) proposed the integration into the MF model of location-based contexts of users and Web services. In particular, a user context comes from QoS similarity with its neighbouring users, while a service context comes from QoS similarity between services of similar service provider. Zhu et al. (2018) integrate geographic location information, i.e., continent, latitudes, and longitudes, into the matrix factorisation, aiming to achieve a privacy-preserving QoS prediction. Other additional information embedded into MF for QoS prediction include: content similarity (i.e., based on descriptions) (Yao et al., 2015), neighbourhood QoS similarity (Y. Zhang

et al., 2019), and context (i.e., the interactions between user and service environment) (H. Wu et al., 2018).

However, the quality of recommendation from QoS-based approaches largely depends on the quality of available QoS. Most QoS-based recommendation approaches assume that a QoS information with guaranteed quality is readily available. Unfortunately, obtaining QoS has become difficult for various reasons, such as the oftentimes reluctance of users to give feedback, or system limitations in collecting user feedback. Moreover, the reliability of the obtained QoS values can be uncertain due to the impact of varying invocation environment (Zheng et al., 2010; Yao et al., 2018). Although QoS ratings could generally give more accurate recommendations (Jawaheer et al., 2010), the simplicity and better availability of *implicit feedback* has attracted attention towards Web service recommendation techniques that are built upon implicit data. The abundance of implicit feedback comes from merely observing the user-service invocation history, i.e., whether a user invoked a Web service or not. The downside of this binary invocation data is the difficulty of interpreting the implied degree of preference. For instance, a mashup-API invocation is denoted by a single value, i.e., $r_{ui} = 1$, and this value does not tell anything about the degree of preference for an interaction. We emphasise that the effectiveness of a typical QoS-based recommendation relies on this explicitly expressed degree of preference, because the gathered explicit QoS ratings are naturally mapped to a preference scale that runs from a positive to negative extremity.

Hence, the use of additional information to augment the binary user-service invocation data has become more necessary. This complementary information can be generally called contextual information, that includes the available information about the users and services (Y. Li, Hu, Zhai & Chen, 2010), and circumstances describing the user-service interaction (Adomavicius & Tuzhilin, 2011). Several works have attempted to augment the implicit user-service invocation data, to improve the service recommendation performance of MF models. Tags, topic, co-occurrence, and popularity

factors are used to improve the accuracy of Web API recommendation in the model proposed in H. Li et al. (2017). A two-level topic model which combines service content and service network to perform clustering based Web API recommendation for mashups, is proposed in B. Cao et al. (2017). An MF model regularisation based on API co-invocation history is proposed in Yao et al. (2018), which aims to improve API recommendation diversity. A time-aware service recommendation for mashups which integrates temporal information with content similarity is proposed in Zhong et al. (2015).

## 2.6 Chapter Summary

How to create service-based systems that adapt is hard. The characteristics of service-based systems have allowed more convenience for adaptability implementations. Hence, engineering adaptive compositions, reasoning about them, and exploring their use in emerging technology frontiers have become compelling areas for research. Moreover, service composition and the new developments in service computing are raising interesting questions about realising the needed software adaptability in today's ecosystems (Bouguettaya et al., 2017; Schmerl et al., 2017). There are various key drivers for treating adaptability as a first-class concern in modern software systems. For instance, the increasing interactions among service-based systems and their components, require continuous operations in highly dynamic environments. Within such dynamic environments, changes in user demands, system interactions, and business goals are expected to occur. Design decisions that lead to fulfilling the goals of service-based systems must react and adapt to the changes. Likewise, service-based systems operate in an open environment where heterogeneity of component services, diversity in service platforms, and variations in user preferences are expected (Y. Zhou & Chen, 2017). In this open environment, the richness and complexity of context have brought additional concerns

for adaptability.

This chapter has introduced the concepts underlying service computing and adaptability. A comprehensive summary of prior works related to service-based systems' adaptability has been presented, which shows a variety of adaptability concerns and approaches. The integration of adaptability activities into the service composition life cycle has defined the conceptual framework that binds the diverse adaptability solutions. Despite extensive research on the adaptability of service-based systems, many challenges still need attention, among which, three challenges are addressed in the thesis. The first challenge is incorporating adaptability as a design quality attribute in service-based systems. This challenge suggests the need to quantify adaptability. The second challenge is the need for a requirements variability analysis framework in service compositions. Such challenge demands explicitly representing context and preferences in the variability analysis. The third challenge is the need for a context-aware discovery and selection of the most suitable services for a composition. The latter sections of this chapter have specifically focused on works related to the three challenges. Detailed discussions of the works have been presented respectively.

# Chapter 3

# Quantifying the Adaptability of Service Compositions

New and value-added applications, called *service compositions* or *composite services* (Paik, Lemos, Barukh, Benatallah & Natarajan, 2017), have been developed by integrating existing Web services to fulfil more complex user goals. The widespread adoption of service computing, compelled by the expansion of services to both social and economic sectors of society (Bouguettaya et al., 2017), has led to the rapid increase of Web services. Service composition has therefore become a promising development model for modern distributed systems, which accordingly are called *service* (-based) *systems* (Bouguettaya et al., 2017; Kazhamiakin et al., 2010).

Adaptability is widely acknowledged as an inherent quality attribute of service-based systems and has become a major challenge in their design (Kazhamiakin et al., 2010; Metzger et al., 2012; Miorandi et al., 2012; Sheng et al., 2014; Mistrik et al., 2017). Finding ways to handle the dynamic characteristics of service-based systems (e.g., the distributed ownership of component services, heterogeneity of users, and the varying execution environment), has raised the importance of adaptability as a first-class concern. However, the immense focus on the development and classification of approaches

to make systems adaptable, has left little attention on how to evaluate adaptability, which is also equally important. Adaptability allows a system to continuously offer the required functionality and quality of service in a dynamic context, where requirements and the environment constantly change. With adaptability, a composite service can change its process flow logic or partner services at runtime for a particular context of use. In contrast to traditional monolithic systems, service compositions consider the crucial role of adaptability for achieving acceptable system utility, i.e., the measure of a composition's quality (Caporuscio, Grassi, Marzolla & Mirandola, 2016). Adaptability can positively or negatively affect other quality attributes (Perez-Palacin et al., 2014). Hence its design, implementation, and evaluation in the life cycle of service compositions has become a significant undertaking. In moving towards such goal, an important step is to *quantify* adaptability. This can raise developers' and designers' awareness, and aid their decisions in relation to composing adaptable systems. A concrete measurement will enable adaptability to be considered against other quality attributes during design trade-offs. Besides, unlike other quality attributes such as performance and reliability, there has been little work on adaptability metrics; thus, little attention has been given to adaptability in systems design trade-offs (Mistrik et al., 2017).

This chapter presents proposed metrics to quantify the adaptability of service compositions, particularly those specified from the adaptive BPEL-based frameworks. The metrics specifically accommodate two prominent adaptability dimensions: *i*) *structure variability* – the variability of a process[1] to adapt to runtime changes in the composition logic, resulting in changes to the process structure; and *ii*) *binding variability* – the variability of a process that allows it to dynamically select the most suitable concrete service. Quantifying adaptability could yield several benefits to service compositions. On the one hand, it allows designers to evaluate the degree of process adaptability before or during development. For example, designers can get direct feedback for changes

[1]The term *process* is used to refer to a *composite service*.

they introduce, making them aware of changes that limit adaptability. Consequently, quantifying adaptability can improve the overall design of a process, especially if used in conjunction with evaluating other quality attributes. On the other hand, the adaptability of process configurations can be compared between processes specified from the same or different frameworks. This enables designers to evaluate alternative process configurations and decide which is the most suitable to their adaptability requirements and development priorities. Moreover, the metrics extend to quantifying the adaptability of the specification frameworks. This facilitates the designer's decision on the framework to use in specifying a certain process. The metrics are evaluated through a travel booking process case study, in which various representative frameworks that respectively implement existing BPEL-based service composition adaptability mechanisms are examined.

Although adaptability has long been recognised as a quality characteristic and been part of some standardised software quality models, e.g., it is defined as a subcharacteristic of portability in the ISO/IEC standard (ISO, 2001), the perspective may need to vary to accommodate adaptation aspects of higher relevance to service compositions. The work presented in this chapter is closely related to measures for adaptability based on a software architecture viewpoint. That is, it considers the system components, their structure, and their interrelationships that will control the overall design and system evolution. This chapter presents an approach to quantifying the adaptability degree of processes generally defined using the adaptive BPEL-based frameworks. The approach does not only regard adaptability from the perspective of runtime changes in the workflow to expose a new process behaviour, but also considers the changes in runtime binding to partner services. These points of view on adaptability are a common focus of the extended-BPEL frameworks, but their corresponding evaluations have been rarely considered in the literature. Moreover, this approach demonstrates flexibility to accommodate a framework-specific adaptability evaluation which may be more suitable

to some application scenarios.

The rest of the chapter is organised into several main sections. Section 3.1 presents the underlying concepts, Section 3.2 provides the metrics definitions, and Section 3.3 examines the application of the metrics in various adaptive service compositions and frameworks. There are discussions in Section 3.4 about utilising the metrics in the comparison of processes and frameworks, the inclusion of adaptability in design decisions, and the limitations of the proposed metrics. Section 3.5 describes the implementation tool that has been developed and the evaluation of the metrics' comparability property, and Section 3.6 concludes the chapter.

## 3.1 Metrics Rationale

This section introduces the concepts underlying the proposed adaptability metrics. The discussions include the basic elements of a BPEL specification and the major adaptability mechanisms implemented in BPEL. Then, details about the two adaptability dimensions considered in the metrics are presented.

### 3.1.1 The WS-BPEL process

With the emerging service technologies, the Web Services Business Process Execution Language (WS-BPEL or BPEL) has become a popular standard[2] to specify enterprise-level service compositions (J. Yu et al., 2015). It supports the orchestration not just of the conventional XML-based or SOAP-based Web services, but also services using other interfaces or protocols such as RESTful Web services (Pautasso, 2009) and OGC Web services (Fleuren & Müller, 2008).

[2]BPEL has been supported by the industry and open-source community as shown by the development of some popular BPEL engines: e.g., IBM WebSphere Process Server, Oracle BPEL Process Manager (now part of Oracle SOA Suite), Microsoft BizTalk Server, SAP Exchange Infrastructure, and Apache ODE.

Specifying a BPEL composition is essentially defining a new Web service that is a composition of existing services. A BPEL composition shows how multiple service interactions with partner Web services are coordinated to achieve a business goal, and the logic necessary for this coordination (Barreto et al., 2007). The standard BPEL specification describes the control flow and data flow dependencies among the interacting partners (clients and Web services) forming the composition (Charfi & Mezini, 2007; Karastoyanova & Leymann, 2009). The control flow and data flow respectively describe the ordering of interactions and exchange of data. BPEL specification is mainly composed of elements categorised into *atomic* or *structured* activities. The atomic activities (i.e. basic activities), are <$invoke$>, <$receive$>, <$reply$>, and <$assign$>, which respectively invokes a partner Web service, receives a message from a client, generates responses for synchronous operations, and manipulates data variables. The activities <$invoke$>, <$receive$>, and <$reply$> are called "messaging activities" for their use in either one-way (asynchronous) or request/response (synchronous) messaging operations. A structured activity defines the execution flow among the atomic and structured activities within it. Common structured activities are <$sequence$>, <$switch$>, and <$while$>, each of which defines a basic execution control. Other structured activities are <$flow$>, which defines synchronisation and concurrency, and <$pick$>, which selects an execution choice subject to an external event (i.e., a message or alarm event). A BPEL specification (i.e., a BPEL process) is deployed and executed on BPEL compliant orchestration engines. Generally, BPEL describes a workflow-oriented composition model for the behaviour of a business process based on interactions between the process and its partner Web services.

The example used in this chapter considers a simplified travel booking process that arranges both the flight and accommodation booking for a client. The process connects to the airline's Web service which provides the flight booking function and the hotel's Web service which performs the accommodation booking. It takes as input the

flight details and client details, and returns a string description of the travel package. Figure 3.1 shows a BPEL process outline of the booking process. Partner links (lines 2-5) define the different parties that interact with the process. The three partner links could be defined respectively as one for the client that invokes the process, with the other two being for the Web services invoked by the process. The defined variables (lines 6-9) are used to store, reformat, and transform messages sent to and received from partners. Each variable has a message type that is defined in the Web Service Definition Language (WSDL) file of the composition or one of its partners. The process main body (lines 10-19) is a sequence of activities: <$receive$>, which waits for a travel request message from the client (i.e. the process input); <$assign$>, which copies needed data into variables as input to the web services defined in the <$invoke$> activities; and another <$assign$> activity which transforms the outputs of the invoked web services into a message which is sent to the client through the <$reply$> activity.

```
1  <process name="TravelBooking" … >
2   <partnerLinks>
3     <partnerLink name="airline" … />
4     …
5   </partnerLinks>
6   <variables>
7     <variable name= "clientrequest" … />
8     …
9   </variables>
10  <sequence name="mainSequence">
11    <receive name="receiveClientRequest"… />
12    <assign>… </assign>
13    <invoke name="invokeAirlinesService"…
14              … operation="bookFlight" />
15    <invoke name="invokeHotelsService" …
16              … operation="bookHotel" />
17    <assign>… </assign>
18    <reply name="responseToClient" … />
19  </sequence>
20 </process>
```

Figure 3.1: A travel booking process

A BPEL process can be visualised as an hierarchical tree structure with the activities as nodes, and the relationships among activities (control flow) as edges (see Fig. 3.5 on

page 130). The subsequent sections show how this structural representation including the logical behaviour of each activity, becomes relevant to the metrics.

### 3.1.2 Extending BPEL for Dynamic Adaptability

Various categories of adaptability approaches have been implemented in service compositions such as those presented by J. Yu et al. (2015) (i.e., *static* vs *dynamic*, *functional* vs *non-functional*, and *reactive* vs *proactive*). A *static/manual adaptability* needs human efforts every time a system needs to adapt (e.g., inserting code or (re)configuration that requires a system restart). A *dynamic/automatic adaptability* enables runtime changes in the behaviour of a system or some parts of it. Dynamic adaptability is performed by the system itself during execution based on predefined adaptability conditions, and without stopping or restarting (e.g., automatically replacing a non-available service with its alternative). There are approaches focused on adaptability to ensure the *non-functional* quality requirements of a system (e.g., automatically replacing a low-performing partner service with a better one). Other approaches are focused on the *functional* adaptability which copes with the changes in business requirements and environment contexts. Some works also demonstrate *proactive* and *reactive* adaptability approaches, which prescribe adaptabilities that happen before or after an event, respectively.

The standard BPEL provides limited support for dynamic adaptability, and the only way to implement runtime changes is to stop a running process, modify the composition, and then restart the process (Charfi & Mezini, 2007). However, in the volatile environment that modern software systems are expected to operate in, it is undesirable and inefficient to terminate and reconfigure a running process every time a change occurs. Moreover, some critical processes cannot be instantly shut down to implement necessary changes. Hence, various BPEL extension efforts have adopted mechanisms to create frameworks that enable the dynamic adaptability of processes.

Examples of these works, which are called "adaptive BPEL-based frameworks", are discussed by J. Yu et al. (2015). Such extensions address some dimensions of a dynamic environment, such as: changing business rules to fit new requirements, replacing an obsolete partner service, or selecting a new or better service that becomes available. Consequently, dynamic adaptability is characterised by runtime changes in either the process flow logic or partner services.

In general, the various mechanisms applied to BPEL to realise dynamic adaptability are categorised into four types (J. Yu et al., 2015): *i) message interception*, where adaptability takes effect by intercepting BPEL messages (Rosenberg & Dustdar, 2005); *ii) late binding*, where a proxy service is used so that binding to the real service happens at runtime (Colombo et al., 2006); *iii) aspect injection*, where aspects are weaved into the BPEL process to accommodate runtime adaptability (Charfi & Mezini, 2007); and *iv) explicit integration*, where rule activities are explicitly integrated into the BPEL process (Paschke & Teymourian, 2009). The details about these various adaptability mechanisms are discussed in Section 3.3.3 on page 146.

### 3.1.3 Structure and Binding Variabilities

Underlying a service composition is a business workflow, which is a sequence of business activities driven by business rules. Business rules convey specifications of organisational regulations, policies, and decisions (J. Yu et al., 2015). These rules define and constrain the business process, assert the business's structure, and influence the behaviour of the business. Considering a composition as a higher level realisation of a business process, its requirements (i.e., software requirements) are subsequently determined from business rules. The composition's functional requirements specified by the structure and arrangement of component services, are imposed by business rules. However, due to the dynamic nature of organisational and business settings, business

rules are often subject to changes. For instance, business rules may take into account the varying service needs of business clients and their circumstances. Hence, the volatility of business rules becomes a primary source of requirement changes.

An adaptive BPEL process can be extended, changed, or customised to adapt to changing requirements. Additional activities, which associate with external elements such as decision rules, service selection rules, composite services, or services that encapsulate rules, are dynamically introduced to the process. Those activities are called *variabilities*. A process goes through several options; it may offer new functionalities by adding new activities, it may disable or replace its activities, or it may select or replace its partner services. The variabilities are described as either: *i*) *structure variability* or *ii*) *binding variability*. These two variabilities are considered the most relevant adaptability dimensions for service compositions. On the one hand, structure variability enables a BPEL process to be modified at runtime that leads to a structure-variable process. It introduces the capability of adding, removing, or replacing process activities with the variabilities. This typically describes a variability in the business workflow (S. H. Chang & Kim, 2007). For instance, considering the differences in user needs and circumstances, an activity in the workflow may not be requested or a new business rule has to be applied for a particular user. Thus, some parts of the process may be performed differently for each user. On the other hand, binding variability gives the ability of the process to select or replace a partner service at runtime. An <$invoke$> activity in the process may have more than one concrete service candidate to provide the associated functionality. In this case, variability occurs when choosing the most appropriate concrete service among the candidates. Both variabilities are regarded as the basis of the adaptability metrics.

Variabilities are generally characterised as activity-driven (i.e., a variability is specified to a particular process activity). The place in the process where such variability occurs is the *variation point*. In structure variability, the *variability concern* $vc \in$ {*Before*,

*Around*, *After*} can be specified to a variation point. Figure 3.2 shows a simplified graph formalism of a process $\pi$ with nodes that represent activities and directed edges that represent control flow dependencies between activities. To simplify the discussion, data flow dependencies are not considered here. A directed edge between two nodes means that the activity associated with the origin node must finish before proceeding to the activity that corresponds to the destination node. Variability concerns $vc_1$, $vc_4$, both $vc_2$ and $vc_3$ are specified on the variation points $a$, $c$, and $d$ respectively. Variability concerns are treated differently depending on their placement in a variation point. For example, $vc_1$ is specified *Before* $a$; $vc_4$ is specified *Around* (i.e., to replace) $c$; $vc_2$ and $vc_3$ are specified *Before* and *After* $d$ respectively. Figure 3.2(b) shows the execution flow of $\pi$ after considering the variability concerns. Each $vc$ is associated with a variability element that is independent of the process. Likewise, a $vc$ may be associated with a single activity represented by a single node, or a set of activities represented by a subgraph of nodes. A variability is invoked once the process execution reaches a variation point where a $vc$ is specified. For example, before and after performing $d$, external activities associated to $vc_2$ and $vc_3$ respectively must be executed (e.g., to fulfil activities defining new business rules). Likewise, the external activity associated with $vc_1$ must be invoked before $a$, while the one associated with the *Around* variability concern $vc_4$ is invoked instead of $c$. The external activities are modified independently upon further changes in the business rules. Hence, the more variability concerns that exist, the more likely a process can be modified and adapted to runtime changes. It then becomes more plausible to find a way to implement external changes when modifying a process. For instance, there are options on how changes are to be accommodated concerning activity $d$. New requirements can be specified or de-specified through $vc_2$ or $vc_3$.

As an example, the AO4BPEL (Charfi & Mezini, 2007) framework extends the standard BPEL through aspect injection to realise runtime process adaptability. AO4BPEL

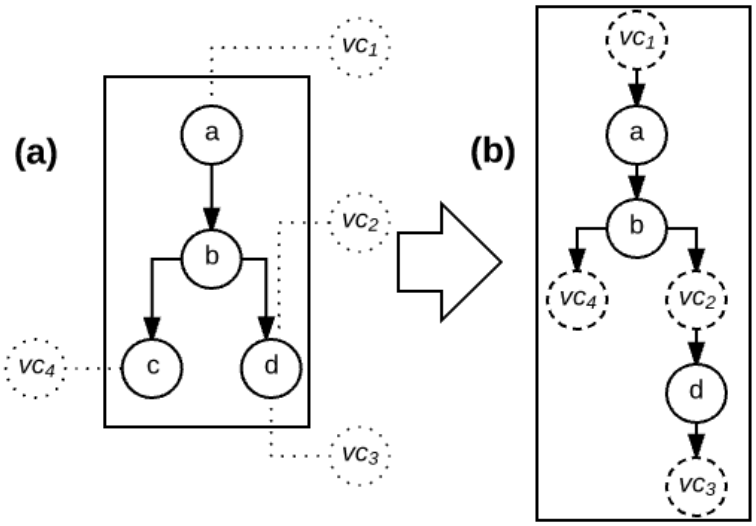



Figure 3.2: An execution logic of a process before (a) and after (b) implementing structure variabilities

introduces the capability of adding, excluding, or replacing process activities. This capability is defined in the *aspects* that encapsulate the *crosscutting concerns*. The places in the process where crosscutting concerns can be specified are generally called *join points*. An aspect describes the *pointcut* — a specific join point, the *advice* — new activity to realise a crosscutting concern, and the *advice type* — the timing of execution of the advice with regards to the occurrence of the pointcut. The advice is specified to a pointcut as *Before*, *Around*, or *After* activity. The *Before* and *After* advice types execute respectively before and after the pointcut. An *Around* advice is interpreted as a replacement activity for the pointcut. The AO4BPEL concepts are illustrated in the simplified process representation shown in Fig. 3.3. The process is composed of five activities: $a$, $b$, $c$, $d$, and $e$. Each activity can be a join point for crosscutting concerns that can be defined in the aspects. Activity $a$ is specified as a pointcut for the aspect $Q1$; when the process executes, activities defined in the advice of $Q1$ are executed *Before* performing $a$. Likewise, aspect $Q2$ has an *Around* advice to be executed instead of the specified pointcuts $b$ and $c$.

Considering the travel booking process in Figure 3.1 on page 121 wherein a new

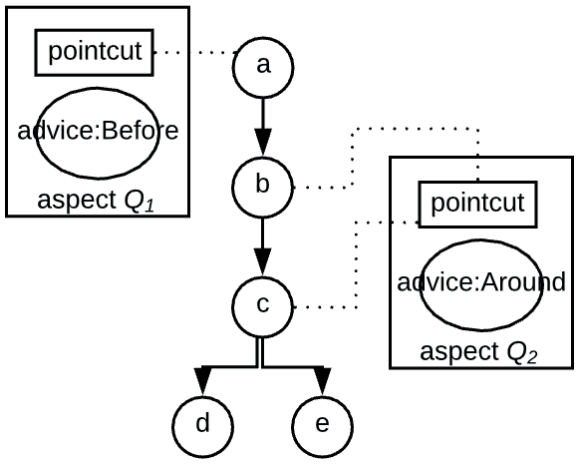



Figure 3.3: A process with specified aspects

requirement arises: to check the validity of a client's travel request (e.g., verify if the request has valid departure and arrival dates) before invoking the airline's Web service. AO4BPEL implements this change as an aspect. Figure 3.4 shows an outline of the *VerifyRequest* aspect. This aspect captures the occurrence of the invoke operation *bookFlight* as the pointcut (lines 8-11). The advice, which invokes operation *verify* of the Verifier Web Service, is a *sequence* activity that executes before the join point activity (lines 12-19). Hence, changes are encapsulated as aspects that are separate from the process. If a business rule changes, appropriate aspects can be activated or deactivated dynamically while the respective process is running, thus applying the adaptation at runtime.

For binding variability, the available *concrete service* $cs$ that can provide the required functionalities of a process are considered. Binding variability has also been studied by Mirandola et al. (2015) and Colombo et al. (2006). This variability is particularly associated with the <$invoke$> activities of a process. Figure 3.5 on page 130 shows a BPEL process in a tree-structure composed of structured activities (nodes $n_1$ and $n_2$) and atomic activities (leaf nodes). The process orchestrates three services, $s_1$, $s_2$, and $s_3$, to meet its overall goal. For each service invoked (e.g., $s_1$), there are concrete

```
1  <aspect name="VerifyRequest" … >
2   <partnerLinks>
3     <partnerLink name="verifier" … />
4   </partnerLinks>
5   <variables>
6     ….
7   </variables>
8   <pointcut name="crosscut1"… >
9     //process[@name="TravelBooking"]
10    //invoke[@operation="bookFlight"]
11  </pointcut>
12  <advice type="before">
13    <sequence>
14      <assign>… </assign>
15      <invoke partnerLink="verifier"…
16                … operation="verify" />
17      <assign>… </assign>
18    </sequence>
19  </advice>
20 </aspect>
```

Figure 3.4: The VerifyRequest aspect

services that match its description (i.e., $cs_1$, $cs_2$, and $cs_3$ are all functionally compliant to $s_1$). These concrete services may offer similar functionalities, but are different in their implementation logic and quality of service (QoS). In Figure 3.5, it is assumed that $cs_2$, $cs_5$, and $cs_6$ are presently used concrete services. The more concrete services exist, the higher the binding variability for the process. In this case, the likelihood of finding an alternative service that equally provides a needed function gets higher.

## 3.2 Quantifying Adaptability

This section describes measurement approaches for the structure and binding variabilities. Both start from defining element-level adaptability metrics followed by the metrics for the entire process, derived from these elemental adaptabilities. The section further presents metrics that combine the structure and binding variabilities, measure

the relative maximal adaptability of a process, and capture the runtime variations of process adaptability.

### 3.2.1 Measuring Structure Variability

The approach aligns with the architectural viewpoint of adaptability (Subramanian & Chung, 2001a; Perez-Palacin et al., 2014; Lenhard et al., 2015), which considers component-level measurement being aggregated to derive a process-level one.

#### Adaptability of Atomic Process Elements

It is common among adaptive BPEL processes to designate atomic activities as variation points. An activity that can be specified with variabilities is considered *variable*, otherwise it is *non-variable*. Structure variability is concerned with the variable atomic activities, and each activity is tagged with a structure variability value.

**Definition 1**: *Structure Variability Value* (*SVV*) — *SVV*<$a$> is the cardinality of the set of variability concerns $\{vc\text{<}a\text{>}_1,...,vc\text{<}a\text{>}_n\}$ specified to an activity $a$.

*SVV*<$a$> $\mapsto \mathbb{N}$ such that *SVV*<$a$> = $\left|\{vc\text{<}a\text{>}_1,...,vc\text{<}a\text{>}_n\}\right|$, where $a$ is a variable atomic activity in the process and $vc$<$a$> is a specified variability concern for $a$. □

In Figure 3.5, variability concern $vc_1$ is specified as *Around* to the activity <$invoke_{s_3}$>. $vc_2$ and $vc_3$ are *Before* variability concerns to <$invoke_{s_1}$> and <$invoke_{s_2}$> respectively; while $vc_4$ invokes a variability after <$invoke_{s_2}$>. To derive the structure variability values, all messaging activities become variable, as indicated by the shaded elements. Most adaptive BPEL-based frameworks are found able to implement variabilities on the messaging activities. Thus, the computation is as follows: *SVV*<$invoke_{s1}$> = 1, *SVV*<$invoke_{s2}$> = 2, *SVV*<$invoke_{s3}$> = 1, *SVV*<$receive$> = 0, and *SVV*<$reply$> = 0. Both <$reply$> and <$receive$> have no specified $vc$.

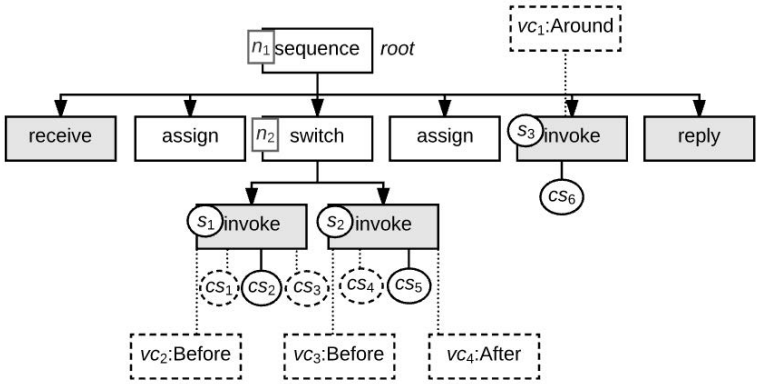



Figure 3.5: A BPEL process showing variation points with specified variabilities

However, absolute values are difficult to interpret, especially if used against other quality attributes. Each value should be mapped into a common range interval (i.e., percentage scale $[0, 1]$), which sets a common interpretation of variability values. Such mapping, which transforms a variability value into a variability degree, normalises and provides a concise meaning to values. That is, a variability value mapped to 0 means no adaptability while a value mapped to 1 means full adaptability. Likewise, a value towards the upper bound of the scale indicates better adaptability, and if mapped towards the lower bound indicates otherwise. To normalise the variability values, a reference value $R$ has to be determined by designers. An $R\langle a\rangle$ is therefore set as the reference value for an activity $a$.

**Definition 2**: *Structure Variability Degree* (*SVD*) — *SVD*$\langle a\rangle$ is the *SVV* of $a$ relative to the value of $R\langle a\rangle$.

*SVD*$\langle a\rangle \mapsto \{x \in \mathbb{Q} \mid 0 \leq x \leq 1\}$ such that *SVD*$\langle a\rangle = \dfrac{SVV\langle a\rangle}{R\langle a\rangle}$, where *SVV*$\langle a\rangle$ is the *Structure Variability Value* of activity $a$, and $R\langle a\rangle$ is the maximum number of variability concerns that can be specified for $a$ such that $\forall a$, *SVV*$\langle a\rangle \leq R\langle a\rangle$. □

Referring to Figure 3.5, the maximum number of variability concerns for every variable activity is three (i.e., *Before*, *Around*, *After*). Based on the structure variability model presented in Section 3.1.3, $R = 3$ for all $a$. Thus, the computation

is: *SVD*<$invoke_{s_1}$> = $1/3 = 0.33$, *SVD*<$invoke_{s_2}$> = $2/3 = 0.67$, *SVD*<$invoke_{s_3}$> = $1/3 = 0.33$, *SVD*<$receive$> = $0/3 = 0$, and *SVD*<$reply$> = $0/3 = 0$.

It should be emphasised that the reference value is not arbitrary and has to be precisely determined by designers. The value of $R$, which is used to normalise the *SVV*s, is based on the maximum number of variabilities found from the implementations of structure variability among the adaptive BPEL-based frameworks. Cautiously setting this value brings two significance. *First*, the resulting metrics always range in the interval $[0, 1]$ which is commonly interpreted as a percentage value. This scale is easy to understand and interpret, which is important for the adoption of any metrics (Lenhard, 2014). *Second*, as these element-level adaptability metrics are aggregated to a single metric describing the entire process, such process-level metric can be used for direct comparison of processes.

### Deriving Process-level Adaptability

There are two approaches to aggregating the individual atomic variability degrees to derive a metric for the whole process. The first approach is straightforward, deriving the mean of the variability degrees. The second approach, which defines an "effective" variability degree, considers runtime behaviours of the elements constituting the process.

**Definition 3**: *Mean SVD (MSVD) — $MSVD_\pi$* is the *SVD* of process $\pi$ relative to the variable activities of $\pi$.

$MSVD_\pi \mapsto \{x \in \mathbb{Q} \mid 0 \leq x \leq 1\}$ such that $MSVD_\pi = \frac{\sum_{i=1}^{n} SVD_{<a_i>}}{n}$, where $\pi$ is a process having $n$ variable activities $a_1$,...,$a_n$. $MSVD_\pi$ is the arithmetic mean of the *SVD*s of the variable activities of $\pi$. □

Referring to the process in Figure 3.5, its *MSVD* is computed as:

$MSVD_\pi$

= (*SVD*<$invoke_{s1}$> + *SVD*<$invoke_{s2}$> + *SVD*<$invoke_{s3}$> + *SVD*<$receive$> + *SVD*<$reply$>)

$\big/ 5$

= (0.33 + 0.67 + 0.33 + 0 + 0) $\big/$ 5 = 0.27.

*MSVD* can give an intuitive measure of adaptability for variable atomic activity comprising the process. The higher the value of this metric for a process, the more adaptable the process elements are, on average. For instance, a fully-adaptable process $\pi$ will generate an $MSVD_{\pi}$ = 1. That is, every variable activity $a$ in $\pi$ is specified with the maximum variability, which is defined by $R$<$a$>. In contrast, a non-adaptable process $\pi$, one without any variability, will have $MSVD_{\pi}$ = 0. A fully-adaptable <$invoke$> activity is shown in Figure 3.6.

In addition to describing the adaptability of the whole process, *MSVD* can be used to compare different processes. An aggregation approach, which considers every variable element comprising the process, could allow comparisons of various processes regardless of their size. The aggregation of individual variability degrees is normalised with respect to the number of variable elements in the process. Moreover, *MSVD* will not just provide immediate quantification of the respective variabilities concerning the entire process, but can also imply the degree of complexity of a process in the effort to implement variabilities. For instance, additional codes that would require some programming efforts, are necessary for the inter-operability of variability interfaces with the process. Hence, *MSVD* can offer an insight into the mean size of potential interactions of each variable activity with external elements. More interactions have to be managed, as additional variabilities are linked to the process.

If the *MSVD* of the process $\pi_2$ in Figure 3.6 is computed, the same assumptions are maintained: the messaging activities are variable, and $R$<$a$> = 3. Hence, $MSVD_{\pi 2}$ = (*SVD*<$receive$> + *SVD*<$invoke$> + *SVD*<$reply$>) / 3 = (0/3 + 3/3 + 0/3) / 3 = 0.33. It is observed that the process $\pi_2$ has higher adaptability (i.e., in terms of structure variability) than process $\pi$ in Figure 3.5 on page 130.

*MSVD* regards equally the adaptability of each variable atomic activity comprising

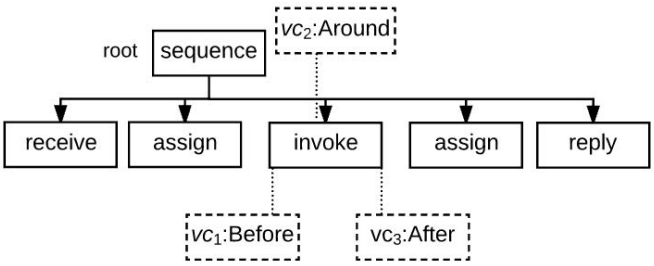



Figure 3.6: A process $\pi_2$ that shows a fully-adaptable element <$invoke$>

the process. This may not always be the case since atomic activities are integrated within structured activities. Both the execution and the variability behaviour of atomic activities are constrained by the structured ones. The runtime behaviour of structured constructs should be considered in the aggregation of adaptability. Each structured construct has its unique behaviour that affects the execution of activities within it. By considering the actual execution behaviours of process elements, the metrics become more concrete with a higher chance of accuracy. Thus, it is important to define aggregation formulations for the structured activities.

**Sequence activity** – a <$sequence$> construct is used to contain activities to be performed in sequential order.

**Flow activity** – a <$flow$> construct provides concurrent execution of its activities. It also offers the possibility of synchronisation among the enclosed activities.

**While activity** – a <$while$> construct defines the iterative execution of activities. The activities within its scope are executed until the specified Boolean condition becomes false.

The structured constructs <$sequence$>, <$flow$>, and <$while$> similarly execute *all* the activities within their scope. Hence,

*SVD*<$c$> = $\frac{\sum_{i=1}^{m} SVD_{<a_i>}}{m}$, where $a_i \big| i = 1, ..., m$ is the $i$th variable activity within the construct $c \in$ {<$sequence$>, <$flow$>, <$while$>}, and $m$ is the number of variable activities contained in $c$.

Furthermore, the conditional constructs <$switch$> and <$pick$> execute only one activity per process instance.

**Switch activity** – a <$switch$> construct defines a conditional behaviour. It consists of an ordered list of conditional branches, each specified by a case element. Each branch contains an activity to be executed when the condition of the case is true. An optional branch specified by a default element can also be included; it is executed when none of the cases is true.

**Pick activity** – a <pick> construct contains activities connected to events (i.e., timer events or message events). It waits for an event to occur before executing an activity associated with the event. Its behaviour is similar to the <$switch$> construct. On simultaneous events, only the first event to occur is processed.
For both constructs, the variability degree depends on the probability of each conditional branch to be executed. The probability is application specific, and can be determined from actual execution data, although methods of determining such probabilities are beyond the scope of this chapter. Hence,

*SVD*<$c$> = $\sum_{i=1}^{m} (SVD_{<a_i>} \cdot p_i)$, where $m$ is the number of activities within the construct $c \in$ {<switch>, <pick>}; $a_i \big| i = 1, ..., m$ is the $i$th variable activity; and $p_i$ is the probability of $a_i$ to be executed, $\sum_{i=1}^{m} (p_i) = 1$.

Examples in this chapter generally apply a discrete uniform distribution, so that each branch has the same chance of execution, such that $p = \frac{1}{n}$, where $n$ is the number of activities contained in $c$.

A structured activity can be placed within another structured activity. *SVDs* are hierarchically aggregated to one or more levels of structured constructs until a value is derived for the whole process. An effective *SVD* of a process is defined by applying the preceding formulations in the aggregation of variability degrees.

**Definition 4**: *Effective SVD (ESVD) — ESVD*$\pi$ = *SVD*<$c$>, where $c$ is the root node in process $\pi$.

*ESVD*$\pi$ is the hierarchical aggregation of *SVD*s considering the runtime behaviour of the structured constructs in a process $\pi$. □

Referring to the process $\pi$ in Figure 3.5 on page 130, the *SVD*<$c$> of each structured construct $c$ is hierarchically derived until the *SVD* of the root node is reached. When a structured construct is an element within another structured construct, the former is considered as a variable activity of the latter.

*SVD*<$switch_{n2}$>
= (*SVD*<$invoke_{s1}$> * $\frac{1}{2}$) + (*SVD*<$invoke_{s2}$> * $\frac{1}{2}$)
= (0.33*0.5) + (0.67*0.5) = 0.5
*SVD*<$sequence_{n1}$>
= (*SVD*<$receive$> + *SVD*<$switch_{n2}$> + *SVD*<$invoke_{s3}$> + *SVD*<$reply$>) / 4 = 0.21.

Since <$sequence_{n1}$> is the root node, *ESVD*$\pi$ = *SVD*<$sequence_{n1}$> = 0.21.

### 3.2.2 Measuring Binding Variability

Binding variability concerns the <$invoke$> activities in a process. Each <$invoke$> activity is tagged with a binding variability value. Computing binding variability follows

the same approach used for structure variability explained in Section 3.2.1, except the mapping of atomic variability values to their corresponding variability degrees.

**Definition 5**: *Binding Variability Value* (*BVV*) — *BVV*<$a$> is the cardinality of identified concrete service choices $\{cs\text{<}a\text{>}_1,...,cs\text{<}a\text{>}_n\}$ specified for an activity $a$.

*BVV*<$a$> $\mapsto \mathbb{N}$ such that *BVV*<$a$> = $\left|\{cs\text{<}a\text{>}_1,...,cs\text{<}a\text{>}_n\}\right|$, where $a$ is an <$invoke$> activity in the process and $cs$<$a$> is an identified concrete service for $a$. □

In Figure 3.5 on page 130, $cs_1$, $cs_2$, and $cs_3$ are the alternative concrete services for <$invoke_{s_1}$>, $cs_4$ and $cs_5$ are for <$invoke_{s_2}$>, and only one concrete service $cs_6$ for <$invoke_{s_3}$>. Therefore, *BVV*<$invoke_{s_1}$> = 3, *BVV*<$invoke_{s_2}$> = 2, and *BVV*<$invoke_{s_3}$> = 1.

Since there is no determined maximum number of concrete services for an <$invoke$> activity, a reference value cannot be set at once. This is unlike the structure variability where 3 is the resolved reference value for every variable activity. Hence, the normalisation of individual variability values is skipped, and the *BVV*s are instead used in the aggregation for deriving a metric of the whole process.

**Definition 6**: *Mean Binding Variability (MBV)* — $MBV_\pi$ is the arithmetic mean of the *BVV*s of the <$invoke$> activities of $\pi$.

$MBV_\pi \mapsto \{x \in \mathbb{Q} \mid x \geq 0\}$ such that $MBV_\pi = \frac{\sum_{i=1}^{n} BVV_{<a_i>}}{n}$, where $\pi$ is a process having $n$ <$invoke$> activities $a_1$,...,$a_n$. □

With reference to the process in Figure3.5, its *MBV* is computed as:

$MBV_\pi$

= (*BVV*<$invoke_{s1}$> + *BVV*<$invoke_{s2}$> + *BVV*<$invoke_{s3}$>) ⁄ 3

= (3 + 2 + 1) ⁄ 3 = 2.

With *MBV*, the binding variability of processes can be compared. A higher value derived for the metric means a higher binding variability of the <$invoke$> activities, on

average.

Furthermore, considering the runtime behaviour of the structured constructs, an effective binding variability can also be computed in the same way *ESVD* is derived. A similar approach is applied to aggregating $BVV$s within the <$sequence$> and <$flow$> constructs but not with the <$while$>. Hence, *BVV*<$c$> = $\frac{\sum_{i=1}^{m} BVV_{<a_i>}}{m}$, where $a_i | i = 1, ..., m$ is the $i$th <$invoke$> activity within the construct $c \in$ {<$sequence$>, <$flow$>} and $m$ is the number of <$invoke$> activities contained in $c$.

It is worth mentioning the difference between the aggregation of *SVD*s and $BVV$s within the <$while$> construct. On the one hand, an activity within the <$while$> construct is iteratively executed, but this does not change its *SVD*. The number of variability concerns, *i.e.*, $vc$, which is specified to an activity during design time, does not change. As mentioned in Section 3.1.3, the only change at runtime is the external activity associated with the $vc$. Individually considering the derived *SVD* value for every iteration eventually results in the same value. Hence, the semantics for *SVD* aggregation for the <$while$> construct is similar to <$sequence$> and <$flow$>. On the other hand, the availability of concrete services may vary with each iteration $j \mid j$=1 to $n$ number of iterations. Hence, *BVV*<$while$> = $\frac{\sum_{j=1}^{n}(BVV'_{<while>})_j}{n}$, where $BVV'_{<while>}$ = $\frac{\sum_{i=1}^{m} BVV_{<a_i>}}{m}$, $a_i | i = 1, ..., m$ is the $i$th <$invoke$> activity among the $m$ number of <$invoke$> activities within <$while$>.

Similar aggregation applies for the conditional constructs. *BVV*<$c$> = $\sum_{i=1}^{m} (BVV_{<a_i>} \cdot p_i)$, where $m$ is the number of <invoke> activities within the construct $c \in$ {<switch>, <pick>}; $a_i | i = 1, ..., m$ is the $i$th <invoke> activity; and $p_i$ is the probability of $a_i$ to be executed, $\sum_{i=1}^{m} (p_i) = 1$.

**Definition 7**: *Effective Binding Variability (EBV) — EBV*$\pi$ = *BVV*<$c$>, where $c$ is the root node in process $\pi$.

*EBV*$\pi$ is the hierarchical aggregation of *BVV*s considering the runtime behaviour of

the structured constructs in a process $\pi$. □

This aggregation is illustrated using the process $\pi$ in Figure 3.5 on page 130. A structured construct within another structured construct is treated similarly as another <$invoke$> activity.

*BVV*<$switch_{n2}$>
= *BVV*<$invoke_{s1}$> * $\frac{1}{2}$ + *BVV*<$invoke_{s2}$> * $\frac{1}{2}$
= (3 * 0.5) + (2 * 0.5) = 2.50.
*BVV*<$sequence_{n1}$>
= (*BVV*<$switch_{n2}$> + *BVV*<$invoke_{s3}$>) / 2 = 1.75.

Hence, $EBV\pi$ = *BVV*<$sequence_{n1}$> = 1.75.

### 3.2.3 Compound Adaptability

Although structure variability and binding variability refer to two different dimensions of adaptability, they can be combined to define a compound adaptability metric for a process. There may be instances when designers need to consider both dimensions in a process evaluation. Such combination is applied to the mean and effective calculations of both structure and binding variability. However, the *MBV* and *EBV* need to be mapped to the common range interval [0,1]. Since the upper limit of these binding variability measures is unbounded, a derivative of the sigmoid function is used in the normalisation. This strictly maps any value of *MBV/EBV* to [0,1], and although sigmoid is a non-linear function, the scope for linear mapping can be approximated by setting a dispersion factor.

**Definition 8**: *Mean/Effective Binding Variability Degree (MBVD/EBVD) — MBVD*$\pi$*/EBVD*$\pi$ $\mapsto \{x \in \mathbb{Q} \mid 0 < x \leq 1\}$, respectively refers to the normalised *MBV*$\pi$*/EBV*$\pi$ through the function

$f(y) = \left( \frac{1}{1 + e^{(-1 \cdot \frac{y}{df})}} \cdot 2 \right) - 1$, where $y$ = *MBV*$\pi$ / *EBV*$\pi$, and *df* is a constant denoting the dispersion factor. □

The dispersion factor has to be carefully determined by designers. For instance, *df*=10 approximates a linear mapping for values from 0 to 10, and provides a gradual scaling with a more dispersed distribution for higher *MBV/EBV* values up to a few tens. Without this dispersion factor, the sigmoid function would quickly map to 1 the values of *MBV/EBV* (*i.e.*, values above 3 would yield variability degrees close to 1 at a dense interval such that their differences become hard to distinguish), and the subsequent rounding-off would make the results practically similar.

In the process in Figure 3.5 on page 130, *MBVD*$\pi$ = 0.100 and *EBVD*$\pi$ = 0.087 are derived.

**Definition 9**: *Compound Adaptability Metric (CAM)* — is distinguished into *MCAM*$\pi$ or *ECAM*$\pi$. *MCAM*$\pi$ is the weighted sum of *MSVD*$\pi$ and *MBVD*$\pi$, while *ECAM*$\pi$ is the weighted sum of *ESVD*$\pi$ and *EBVD*$\pi$.

$$MCAM\pi = (w_x \cdot MSVD\pi) + (w_y \cdot MBVD\pi),$$
$$ECAM\pi = (w_x \cdot ESVD\pi) + (w_y \cdot EBVD\pi)$$

where $w_x, w_y$ are the assigned weights such that $w_x + w_y = 1$, $0 \leq w_x \leq 1$, $0 \leq w_y \leq 1$. □

*MCAM*$\pi$ or *ECAM*$\pi$ respectively refers to the compounded mean or effective variability metrics. Both consider at once the structure and binding variabilities of a process $\pi$. Weights can be assigned to specify the relative importance associated with each adaptability dimension in the metrics.

In the example in Figure 3.5 on page 130, weights 0.7 and 0.3 are assigned to *MSVD* and *MBVD* respectively. Thus, *MCAM*$\pi$ = $(w_x \cdot$ *MSVD*$\pi)$ + $(w_y \cdot$*MBVD*$\pi)$ = (0.7 * 0.27) + (0.3 * 0.100) = 0.219. A similar weighting scheme applied to *ESVD* and *EBVD*

derives $ECAM\pi = 0.173$.

### 3.2.4 Maximal Adaptability

It may be necessary to know the maximum structure variability that a process can obtain, considering every element comprising its specification. This primarily regards *all* atomic elements of the process, particularly looking at the proportion of variable elements against the non-variable ones. Although both *MSVD* and *ESVD* can be similarly augmented to meet this purpose, the more practical metric *ESVD* is chosen as a baseline to define a relative maximal structure variability degree.

**Definition 10**: *Relative Maximal Structure Variability* (*RMSV*) — $RMSV_\pi = SVD^*\langle c\rangle$, where $c$ is the root node of process $\pi$.

$RMSV_\pi$ is the maximal *ESVD* that can be achieved by a process $\pi$ considering its variable components relative to the non-variable ones. Each atomic activity $a$ is tagged with its maximal structure variability value $SVV^*$: if $a$ is variable, $SVV^*\langle a\rangle = R^*\langle a\rangle$, where $R^*\langle a\rangle$ is the maximum number of variability concerns that can be specified to $a$. Otherwise, if $a$ is non-variable, $SVV^*\langle a\rangle = 0$. The maximal structure variability degree $SVD^*\langle a\rangle = \frac{SVV^*<a>}{R^*<a>}$. For every aggregation $SVD^*\langle c\rangle$, where $c$ is a structured construct, all the elements within $c$ are considered. □

Obtaining a maximal structure variability degree $SVD^*\langle a\rangle$ and aggregating such individual variability degrees follow the *SVD* derivation and aggregation previously described in Section 3.2.1. Referring to the process in Figure 3.5 on page 130, these assumptions are expressed: that all messaging activities are variable, and the reference value $R\langle a\rangle = 3$ for all $a$. As the reference value $R\langle a\rangle$ also implies the maximum number of variabilities for a certain variation point $a$, $R^*\langle a\rangle = R\langle a\rangle$. Thus, $R^*\langle a\rangle = 3$. As a result, the individual $SVD^*\langle a\rangle$ of the process elements can be either 1 or 0, when

$a$ is variable or non-variable respectively. To derive the $RMSV_\pi$:

$RMSV_\pi$

= $SVD^*$<$n_1$> = ($SVD^*$<$receive$> + $SVD^*$<$assign$> + $SVD^*$<$n_2$> + $SVD^*$<$assign$> +

$SVD^*$<$invoke_{s1}$> + $SVD^*$<$reply$>) / 6

= (1 + 0 + 1 + 0 + 1 + 1) / 6 = 0.67, where

$SVD^*$<$n_2$> = $SVD^*$<$switch_{n2}$> = ($SVD^*$<$invoke_{s3}$> $\cdot$ $\frac{1}{2}$) + ($SVD^*$<$invoke_{s3}$> $\cdot$ $\frac{1}{2}$) =

(1 * 0.5) + (1 * 0.5) = 1.

### 3.2.5 Extant Adaptability

It may be necessary for designers to track the adaptability of a process as it executes, especially with a large process composed of a long sequence of activities. A set of extant adaptability values can be derived for a process adaptability metric $v$ (e.g., $v \in \{MBV$, $MBVD$, $MSVD$, $EBV$, $EBVD$, $ESVD$, $CAM\}$), that considers the position of the *program counter* during process execution. An activity in the process is designated as an *execution point* from which an adaptability value for $v$ can be computed. That adaptability value tells the actual adaptability at an execution point considering the remaining activities during the execution of a process. Each derived adaptability value disregards the variability of activities preceding the execution point. The execution points should be the child nodes of a sequence activity, e.g., a root node.

**Definition 11**: *Extant Adaptability* ($v_\pi^@$) — $v_\pi^@$ is a set of values $\{x \in \mathbb{Q} \mid 0 \leq x \leq 1\}$ for a process adaptability metric $v$ derived at every execution step of process $\pi$ considering some designated execution points. □

In a tree structure representation of a process $\pi$, the potential execution points for $\pi$ are the child nodes of the root node. Referring to the process in Figure 3.5 on page 130, specifying all the top-level activities within the root node as execution points obtains

an $ESVD^{@}_{\pi}$ = [0.21, 0.28, 0.28, 0.17, 0.17, 0]. Each value in $ESVD^{@}_{\pi}$ corresponds to a derived $ESVD_{\pi}$ as the process progresses in its execution. Each derivation excludes activities that have been completed (i.e., any activity before a certain execution point is considered done). For example, the third element (0.28) of $ESVD^{@}_{\pi}$ is the derived $ESVD_{\pi}$ (i.e., $SVD_{n1}$) when the execution point is at activity <$switch_{n2}$>. Thus, aggregating $SVD_{n1}$ only includes $SVD_{n2}$, $SVD_{s3}$, and $SVD_{reply}$. The activity <$receive$> is excluded since its execution has been completed at that point. Notice that <$assign$>, being a non-variable activity, is discarded but it can be used as an execution point. Likewise, an $EBVD^{@}_{\pi}$ = [0.06, 0.06, 0.06, 0.03, 0.03, 0] is derived for the same process in Figure 3.5.

## 3.3 Case Study: Application of the Metrics in the Adaptive Service Compositions and Frameworks

A case study has been conducted to evaluate the applicability of the metrics and address the following questions:

— *First*, how do we compare the adaptability of processes that may differ in size or specification framework? With the diversity of adaptive BPEL-based frameworks, the broader usability of the metrics is investigated, particularly in comparing processes specified even from different frameworks.

— *Second*, can we quantify the adaptability of specification frameworks? Answering this question examines the usability of the metrics in directly comparing frameworks instead of processes.

— *Third*, how does adaptability measurement impact design decisions? Along with presenting the results of the case study, insights that would rationalise the significance of measuring adaptability are formulated.

This section presents the travel booking process used in the case study, the application of binding variability measurement, and the exploration of various specification frameworks for structure variability measurement.

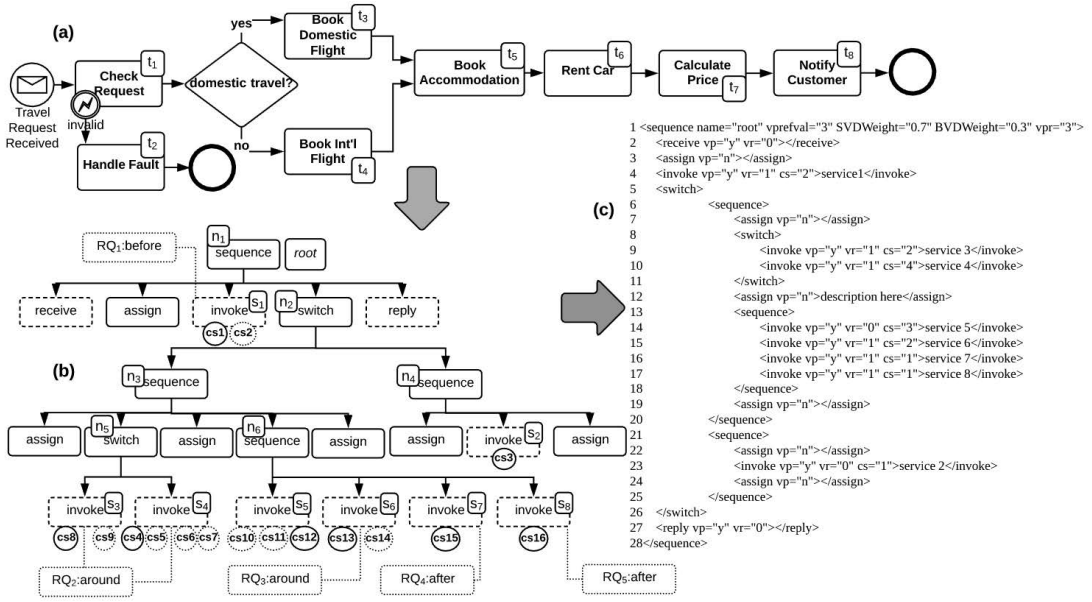



Figure 3.7: A travel booking process in BPMN (a) transformed into a visualised BPEL process (b), and its abstract specification (c)

### 3.3.1 The Travel Booking Process

The case study utilises variants of the popular travel booking process, an example frequently used in the literature (e.g., J. Yu et al., 2015; Charfi & Mezini, 2004; J. Shen et al., 2007).

The composition logic of the travel booking process shown in Figure 3.7(a), is expressed in Business Process Modelling Notation (BPMN) (von Rosing et al., 2015). The process is composed of $m$ tasks $t_i|i = 1, ..., m$. It starts with a customer's request for travel booking. The request contains the customer's personal details and travel preferences. Then, it is checked for validity, i.e., that the request has correct dates, departure, and arrival cities ($t_1$). If invalid, the process invokes a Web service to

inform the customer about the booking failure ($t_2$). Otherwise, based on the specified destination in the travel request, the process will book a domestic or international flight ($t_3, t_4$). Likewise, the process books accommodation ($t_5$) and rents a car ($t_6$). The total price is calculated ($t_7$) and the best offer with a detailed travel plan is sent to the customer ($t_8$).

Figure 3.7(b) shows a transformation of the BPMN into a BPEL process visualised as an activity tree. This serves as the generic base model in the case study. It is simplified to include only the required atomic and structured BPEL activities associated with the BPMN process. The atomic activities are the leaf nodes which can be distinguished as variable or non-variable. The structured activities are the internal nodes tagged with the identifiers $n_1, ..., n_6$. It is also shown that the assumed partner concrete services $cs_1, ..., cs_{16}$ that can provide the functional capabilities for the $m$ number of services $s_i | i = 1, ..., m$ orchestrated by the process. The concrete service with a solid borderline is presently bound to provide $s_i$. It is denoted that task $t_i$ in the BPMN process is fulfilled by service $s_i$ in the BPEL process.

### 3.3.2 On Measuring Binding Variability

Measuring binding variability is straightforward in any process regardless of the process size or specification framework. Given the alternative concrete services in Figure 3.7(b), the process adaptability with respect to its binding variability can be calculated. The calculation is keen on the <$invoke$> constructs which are the variable elements as indicated by the nodes in dashed lines. The following calculations get $MBV_\pi$ = 2.0 and $EBV_\pi$ = 1.844.

$MBV_\pi$

= ($BVV$<$invoke_{s1}$> +...+ $BVV$<$invoke_{s8}$>) / 8 = 2.0.

$EBV_{\pi}$

= $BVV<n_1>$ = ($BVV<invoke_{s1}>$ + $BVV<n_2>$) / 2

= (2 + 1.6875) / 2 = 1.844, where

$BVV<n_6>$ = (3 + 2 + 1 + 1) / 4 = 1.75

$BVV<n_5>$ = (2 * $\frac{1}{2}$) + (4 * $\frac{1}{2}$) = 3

$BVV<n_4>$ = 1

$BVV<n_3>$ = (3 + 1.75) / 2 = 2.375

$BVV<n_2>$ = (2.375 * $\frac{1}{2}$) + (1 * $\frac{1}{2}$) = 1.6875.

The present form of the process in terms of its current alternative concrete services can be assessed. *MBV*=2.0 implies the average number of identified concrete services per <$invoke$> activity in the process. This can become useful to designers particularly in handling trade-offs between binding variability and other measurable qualities. For instance, increasing binding variability could increase process complexity, as demonstrated by Mirandola et al. (2015). That is, additional adaptability implementation codes are necessary to create inter-operability interfaces between the concrete services and the process. Hence, designers may set an arbitrary reference variability value during process configuration, *e.g.*, *MBV*=3.0. Assessing binding variability may lead to either increasing alternative concrete services to improve adaptability if *MBV* falls significantly short of the reference value, or pruning alternatives that would just add to the composition's complexity (*e.g.*, by removing those alternatives with poorer QoS).

The extant adaptability metric could also be helpful in the aforementioned case. Suppose for instance, the child activities of node $n_3$ are specified as execution points, such that an $MBV^{@}_{n3}$ = [2.17, 2.17, 1.75, 1.75, 0] is obtained. The first two execution points pose an *MBV*=2.17, which means the <$invoke$> activities within those two points may have concrete services more than a supposed reference value 2.0. Then,

a transition to *MBV*=1.75 in the third execution point happens, and that would mean the remaining adaptability is deficient. The <$invoke$> activities within the remaining execution points may have concrete services less than the reference value 2.0. Generally, such use of extant adaptability can be similarly applied to any metric. By observing the values and transitions (left to right) between two adjacent execution points, the designer can approximate the portions that significantly affect the increase or decrease of an adaptability metric. If neither increase nor decrease happens, it means any of the adjacent nodes are non-variable constructs concerning the involved metric, or the preceding node has the same adaptability with the remaining nodes. Hence, designers would be able to determine the portions of the process to perform the needed configurations, especially for large processes.

Meanwhile, *EBV*=1.844 considers not just the number of alternatives, but also the runtime behaviours of the associated structured constructs. *EBV* provides the expected adaptability, particularly when the process is executed. For instance, the assumed uniform probability distribution for the <$switch$> construct can change in each application. The actual probability distribution, which can be determined from the application's execution history, may change the derived EBV. Referring to the node $n_2$ in Fig.3.7(a), if branch $n_3$ has a high probability of execution, changes in the number of partner services under $n_3$ will significantly impact adaptability. In contrast, a branch with a low probability of execution has less impact on adaptability. Hence, *EBV* can give a more precise adaptability measure when the execution domain is known.

### 3.3.3 On Measuring Structure Variability

The case study, however, concentrates more on the measurement of structure variability. The dynamic context of the travel booking process, as a result of constant changes in the business environment, organisational policies, and user preferences, increases

the tendency for requirements to change. As a result, the business workflow has to change to cope with the new requirements. The following future (new) requirements *RQ* anticipated by the designer are assumed:

- $RQ_1$ – the system must verify the identity of the customer upon receiving the travel booking request;
- $RQ_2$ – the domestic and international flight booking are merged into one service;
- $RQ_3$ – the renting of a car is based on the customer's preference (e.g., if the preferred tourist attraction is distant from the accommodation);
- $RQ_4$ – new promotion discount rules are formulated – e.g., frequent flyers are given 15% discount; and
- $RQ_5$ – in addition to the existing notification system, customers are notified by call or SMS depending on their situation. These *RQs* represent the potential structure variabilities (*vc*) that the travel booking process has to accommodate.

Generally, the following steps are performed to evaluate the metrics:

1. transforming the business process into its BPEL form applying the described mechanism of an adaptive BPEL framework;
2. examining how a framework implements process adaptability using the sample set of potential structure variabilities; and
3. measuring the adaptability of the resulting process using the proposed metrics.

In this section, various adaptive BPEL frameworks are surveyed and the frameworks are examined on how they implement dynamic changes in the business workflow. The categories of adaptability mechanisms applied by these frameworks are briefly mentioned in Section 3.1.2. For each category, at least one representative framework is chosen. A representative framework is examined on its approach that enables the given travel booking process to become dynamically adaptable. The examined frameworks

are summarised in Table 3.1 on page 157 and their respective implementation details are presented in the following discussion.

### Aspect Injection

Aspect injection exploits the aspect-oriented mechanism to implement runtime changes to a process. The changes are modularised as separate entities, *i.e.*, aspects, which are integrated into a running process without interrupting its execution. Integrating the aspect-oriented mechanism to BPEL aims to efficiently handle runtime adaptation to new business requirements. These requirements, a.k.a. crosscutting concerns, are classified into non-functional crosscutting concerns (e.g., auditing, authentication, logging) and functional crosscutting concerns (e.g., changing business rules, policies, and regulations which affect process flow logic) (Charfi & Mezini, 2007). Three works on aspect-oriented BPEL are examined. These works have similarly utilised the definition of aspects for specifying crosscutting concerns to achieve dynamic adaptability.

A) *AO4BPEL* (Charfi & Mezini, 2007) considers new requirements as crosscutting concerns. Each requirement is encapsulated into an aspect that defines both the advice (i.e., BPEL activities to fulfil the new requirements), and the pointcut (i.e., specified join point in the process to insert the advice). The aspect also defines the advice type that tells whether to execute the advice *before*, *after*, or *instead of* a join point. AO4BPEL uses activity-driven join points, which means join points are specified on the occurrences of process activities. The implementation of AO4BPEL supports join points on messaging activities particularly on the <$invoke$>, <$receive$>, and <$reply$> activities. These activities are considered as variable. For each messaging activity, three variabilities can be specified as $Before$, $Around$, and $After$ aspects. Figure 3.7(b) illustrates an AO4BPEL's implementation of the travel booking process showing the variable activities with dashed borderlines. The five aspects that represent the new

requirements to be specified in the process are also shown. The pointcuts of the aspects, represented by the dotted lines, are counted as the variability concerns for each activity. It is illustrated that requirement $RQ_2$ can be implemented in the process as an $Around$ aspect to both $s_3$ and $s_4$. The adaptability measures for the process (i.e., $MSVD_\pi$ = 0.20 and $ESVD_\pi$ = 0.12) are calculated as follows:

$MSVD_\pi$

= ($SVD<receive> + SVD<invoke_{s1}> +...+ SVD<invoke_{s8}> + SVD<reply>$) / 10

= 0.20.

$ESVD_\pi$

= $SVD<n_1>$ = ($SVD<receive> + SVD<invoke_{s1}> + SVD<n_2> + SVD<reply>$) / 4

= 0.12, where

$SVD<n_6>$ = (0 + 0.33 + 0.33 + 0.33) / 4 = 0.25

$SVD<n_5>$ = (0.33 * $\frac{1}{2}$) + (0.33 * $\frac{1}{2}$) = 0.33

$SVD<n_4>$ = 0

$SVD<n_3>$ = (0.33 + 0.25) / 2 = 0.29

$SVD<n_2>$ = (0.29 * $\frac{1}{2}$) + 0 = 0.145.

B) *MODAR* (J. Yu et al., 2015) adopts a model-driven aspect-oriented approach to support the development of adaptive BPEL-based systems. The approach has to define a *base model* and a *variable model* for a process. On the one hand, the base model abstracts the main procedure or flow logic of the process. It describes the activities (i.e., the general requirements of a business process), and connects these activities using sequential or parallel flows. The base model re-uses BPMN constructs and has two key elements: the *flow objects* are the processing elements, and the *connecting objects* specify the flow relations between flow objects. On the other hand, the variable model captures the volatile decision features of a business requirement. It abstracts into a set of *business rules* the conditional branches in the base procedure (e.g., decision

gateway) and the expected changes to requirements specifications (e.g., new business policy). In addition, MODAR defines a *weave model* to associate the elements of the variable model to the base model. Sets of aspects are created such that each aspect $A$, weaves a rule set from the variable model into a business activity in the base model: $A \in \{\{Before, Around, After\}$ x $T$ x $RS\}$, where $T$ is the set of business activities and $RS$ is the set of rule sets. The aspects, which can be later invoked as $Before$, $Around$, or $After$ aspect services, define the corresponding rules for each activity. The weave model is later transformed into a BPEL process (i.e., the BPEL code is automatically generated from the model). Figure 3.8 shows a schematic BPEL transformation of the travel booking process weave model. A dashed borderline in the weave model in Figure 3.8(a) indicates a variable business activity. It is anticipated that the logic of a variable activity is changeable at runtime, by modifying the specified business rule $RQ$. Notice also how the fault throwing in $t_1$ and the fault handling in $t_2$, are combined into a variable activity *Check Request*. The rules on decisions with regards to fault management are abstracted in $RQ_1$. Likewise, the decision gateway from the BPMN model and the attached activities (i.e., $t_3$ and $t_4$) are encapsulated as the *Book Flight* activity where the associated variabilities are formed as $RQ_2$.

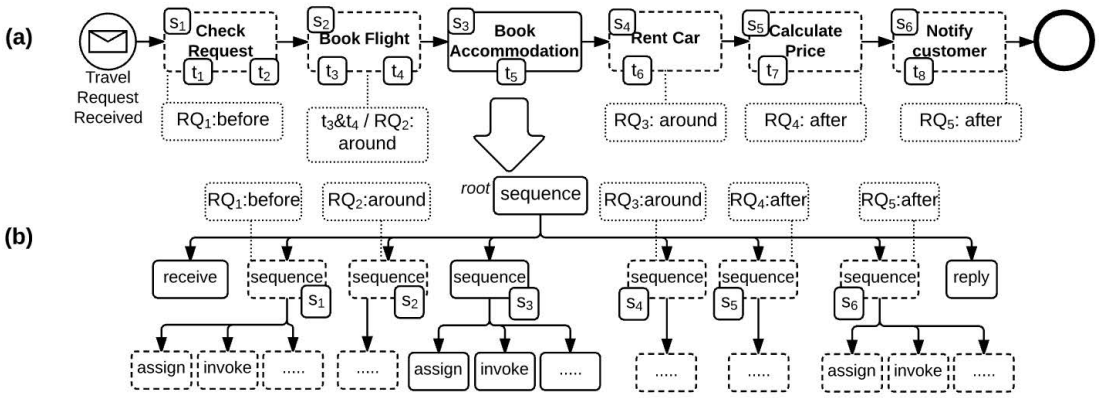



Figure 3.8: The MODAR approach: a weave model (a) transformed into a BPEL process (b)

In transforming a weave model into its BPEL process equivalent, MODAR uses the

<$sequence$> construct to encapsulate each set of BPEL constructs derived from a weave model activity. This abstracts the transformed BPEL process into a series of <$sequence$> constructs. The variation points and variabilities are carried over from the weave model. This approach is illustrated in Figure 3.8(b). The variation points are specified to the <$sequence$> structured constructs. In this case, an *SVV* can be directly derived from a structured activity. Regardless of the activities within it, an encapsulation is regarded as a single activity which can be a counterpart to the variable atomic construct <$invoke$>. In addition, it is still maintained that the messaging activities <$receive$> and <$reply$> are variable. Therefore, $MSVD_{\pi}$=0.21 and $ESVD_{\pi}$=0.21 are the calculated adaptability measures for the process.

$MSVD_{\pi}$

= (*SVD*<$receive$> + *SVD*<$s_1$> +...+ *SVD*<$s_6$> + *SVD*<$reply$>) / 8 = 0.21.

$ESVD_{\pi}$

= *SVD*<$sequence_{root}$> = (*SVD*<$receive$> + *SVD*<$s_1$> +...+ *SVD*<$s_6$> + *SVD*<$reply$>) / 8 = 0.21.

It may always be the case that the derivations of *MSVD* and *ESVD* of a MODAR process are significantly similar, besides generating the same adaptability values. However, the semantics of the two measurements are different. *MSVD* generates the mean variability degree among all variable activities in the process while *ESVD* follows the aggregation formulation that considers the behaviour of the root <$sequence$> activity encompassing the process activities.

C) *BPEL'n'ASPECTS* (Karastoyanova & Leymann, 2009) extends the BPEL engine with a notification framework so that the event life cycles of process activities are published and propagated to a broker. The broker handles the aspect management tool (i.e., used to create, edit, delete, deploy or undeploy aspects), the weaver, and the

communication between the BPEL engine and Web services. An aspect, which defines the adaptation logic, contains an event subscription (i.e., pointcut) and the advice (i.e., Web service operation). Only an <$invoke$> activity is considered as potential join point. The weaver calls the weaved advice $Before$, $Instead$, or $After$ a process activity upon notification from the engine of an event defined by a pointcut in the aspect. This framework can produce a process specification for the travel booking similar to that in Figure 3.7(b). Therefore, $MSVD_{\pi}$=0.20 and $ESVD_{\pi}$=0.12 are calculated, which are expected to be similar to those of the AO4BPEL process.

**Message Interception**

This approach implements runtime changes by intercepting messages that go in or out of the BPEL process. Generally, the changes are exposed as rule services that can be invoked before or after the execution of a process activity. Among several efforts similarly implementing this approach, the work of Rosenberg and Dustdar (2005) is chosen to be examined. In this work, adaptability is realised by specifying business rules upon intercepting messages exchanged between the BPEL process messaging activities (i.e., <$invoke$>, <$reply$>, and <$receive$>) and the partner services. The runtime architecture components, which include the BPEL engine, the Rule Interceptor, the Rules Broker, and the Web Service Gateway, are connected to the Enterprise Service Bus (ESB). The BPEL engine running the process uses the ESB as a messaging layer, and communicates directly with the Web Service Gateway to call external services. The Rule Interceptor checks every message that goes in or out the BPEL process, and determines from the activity-to-rule mapping document whether a rule is associated with the message. The Rules Broker automatically generates Web services for executing rules, which are invoked $before$ or $after$ the execution of a partner service. Hence, the messaging activities of a BPEL process are variable. However, only two variabilities, *Before* and *After*, can be specified to these variability points. Considering the travel

booking process in Figure 3.7(b), the *Around* rules $RQ_2$ and $RQ_3$ cannot be applied. This leaves only $RQ_1$, $RQ_4$, and $RQ_5$ as potential structure variabilities. Henceforth, $MSVD_\pi$=0.10 and $ESVD_\pi$=0.09 are calculated for the process.

$MSVD_\pi$

= ($SVD\langle receive\rangle$ + $SVD\langle invoke_{s1}\rangle$ +...+ $SVD\langle invoke_{s8}\rangle$ + $SVD\langle reply\rangle$) / 10 = 0.10.

$ESVD_\pi$

= $SVD\langle n_1\rangle$ = ($SVD\langle receive\rangle$ + $SVD\langle invoke_{s1}\rangle$ + $SVD\langle n_2\rangle$ + $SVD\langle reply\rangle$) / 4 = 0.09, where

$SVD\langle n_6\rangle$ = (0 + 0 + 0.33 + 0.33) / 4 = 0.165

$SVD\langle n_5\rangle$ = 0

$SVD\langle n_4\rangle$ = 0

$SVD\langle n_3\rangle$ = (0 + 0.165) / 2 = 0.08

$SVD\langle n_2\rangle$ = (0.08 * $\frac{1}{2}$) + 0 = 0.04.

**Explicit Integration**

In this approach, the standard BPEL is extended by explicitly defining and integrating rule activities in the BPEL process. Rule activities become additional constructs to the existing BPEL elements. A rule activity invokes a rule service that defines new policies and requirements. Implementing this approach is the BPEL+Rules framework proposed by Paschke and Teymourian (2009). This introduces a rule-based business process execution environment wherein a rule engine is deployed on an Enterprise Service Bus and exposed as a Web service. An added *Rule Activity* in the BPEL process invokes a rule service which runs a rule engine and executes a rule logic. The rule service defines a declarative specification of a new requirement arising from a change in business policy or regulation. This supports dynamic adaptability, in that rules are modified and applied,

without redeploying the BPEL process.

To apply the metrics for this framework, the Rule Activities are considered as variation points of the process. A rule activity is inserted adjacent and prior to any messaging activity in the process. Hence, each messaging activity can be associated with one rule activity as a *Before* variability. Referring to the travel booking process in Figure 3.7(b) and the given set of potential variabilities, only the *Before* requirement $RQ_1$ can be accommodated. The following calculates $MSVD_\pi$=0.03 and $ESVD_\pi$=0.08 for the process:

$MSVD_\pi$

= (*SVD*<$receive$> + *SVD*<$invoke_{s1}$> +...+ *SVD*<$invoke_{s8}$> + *SVD*<$reply$>) / 10 = 0.03.

$ESVD_\pi$

= *SVD*<$n_1$> = (*SVD*<$receive$> + *SVD*<$invoke_{s1}$> + *SVD*<$n_2$> + *SVD*<$reply$>) / 4 = 0.08, where

*SVD*<$n_2$> = 0.

**Late Binding**

This approach intends to provide dynamic adaptability by using rules for the selection of a concrete service that provides a functionality required by the process. It is assumed that each process instance would require a different partner concrete service, such that a service is based on the process execution environment, user, or another context. In addition, with several services offering similar functionality, late binding solutions enable the runtime selection of the most suitable service (e.g., having the best quality attribute) or runtime replacement of a misbehaving service.

One framework that demonstrates this approach is SCENE (Colombo et al., 2006). It extends the standard BPEL by providing rules to guide the binding of a process

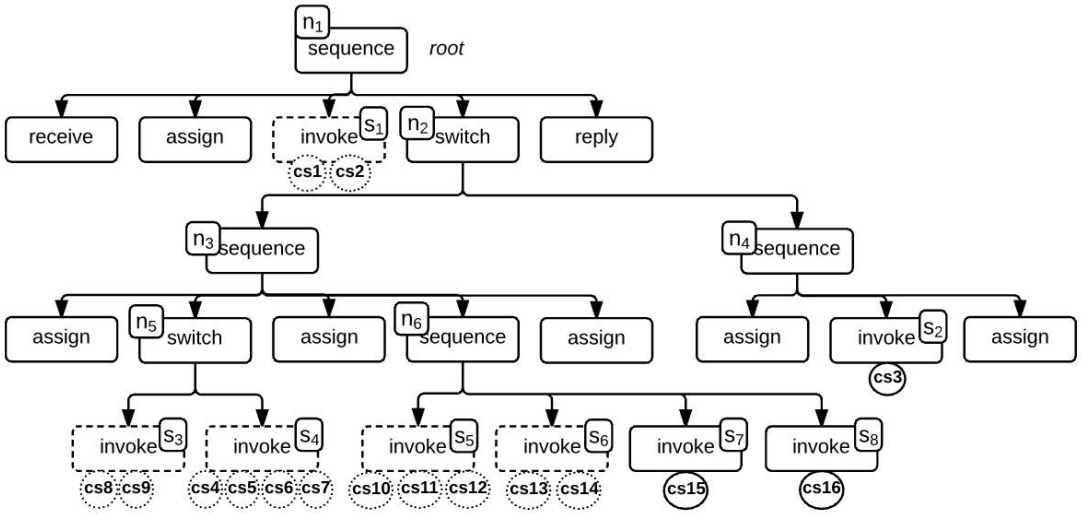



Figure 3.9: The travel booking process with late binding capability

to partner services. The rules define the constraints, policies, and preferences in the selection or changing of services at runtime. It uses a proxy service that is bound to an <$invoke$> activity. All <$invoke$> activities having the rule-enabled tag set to true, are bound to specifically instantiated proxies. Hence, a designer can leave the concrete partner service of an <$invoke$> activity undefined. At runtime, when the execution of the process reaches the <$invoke$> activity, the operation offered by the proxy bound to that activity is called. The proxy enables the activation of the binding rules that are defined in a rule engine. The dynamic adaptability here, is not totally changing the composition logic, but by selecting/changing concrete partner services. This capability is similar to the semantics of an *Around* variability, thus, it can still be aligned with structure variability. However, with respect to the given set of potential structure variabilities of the travel booking process, this framework cannot accommodate such assumptions. Instead, new assumptions that apply appropriately to this framework are formulated. Referring to Figure 3.9, it is assumed that the <$invoke$> activities with more than one partner service are tagged with *Around* variability. Technically, their rule-enabled tags are set to "true" and yet to be bound to a concrete service. This is in

contrast to a non-variable invoke activity which is already bounded to a concrete service, *e.g.*, $s_2$. Considering all the <$invoke$> activities as variation points, $MSVD_\pi$=0.21 and $ESVD_\pi$=0.23 are calculated for the process.

$MSVD_\pi$

= (*SVD*<$invoke_{s1}$> +...+ *SVD*<$invoke_{s8}$>) / 8 = 0.21.

$ESVD_\pi$

= *SVD*<$n_1$> = (*SVD*<$invoke_{s1}$> + *SVD*<$n_2$>) / 2 = 0.23, where

*SVD*<$n_6$> = (0.33 + 0.33 + 0 + 0) / 4 = 0.165

*SVD*<$n_5$> = (0.165 + 0.165) = 0.33

*SVD*<$n_4$> = 0

*SVD*<$n_3$> = (0.33 + 0.165) / 2 = 0.2475

*SVD*<$n_2$> = (0.2475 * $\frac{1}{2}$) + 0 = 0.1238.

## 3.4 Discussion

This section presents the use of the metrics in comparing processes and frameworks, as well as in determining the runtime adaptability of a process. It also discusses the integration of adaptability measures in performing design decisions.

### 3.4.1 Comparing Processes

The main motivation of the metrics is to enable the comparison of processes regardless of size and specification framework, which become possible for two reasons. First, the metrics rely on the standard BPEL constructs and their behaviours. Despite the differences in the adaptability mechanisms implemented in the various frameworks, it is observed that the original BPEL language elements are rarely changed. This raises the applicability of the metrics generally, to all processes specified from any

BPEL-based framework. The examined frameworks in Table 3.1, except MODAR, can basically produce a specification for the travel booking process similar to that shown in Figure 3.7 on page 143(b). Second, the aggregation of the atomic adaptabilities into a single value that describes the entire process facilitates direct comparison. Hence, process comparison becomes reasonably straightforward, especially when using binding variability, i.e., *MBV* or *EBV*, where the main concern is identifying the candidate services associated with every <$invoke$> comprising the process.

Table 3.1: The variabilities and variation points of the adaptive BPEL frameworks

| | Adaptability mechanism | Variable constructs | Variabilities per variation point | Supports changes to process flow |
|---|---|---|---|---|
| AO4BPEL (Charfi & Mezini, 2007) | aspect injection | <$invoke$>, <$receive$>, <$reply$> | 3 ($Before$, $Around$, $After$) | yes |
| MODAR (J. Yu et al., 2015) | aspect injection | <$sequence$> | 3 ($Before$, $Around$, $After$) | yes |
| BPELnAspects (Karastoyanova & Leymann, 2009) | aspect injection | <$invoke$> | 3 ($Before$, $Around$, $After$) | yes |
| Rosenberg and Dustdar (Rosenberg & Dustdar, 2005) | message interception | <$invoke$>, <$receive$>, <$reply$> | 2 ($Before$, $After$) | yes |
| BPEL+Rules (Paschke & Teymourian, 2009) | explicit integration | <$invoke$>, <$receive$>, <$reply$> | 1 ($Before$) | yes |
| SCENE (Colombo et al., 2006) | late binding | <$invoke$> | 1 ($Around$) | no |

However, some framework implementation constraints have to be considered when comparing process adaptability based on structure variability. The differences in adaptability implementation among the frameworks of specification are relevant to understanding the values derived by the metrics. Table 3.1 summarises the main observations. *First*, the variable constructs supported by the frameworks are different. These constructs are the process variation points where variabilities can be specified. For instance, AO4BPEL supports variability specification to <$receive$>, <$invoke$>, and <$reply$>, while MODAR specifies variability on a <$sequence$> construct. *Second*, there are differences in the number of variabilities that can be specified in a variation point. Among the frameworks, those implementing aspect injection allow three variabilities, which are interpreted as *Before*, *Around*, and *After*. Others support only one or two of these variabilities. The number of supported variabilities should be the basis of the reference value in the metrics. So far, in the literature, the highest number of variability that can be specified to a variation point is determined as *three*. It is expected that the reference value will change when a new framework that supports a higher number of structure

variability is introduced.

Furthermore, the designer's choice of the reference value is determined by the *scope of use* of the metrics. A three-level usage scenario of the metrics is suggested in the evaluation and/or comparison of adaptability: *i*) *across-frameworks* — its use among processes specified from various frameworks, *ii*) *framework-specific* — for (different) processes specified from the same framework, and *iii*) *process-specific* — for variants of a particular process. To compare processes specified from a particular framework, a reference value that is common to the processes but only becomes valid to such a framework is used. Similarly, for processes coming from different frameworks, a reference value is determined by considering all concerned frameworks. Meanwhile, a reference value specific to a particular process might produce a more meaningful variability degree for that process. This can be useful when deciding a suitable process configuration. However, by using a process-specific reference value, direct comparison with other processes may no longer apply, thus depriving the metrics of one of its primary purposes.

Table 3.2: *Across-frameworks* adaptability degrees for the travel booking process based on structure variability

| Process specification | $MSVD_\pi$ | $ESVD_\pi$ | Applicable variabilities |
|---|---|---|---|
| AO4BPEL (Charfi & Mezini, 2007) | 0.20 | 0.12 | 5 ($RQ_1$, ..., $RQ_5$) |
| MODAR (J. Yu et al., 2015) | 0.21 | 0.21 | 5 ($RQ_1$, ..., $RQ_5$) |
| BPELnAspects (Karastoyanova & Leymann, 2009) | 0.20 | 0.12 | 5 ($RQ_1$, ..., $RQ_5$) |
| Rosenberg and Dustdar (Rosenberg & Dustdar, 2005) | 0.10 | 0.09 | 3 ($RQ_1$, $RQ_4$, $RQ_5$) |
| BPEL+Rules (Paschke & Teymourian, 2009) | 0.03 | 0.08 | 1 ($RQ_1$) |

Table 3.2 presents the results of deriving adaptability degrees for the travel booking process. Given the process and a set of applicable potential variabilities ($RQ$s), the adaptability concerning the structure variability of its various specifications can be evaluated. This answers questions such as which specification of the travel booking

process is more adaptable assuming the potential variabilities, or, which process specification can accommodate the future requirements changes defined by the $RQ$s. Only a process specified from frameworks using aspect injection, *i.e.*, the first three rows, can accommodate all the five given variabilities. As expected, these processes have higher variability degrees than those with a smaller number of applicable variabilities. The SCENE process, which uses the late binding mechanism, is not included in the comparison because the semantics of the metrics in such specification is different, i.e., the variability of <$invoke$> pertains to selecting a concrete service. As shown in Table 3.1, SCENE does not support changes in the process workflow. Hence, none among the given future variability assumptions (i.e., the $RQ$s), can apply to late binding.

The examined frameworks, except MODAR, do not necessarily change the travel booking process transformation in Figure 3.7 on page 143(b) when implementing adaptability. Hence, the same BPEL specification is used for those frameworks. Looking at the aspect-injection-based frameworks, similar adaptability degrees are computed for the processes specified from AO4BPEL and BPELnAspects, as they both support all the given potential variabilities using the same process specification. Although MODAR also supports all the given variabilities, a different result is calculated because of its distinct specification of the process. Besides, the MODAR version of the process poses higher adaptability degrees especially for the *ESVD* because of the smaller number of structured constructs involved in the calculation. It is shown in Figure 3.8 on page 150(b) how MODAR specifies a process and the variabilities, making the aggregation of atomic variability degrees performed at one level of the activity tree.

It is also worth emphasising that different processes regardless of their sizes and specification frameworks, are comparable in an across-frameworks evaluation. The same set of variable constructs has to be considered. That is, in addition to setting a common reference value pertaining to the highest allowed number of variabilities that can be specified to a variable construct. However, when dealing with processes from

a particular framework, it may be necessary to have a *framework-specific* evaluation. The metrics become more cognisant of the framework constraints. This produces a more meaningful adaptability degree for a process. The calculation for adaptability becomes more precise with respect to the capabilities allowed by the framework. Take for example the BPELnAspects process in the case study; despite supporting the specification of a maximum three variabilities per variation point, the BPELnAspects framework allows variabilities to be specified only to the <$invoke$> constructs. If the scope of evaluation is on processes within this framework, it becomes less relevant to include other constructs (e.g., <$receive$> or <$reply$>), as variation points to be regarded in the metrics. Referring to the following calculations of the adaptability degrees of the travel booking process with only the <$invoke$> activity as variable: $MSVD_{\pi}$ = (*SVD*<$invoke_{s1}$> +...+ *SVD*<$invoke_{s8}$>) / 8 = 0.25, and $ESVD_{\pi}$ = *SVD*<$n_1$> = (*SVD*<$invoke_{s1}$> + *SVD*<$n_2$>) / 2 = 0.24.

As expected, there is an increase in the adaptability degrees from these framework-specific calculations compared to the across-framework measurements derived for the same BPELnAspects process shown in Table 3.2 on page 158. If every <$invoke$> activity is specified with three variabilities, a value of 1 will be eventually derived as the *MSVD* or *ESVD* of the process. A framework-specific evaluation ascertains that the typical value of 1 is the maximum adaptability possible for the process. This becomes otherwise (i.e., maximum adaptability < 1) when there are constructs considered variable in the metrics, but are not supported as variation points by the process framework. An ascertained range of process adaptability (i.e., [0,1]), facilitates the outright interpretation of derived adaptability degrees. For example, an *MSVD* close to 0 means that variable constructs can be much more adaptable, such that additional variabilities can be specified. While a value close to 1 means the process is nearly specified with the maximum allowable variabilities.

Moreover, a framework-specific adaptability evaluation should not only be sensitive

to the variation points but also to the variabilities supported by the framework. As shown in Table 3.1 on page 157, the frameworks have differences in the number of variabilities allowed per variation point. For instance, frameworks that utilise message interception generally specify a maximum of two variabilities. In this case, when evaluating adaptability of processes specified from such frameworks, the reference value is adjusted to $R$ = 2. Similarly, adjusting $R$ = 1 can be done when evaluating processes particularly specified from frameworks allowing only one variability, such as those using either explicit integration or late binding mechanisms. This supports the concern in maintaining 1 as the adaptability degree of a process, when all the variation points are specified with maximum variability. Overall, the framework-specific evaluation restricts the scope of the metrics, but the generated results can become more significant in the direct interpretation of adaptability degrees.

### 3.4.2 Comparing Frameworks

An ideal structure adaptability happens when every element of the process becomes a variation point specified with the maximum allowed variabilities. This means a variability concern of any type can be defined anywhere within the process specification. The process becomes *fully-adaptable* such that $RMSV_{\pi}$ = 1. For instance, the process in Figure 3.7 on page 143(b) can be fully-adaptable if variabilities can be specified to <$assign$>. However, this may produce significant implementation and operational overheads for the process. For example, specified variability placeholders on places that are not expected to change are unnecessary, and would just add to the complexity that the process has to handle. Likewise, it increases the expected drawbacks to other quality attributes such as performance and response time (Perez-Palacin et al., 2014; J. Yu et al., 2015). Such implications may have bound frameworks in their implementation of full-adaptability besides the technical limitations.

The potential level of structure variability that a process may achieve significantly depends on the framework of specification. As observed from the examined frameworks, the adopted adaptability mechanism poses constraints that affect variability. For instance, the differences in either the allowed number of variabilities or the process elements that can become variation points influence the specification's adaptability. It is then important to know how adaptable a certain process can become in a given specification framework. To this aim, the applicability of deriving the maximal adaptability through the *RMSV* metric is demonstrated in the succeeding discussion.

Referring to the travel booking process in Figure 3.7 on page 143(b), a question is formulated: what would be its maximal adaptability in a framework utilising late binding such as that of SCENE? Deriving the maximal adaptability considers a framework's constraints in implementing variability. For example, SCENE supports only the specification of an *Around* variability to the <$invoke$> activities. Consequently, only the <$invoke$> activities are variable, and the reference value $R$ = 1, which implies $R^*$<$invoke$> = 1. Hence, *SVV**<$invoke$> = 1. The calculation derives:

*RMSV*$\pi_{(SCENE)}$
= *SVD**<$n_1$>
= (*SVD**<$receive$> + *SVD**<$assign$> + *SVD**<$invoke_{s1}$> + *SVD**<$switch_{n2}$> +
*SVD**<$reply$>) / 5
= (0 + 0 + 1 + 0.365 + 0) / 5
= 0.27.

The derived *RMSV*$_\pi$ reveals the maximum adaptability that the travel booking process $\pi$ can achieve in its present form. This is considering the adaptability implementation capabilities of the SCENE framework from which $\pi$ is specified. The awareness of designers about this maximal adaptability supports a process-specific evaluation that may lead to two ways of improving such potential adaptability degree:

first, re-specifying the process to either decrease the number of non-variable constructs or increase the number of variable constructs, and second, lessening the depth of the process specification structure. Hence, a version of a process specification with the best adaptability potential can be determined.

Furthermore, the need for an adaptability-intensive composition may require the designer to survey different specification frameworks and select among them the one that can provide the highest level of potential adaptability for a certain process of concern. For instance, given the travel booking process, we can derive its *RMSV* in every framework of specification. However, we need to identify a global reference value $R$<$a$> to be used across all frameworks. Such value should be the highest $R$ for an activity $a$ among the different process specifications. By varying the previous *RMSV* derivation of the travel booking process SCENE version, an across-frameworks evaluation using a global reference value $R$<$a$> = 3 is demonstrated as follows. The *SVD*$^*$ of the variable element <$invoke$> becomes *SVD*$^*$<$invoke$> = $SVV^*$<$invoke$> / $R$<$invoke$> = $R^*$<$invoke$> / $R$<$invoke$> = 1/3 = 0.33. Noteworthy here is the difference between the $R^*$<$invoke$> and $R$<$invoke$>. The former is the maximum number of variability that can be imposed on a variation point by the particular framework while the latter refers to the global reference value for such variation point. Deriving the *RMSV* for the whole process:

*RMSV*$\pi_{(SCENE)}$
= *SVD*$^*$<$n_1$> = (*SVD*$^*$<$receive$> + *SVD*$^*$<$assign$> + *SVD*$^*$<$invoke_{s1}$> +
*SVD*$^*$<$switch_{n2}$> + *SVD*$^*$<$reply$>) / 5
= (0 + 0 + 0.33 + 0.121 + 0) / 5
= 0.09.

By adopting a global reference value, the derived *RMSV*$\pi_{(SCENE)}$ for the travel booking process is comparable across different specification frameworks. A similar

derivation approach is applied to the process specifications from the other frameworks – the summarised result is shown in Table 3.3. Hence, questions such as which framework will provide the highest level of structure variability for the travel booking process, or which framework will enable a process specification to accommodate the most number of variability concerns, can now be answered. As the frameworks can similarly support the original process specification, the specification in Figure 3.7(b) is used in the derivation of their *RMSV*. Accordingly, the frameworks can implement adaptation without necessarily changing that BPEL process specification.

It is however, shown in Table 3.3, that the baseline specification in Figure 3.7(b) does not apply to MODAR; a different BPEL specification is used for MODAR. The MODAR framework derives its process specification by transforming its weave model into BPEL. Consequently, variable activities are determined from the weave model depending on where the anticipated variability is specified. In this case, a new variability that is yet to be specified somewhere in the process cannot be accommodated. However, MODAR has an *Advanced Adaptability* transformation option that sets all the weave model's variable activities to full-adaptability. That means a predetermined variable activity can be dynamically specified a *Before*, *Around*, or *After* variability. To derive its *RMSV*, the specification illustrated in Figure 3.8 on page 150(b) is utilised, particularly the top level constructs in the BPEL activity tree. Notice that not all the sequence activities are considered variable (i.e., <$sequence_{s3}$> is not variable). Hence, $RMSV\pi_{(MODAR)}$ = $SVD^*$<$sequence_{root}$> = (0 + 1 + 1 + 0 + 1 + 1 + 1 + 0) / 8 = 0.63.

Table 3.3: *Relative Maximal Structure Variability* of the travel booking process

| Specification framework | $RMSV_\pi$ | Maintained baseline specification |
|---|---|---|
| AO4BPEL (Charfi & Mezini, 2007) | 0.67 | yes |
| MODAR (J. Yu et al., 2015) | 0.63 | no |
| BPELnAspects (Karastoyanova & Leymann, 2009) | 0.27 | yes |
| Rosenberg and Dustdar (Rosenberg & Dustdar, 2005) | 0.45 | yes |
| BPEL+Rules (Paschke & Teymourian, 2009) | 0.22 | yes |
| SCENE (Colombo et al., 2006) | 0.09 | yes |

The difference between the Effective Structure Variability Degree *ESVD*$\pi$ and Relative Maximal Structure Variability degree *RMSV*$\pi$ needs emphasised. Both measure the dynamic adaptability of a process $\pi$ based on changes in process workflow but have to be interpreted differently. On the one hand, *ESVD*$\pi$ indicates the level of structure variability of $\pi$ given the currently specified variabilities. It is assumed that the process designer anticipates the possible runtime changes and specifies a variability placeholder to places where such changes are expected. Adaptability is understood as the adjustments or modifications of such specified variabilities. In terms of the process in Figure 3.7(b), *ESVD*$\pi$ measures its adaptability based on currently specified *RQ*s (e.g., the adaptability brought by either disabling $RQ_1$ or modifying the activities defined in $RQ_5$). On the other hand, *RMSV*$\pi$ measures the maximum degree of structure variability that a process can achieve in a particular specification framework. Adaptability is interpreted as the capability of the process to be specified with runtime variabilities. For instance, a variability of any type (i.e., $Before$, $Around$, and $After$) is dynamically specified to any variable activity. Referring to the same process in Figure 3.7(b), *RMSV*$\pi$ does not regard the predetermined variabilities. Instead, its concerns are the potential variabilities that the process specification can accommodate (e.g., specifying a new requirement $RQ_7$:$Around$ to the activity <$invoke_{s5}$>).

As *RMSV* is sensitive to a framework's capability with regards to implementing adaptation for a process specification, this metric can be used to assess not just processes, but also, their frameworks of specification. For instance, the derived *RMSV* results in Table 3.3 imply that the AO4BPEL framework can offer the most dynamically adaptable specification for the travel booking process in terms of structure variability. Furthermore, *RMSV* can reveal and compare the degree of a framework's *expressiveness* in terms of feature sets. "Feature sets" means both the supported variability points and variability concerns in a framework. For instance, Table 3.3 shows that the AO4BPEL and MODAR are more expressive than the other frameworks, i.e., both support a higher

number of feature sets that are necessary to express more desired adaptability of the given process.

### 3.4.3 Runtime Process Adaptability

It may be necessary for designers to track the adaptability of a process as it executes, especially with processes composed of a long sequence of activities. Given a particular activity designated as an execution point, the *remaining adaptability* of the process can be computed. The remaining adaptability is the aggregated adaptability of the process activities pending execution. This may help identify at which execution point the adaptability increases or decreases as the process runs. For instance, the portion that needs improving adaptability can be located, especially in large processes. The perspective of measuring adaptability applies an element-driven approach. The runtime behaviour of a process also follows a flow of element-based execution. Considering some elements as execution points, an adaptability degree for the process at every execution point disregarding the already executed elements can be derived. Thus, a set of adaptability degrees can be obtained, wherein each element of the set associates to the runtime adaptability of the process at a certain execution point.

Figure 3.10 presents the runtime adaptability degrees derived at each execution step considering top-level activities comprising the root node of the travel booking process. The variations of structure variability degrees $ESVD\pi$ in three specifications of the process are shown. For example, the runtime structure variability of the AO4BPEL specification is a set $ESVD^{@}_{\pi(AO4BPEL)}$ = [0.12, 0.16, 0.16, 0.07, 0], where each element is respectively associated to the execution points $ESVD_{\pi<receive>}$, $ESVD_{\pi<assign>}$, $ESVD_{\pi<invoke_{s1}>}$, $ESVD_{\pi<switch_{n2}>}$, and $ESVD_{\pi<reply>}$. The three adaptability transitions observed between two adjacent execution points are described, and referred to as the transition from point $A$ to $B$ in the figure. An increase in *ESVD* from execution point


Figure 3.10: Variations of runtime adaptability

$A$ to point $B$ means that *ESVD*$<_A>$ is either 0 or less than the mean of the *ESVD*s of all the execution points succeeding $A$. In contrast, a decrease in *ESVD* from point $A$ to $B$ indicates that either *ESVD*$<_B>$ is 0 or *ESVD*$<_A>$ is greater than the mean of the *ESVD*s of all the execution points after $A$. If neither increase nor decrease occurs, it means *ESVD*$<_A>$ or *ESVD*$<_B>$ are non-variable constructs, or *ESVD*$<_A>$ is equal to the mean of the *ESVD*s of all the execution points succeeding $A$.

### 3.4.4 Considering Adaptability in Design Decisions

The impact of adaptability to the satisfiability of another quality attribute can affect design decisions. Table 3.4 illustrates the typical relationships between adaptability and a quality attribute (see Perez-Palacin et al., 2014). The rows indicate three cases for a quality attribute $q$ when adaptability increases: *i*) $q$ tends to increase; *ii*) $q$ tends to decrease; or, *iii*) $q$ is not affected. The columns indicate the formulation of a target requirement pertaining to $q$, *e.g.*, "availability shall be *higher than* 99%" or "response time shall be lower than 5 seconds". The table indicates that increasing adaptability *helps*, *hurts*, or has *no effect* on the target quality requirement. Generally,

when a relationship between adaptability and $q$ exists (*i.e.*, either *case i* or *case ii*), the alternative process configuration offering the best trade-off is chosen.

Table 3.4: Impact of adaptability on a quality requirement

| When adaptability increases... | Requirement formulation | |
|---|---|---|
| | Higher than | Lower than |
| The quality attribute value increases | Helps | Hurts |
| The quality attribute value decreases | Hurts | Helps |
| The quality attribute is not affected | No effect | No effect |

To illustrate an example, different configurations of the travel booking process shown in Figure 3.7b are examined. The relationship between adaptability (i.e., particularly binding variability) and a quality attribute $q$ := "*cost*" is observed. The overall $q$ of a process configuration by summing the individual $q$ of concrete services $cs$ composing the process is computed. The following cost values for each $cs$: {$cs_1$:40, $cs_2$:25, $cs_3$:50, $cs_4$:25, $cs_5$:35, $cs_6$:25, $cs_7$:30, $cs_8$:15, $cs_9$:25, $cs_{10}$:25, $cs_{11}$:20, $cs_{12}$:15, $cs_{13}$:50, $cs_{14}$:35, $cs_{15}$:25, $cs_{16}$:25} is assumed. Table 3.5 presents the derived adaptability and cost of 10 process configurations where process {a to e}, and {f to g}, are variants of the AO4BPEL and BPEL+Rules specification, respectively. The objective is to select the most adaptable process to satisfy the requirement: "*cost shall be lower than 300 monetary units*".

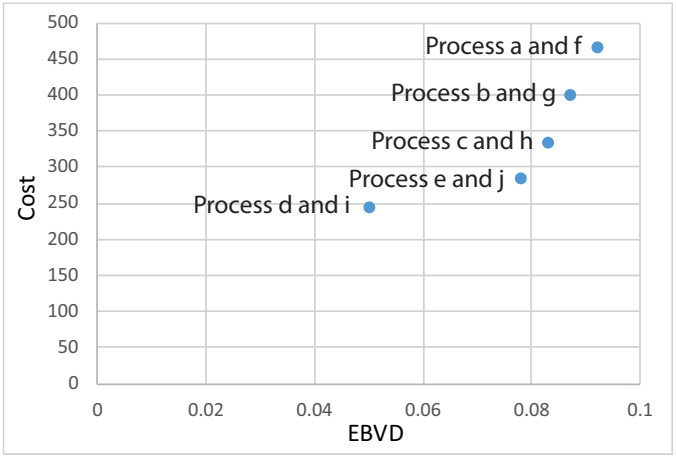



Figure 3.11: Relating adaptability and cost

Table 3.5: Trade-off between adaptability and cost

| Process instance | $ESVD$ | $EBVD$ | $ECAM_A$ | $ECAM_B$ | Cost | Trade-off score (Cost * EBVD) |
|---|---|---|---|---|---|---|
| **a**) Includes all $cs$ | 0.12 | 0.092 | 0.112 | 0.1 | 465 | 42.78 |
| **b**) w/o $cs_8$, $cs_{13}$ | 0.12 | 0.087 | 0.11 | 0.097 | 400 | 34.8 |
| **c**) w/o $cs_6$, $cs_7$, $cs_{13}$, $cs_{10}$ | 0.12 | 0.083 | 0.109 | 0.094 | 335 | 27.805 |
| **d**) w/o $cs_2$, $cs_4$, $cs_5$, $cs_6$, $cs_8$, $cs_{10}$, $cs_{11}$, $cs_{13}$ | 0.12 | 0.05 | 0.099 | 0.071 | 245 | 12.25 |
| **e**) w/o $cs_4$, $cs_5$, $cs_6$, $cs_{10}$, $cs_{11}$, $cs_{13}$ | 0.12 | 0.078 | 0.107 | 0.091 | 285 | 22.23 |
| **f**) Includes all $cs$ | 0.083 | 0.092 | 0.086 | 0.089 | 465 | 42.78 |
| **g**) w/o $cs_8$, $cs_{13}$ | 0.083 | 0.087 | 0.084 | 0.086 | 400 | 34.8 |
| **h**) w/o $cs_6$, $cs_7$, $cs_{13}$, $cs_{10}$ | 0.083 | 0.083 | 0.083 | 0.083 | 335 | 27.805 |
| **i**) w/o $cs_2$, $cs_4$, $cs_5$, $cs_6$, $cs_8$, $cs_{10}$, $cs_{11}$, $cs_{13}$ | 0.083 | 0.05 | 0.073 | 0.060 | 245 | 12.25 |
| **j**) w/o $cs_4$, $cs_5$, $cs_6$, $cs_{10}$, $cs_{11}$, $cs_{13}$ | 0.083 | 0.078 | 0.082 | 0.080 | 285 | 22.23 |

*$ECAM_A$ uses the weighting 0.7 and 0.3 for $ESVD$ and $EBVD$ respectively and $ECAM_B$ uses the weighting 0.3 and 0.7 for $ESVD$ and $EBVD$

Figure 3.11 plots the $EBVD$ in relation to cost of the process configurations. Table 3.4 shows that the adaptability and cost relationship belongs to *Hurts*, since when adaptability increases, cost also increases; the cost requirement is formulated as *lower than*. In this case, it is not practical to just select the most adaptable process. Referring to Table 3.5, process $d$, $e$, $i$, and $j$, satisfy the requirement. The one with the highest trade-off score (i.e., $EBVD \cdot cost$) is selected, however, process $e$ and $j$ have equal trade-off scores. Hence, to further reinforce a decision, the $ECAM$ metric values are used to compute a trade-off score since it is also desirable to consider structure variability. It is noted that ECAM provides a single value for binding and structure variability, and the weights of ECAM can be set according to the adaptability priority. Finally, process $e$ is determined to provide the better trade-off score (i.e., $ECAM_A \cdot cost$).

## 3.5 Automated Support Tool and Comparability Evaluation

A tool called "AdaptQ" has been developed to perform the automatic calculation of the adaptability metrics, particularly the aggregated ones describing the entire process. It reads an input XML file that contains the abstract code specification of the process to evaluate. This code abstraction is necessary to uniformly represent the process specifications from various adaptive BPEL frameworks. Each process element is represented, together with its relevant attributes (i.e., the number of associated variabilities and concrete services needed in the calculation). The tool, however, does not support the automated translation of process specifications into the code abstraction. Figure 3.7 on page 143(c) shows a snapshot of the travel booking process specification, representing the one specified from the AO4BPEL framework. The attributes within the process element tags indicate the values needed in the adaptability evaluation. Line 1 pertains to the root node that integrates the reference value *vprefval=3* for the *SVD*. Likewise, *SVDWeight* and *BVDWeight* define the associated weights that are needed for computing the combined metrics. This method of assigning varying weights is a design choice and facilitates the expression of preferences over the represented adaptability dimensions in the combined metrics. The *vpr* refers to the value associated to $R^*$ when computing $SVD^*$ for the *RMSV*. The boolean attribute *vp=y* means the element is a variation point (i.e., a variable atomic element w.r.t. structure variability). Otherwise, *vp=n*. The attribute *vr* and *cs* respectively indicate the number of associated structure variabilities and concrete services. Specifying these attributes makes the metrics flexible as discussed in Sections 3.4.1 and 3.4.2. It enables a designer to set values for the aforementioned attributes according to the identified characterisations of the process, constraints of the implementing framework, weighting preferences, and purpose of the evaluation (i.e., across-frameworks, framework-specific, or process-specific).

Furthermore, it is important to investigate how well the metrics perform with regards to comparability, which is an important metric property, denoting an *order* relation between the measurement values (Kitchenham, Pfleeger & Fenton, 1995; Reinecke et al., 2010; Lenhard et al., 2015).

### 3.5.1 Discriminative Power

*Discriminative power* (Zuse, 1998) describes the ability of the metrics to distinguish between different processes. The more sensitive a metric in differentiating between different processes, the better it can be used for quality comparison and ranking (Lenhard et al., 2015). The discriminative power of the *mean* and *effective* metrics on a given set of process specification variants are explored and compared in this section. The variants are generated through two types of process variations: *i*) varying the number of variabilities (*resp.* partner services); or, *ii*) varying the number of variation points (*resp.* <$invoke$> activities).

For the binding variability metrics, a total of 192 unique variants of the process in Figure 3.7 on page 143(b) are evaluated. These variants are minimally different from the one in Figure 3.7(b) to generate the closest possible differences between the binding variability measures. Firstly, three sets of processes are generated – those with 15, 16, and 17 partner services respectively (the process in Figure 3.7(b) has 16). Each set has 32 variants. Every process variant within a set uses the same number of partner services that are randomly specified to the <$invoke$> constructs, where each <$invoke$> has at least one partner service. The process structure and its constructs remain unchanged in all variants. Secondly, another three equivalent sets of processes are generated, those with 7, 8, and 9 <$invoke$> activities respectively (the process in Figure 3.7(b) has 8). Every variant within a set uses the same number of <$invoke$> activities that are randomly moved from one structured activity to another. The given number of partner

services (i.e., 16) and the process structured activities remain the same in all variants.

For the structure variability metrics, a minimal process variation to generate the least possible differences between variability degrees is assumed. A total of 192 unique variants of the process shown in Figure 3.7(b) is evaluated. Firstly, three equivalent sets of processes that are respectively specified with 5, 6, and 7 variabilities are generated (the process in Figure 3.7(b) has 6). Each set has 32 variants. For each process variant in each set, the same number of variabilities is randomly specified to the variation points, so each variation point has at most, 3 variabilities. There are no changes made in the given process structure and constructs. Secondly, another three equivalent sets of processes are generated, those with 9, 10, and 11 variability points respectively (the process in Figure 3.7(b) has 10). Each variant within a set uses the same number of variability points, randomly placed within the structured activities. The given number of variabilities (i.e., 6) and the process structured activities remain the same in all variants.

Table 3.6 shows the descriptive statistics of the computed values per metric. The small values for the standard deviation show the dense distribution of the derived measures as a result of the slight variations made among the processes. The discriminative power is determined as the percentage of unique values (i.e., each value is rounded off to three decimal places), derived by a metric from the data set. It can be observed that for both binding and structure variability, the effective adaptability metrics have a significantly higher discriminative power than the mean ones. This shows the sensitivity of the effective adaptability metrics to the runtime behaviour of each structured construct. Hence, two process variants even with the same structure and number of variabilities (*resp.* partner services) may differ in their effective adaptability metrics if: *i*) the variabilities (*resp.* partner services) are specified differently among variation points (*resp.* <$invoke$> activities), and, *ii*) the variation points (*resp.* <$invoke$> activities) belong to different structured constructs. The mean adaptability metrics are only able to distinguish the adaptability of processes having different number of variation points

(*resp.* <$invoke$> activities) or variabilities (*resp.* partner services).

Table 3.6: Statistics of the evaluated process variants and the derived discriminative power of the metrics

| | | | Statistics | | |
|---|---|---|---|---|---|
| | No. of variants | Metric | Mean | Std. deviation | Discrimin-ative power |
| Binding variability | 192 | *EBV* | 2.146 | 0.820 | 0.55 |
| | | *MBV* | 2.011 | 0.165 | 0.03 |
| Structure variability | 192 | *ESVD* | 0.179 | 0.104 | 0.44 |
| | | *MSVD* | 0.201 | 0.022 | 0.03 |

### 3.5.2 Indifference to Process Size

A well-functioning metric should perform similarly regardless of the process size (Lenhard et al., 2015). Differently sized processes entail differences in both the number of atomic or structured elements and the way the elements are structured. Applying an across-frameworks evaluation, the consistency of the metrics in dealing with these processes is examined. Such consistency justifies the comparison between different processes as discussed in Section 3.4.1. Hence, 36 processes of various sizes and forms are randomly created by cloning either a portion of, or the whole process specification in Figure 3.7 on page 143(b). The largest process has a size of 134 (i.e., a combined number of atomic and structured elements), while the smallest one has six. For each process, an assumed number of variabilities or partner services is randomly specified to the variation points or <$invoke$> elements respectively. The metrics for each process are derived, together with the relevant process properties such as size, number of variabilities and variation points, and number of partner services and <$invoke$> activities. Utilising the evaluation results from all the processes, Table 3.7 shows the correlation coefficient between a metric and a process property. Except for in the last column, there is no significant linear correlation between the metrics and the other four process properties. This means that when individually considering any of these four properties, a smaller or

larger value does not necessarily imply a lower or higher adaptability. Moreover, the negative correlation values for the binding variability metrics show that higher measures are likely derived from the ones with smaller sizes in the evaluated processes. Similarly, higher binding variability measures are likely derived from processes with a smaller number of partner services, *invoke* elements, or structured elements. This, however, does not contradict the fundamental notion of binding variability wherein the more partner services exist, the higher the binding variability. It must be likely that the larger processes have smaller overall ratio of partner services to <$invoke$> elements, when compared to the smaller ones.

Table 3.7: The correlation coefficient of the metrics with the process properties

| Metric | Process size | Concrete services (*cs*) | <invoke> elements | Structured elements | Mean no. of *cs* |
|---|---|---|---|---|---|
| *EBV* | -0.47 | -0.39 | -0.49 | -0.45 | 0.99 |
| *MBV* | -0.51 | -0.43 | -0.54 | -0.49 | 1.00 |
| | | Variabil-ities (*vr*) | Variation points (*vp*) | | Mean no. of *vr* |
| *ESVD* | 0.13 | 0.35 | 0.08 | 0.12 | 0.93 |
| *MSVD* | 0.25 | 0.47 | 0.20 | 0.24 | 1.00 |

Contrastingly, the last column shows very strong to maximum correlation values. Both the mean variability metrics *MBV* and *MSVD* have the maximum positive correlation with the mean number of partner services and the mean number of variabilities respectively. This is expected as both metrics are directly derived from the number of variabilities (*resp.* partner services) relative to the number of variation points (*resp.* <$invoke$> activities). Although the derivation of the effective variability metrics *EBV* and *ESVD* strongly rely on the same relationship (i.e., the proportion of variabilities to variation points), there are other factors considered in their derivation (e.g., the structured constructs). This makes the correlation values become less than the maximum. It is therefore set forth that the consideration of variabilities (*resp.* partner services) against the number of variation points (*resp.* <$invoke$> elements), enables the metrics

to perform consistently even with differently sized processes.

## 3.6 Chapter Summary

The integration of heterogeneous Web-accessible services has introduced an open phenomenon for creating software systems that are deployed in an unpredictable execution context, where changes in the requirements, user preferences, and component services frequently occur. The consideration of adaptability in designing such systems has become necessary in such unstable and volatile environments. Despite the intensive efforts to find solutions to make service compositions adaptable, little attention has been given to the evaluation of such adaptability. This chapter has presented a set of metrics to quantify the adaptability of service compositions that are specified from the adaptive WS-BPEL-based frameworks. The metrics are based on two adaptability dimensions: the *structure* and *binding* variabilities. Structure variability enables runtime changes to the composition workflow, while binding variability allows dynamic binding to concrete partner services. The metrics are evaluated through a case study, utilising variants of a travel booking process specified from different adaptive WS-BPEL-based frameworks found in the literature. The metrics are applicable to a wide spectrum of adaptive service compositions and their frameworks. A support tool has also been developed to automate metrics computation. Through the metrics, adaptability of service compositions and their frameworks can be assessed and compared. This can facilitate design decisions in building flexible and robust services where adaptability is a first-class concern.

# Chapter 4

# Context-aware Requirements Variability for Goal-oriented Adaptability

Requirements variability, which is typically characterised by creating multiple subsets of requirements, has long been advocated in requirements engineering to establish a *problem-space-oriented* approach to software adaptability (Dey & Lee, 2017; Metzger & Pohl, 2014; Galster et al., 2014). Goal-oriented requirements engineering (GORE) (Van Lamsweerde, 2001; Negri, Souza, de Castro Leal, de Almeida Falbo & Guizzardi, 2017) has aimed for such a problem-space-oriented approach to software adaptability where adaptability can be realised from the early phase of software development (i.e., at the stakeholders' level of requirements engineering). Effective adaptability in solution design depends on how well the variability in the problem-space is captured and explored (Dey & Lee, 2017). GORE utilises goal models to define a requirements problem by specifying requirements as goals that represent the needs, desires, and expectations of stakeholders. Hence, a goal model has emerged as a popular tool to capture, refine, and reason about stakeholder goals (E. S. Yu, 1997; Mylopoulos, Chung, Liao, Wang &

Yu, 2001; van Lamsweerde, 2009; Liaskos et al., 2011). The decomposition of goals generates various alternative options to achieve the requirements. Consequently, a goal model represents a large space of variabilities, called *requirements variants*, and each variant describes the requirements to achieve, as well as the solutions needed to reach those requirements. Such solutions (i.e., in the form of tasks or actions), define the subsequent design possibilities for a system[1].

To handle the challenge of requirements variability, GORE employs stakeholders' preferences as criteria to determine appropriate solutions from the space of variabilities. Preference-based goal models have previously been proposed to support the goal-oriented approach of requirements specification and analysis (Liaskos et al., 2011; van Lamsweerde, 2009). Such models classify goals into *hardgoals* – the mandatory requirements to be fulfilled that originate from the root goal, and *softgoals* – the optional requirements often expressed as high-level quality goals. As an example, consider the typical 'Online Ordering' problem. *Manage an Order* is a hardgoal which has to be fulfilled by some means to solve the stakeholders' main problem (i.e., achieving each of its subgoals *Receive Order*, *Process Order*, *Pack Order*, and *Ship Order*), and there can be multiple alternative means to achieve each subgoal. Softgoals, such as *Minimise Risk*, *Low Customer Cost*, *Fast Delivery*, and *Happy Customer*, are the preferences. The softgoals are used as criteria to distinguish the best ones among the alternative means of solving the *Manage an Order* problem. Different alternatives have different degrees of satisfying the softgoals and there are alternatives that might not meet some or even any of the softgoals. For instance, between two alternative means to achieve the goal *Ship Order*: *Ship Express* or *Ship Standard*, the option to use express shipping over the standard one could satisfy the goal *Fast Delivery* but does not meet *Low Customer Cost*, or vice versa. Moreover, softgoals may not be equally important, and to express

[1]The term *system-to-be* instead of *system* is used in some literature, to emphasise the potential system to be developed based on the current requirements

prioritisations, qualitative or quantitative valuations are associated with such goals. Generally, solutions that meet the softgoals, or the prioritised softgoals with better degrees of satisfaction, are more desirable. In the case of multiple conflicting softgoals when it becomes impossible to optimise every criterion, a better solution depends on which one satisfies the most appropriate trade-off.

The use of goal models in both specifying goal decomposition and mapping the correlations between alternatives and preferences has been an effective realisation approach to requirements variability (Letier & Van Lamsweerde, 2004; Liaskos et al., 2006a; Ernst et al., 2010; Ali et al., 2013). The *OR-decomposition* of goals has become the main source of variability, shaping a goal model to be a robust representation of the potential variabilities of a system (C. M. Nguyen et al., 2018; Hui et al., 2003; Liaskos, Lapouchnian, Wang, Yu & Easterbrook, 2005; Mylopoulos et al., 2001). However, two other implicit but essential sources of requirements variability have received less attention in the literature: *context variability* and *preference variability*. These two variabilities are correlated with their core element *context*, referring to facts that capture the system's operational environment. On the one hand, context variability describes the changing nature of a system's operational environment where the fulfilment of a goal is intended. Although traditional goal modelling has attempted to represent various instances of a problem in a particular domain, its assumption of a uniform environment for a system ignores context variability as an important source of requirements variability. Goals, and the means to achieve them, can be determined and constrained by context changes. These context-dependent goals are referred to as *contextual goals*. On the other hand, preference variability describes the changing prioritisations of stakeholders towards their goals. Preferences[2] are not fixed and often vary, as different stakeholders can have different preferences in different

[2]From this point forward, a *preference* refers to stakeholders' expression of prioritisation over hardgoal alternatives or softgoals.

situations. Such varying preferences are called *contextual preferences*. Unlike the OR-decomposition of goals (i.e., *intentional variability* [Liaskos et al., 2006a]), neither context variability nor preference variability is an explicit representation of variability, but both can significantly constrain intentional variability. Suppose there is a goal *Bill Customer* in the 'Online Ordering' problem. Although two alternative means to that goal are defined – *Provide Discount* or *Charge Full Price* – it is sometimes limited to a single option since providing a discount is only adopted on large orders. Context variability, in this case, influences either the stakeholder goals to achieve, or the possible ways to fulfil those goals. Meanwhile, the softgoal *Fast Delivery* is prioritised over *Low Customer Cost* when the order is perishable, while the opposite might happen in another situation. Moreover, for a requirements problem with a large set of softgoals, stakeholders may prioritise only a subset of these goals and such prioritisation varies with contexts. Given these situations, how are context variability and preference variability defined in the requirements? How do contextual goals influence the realisation of the requirements variability problem? How can contextual preferences be used to obtain the best solutions from a potentially large space of variabilities? Representing the variations of both contexts and preferences in requirements variability analysis remains to be explored.

Existing works acknowledge the significant influence of either context or preferences to requirements variability. On the one hand, context changes may activate or deactivate certain requirements, constrain the applicability of alternatives to achieve an activated requirement, or determine the contribution of an applicable alternative to the softgoals (Ali et al., 2013). On the other hand, preferences affect the selection of alternatives that satisfy requirements. However, in dealing with the requirements variability problem, most works have treated either contexts or preferences independently. Hence, the problem of modelling and reasoning about requirements variability based on both context and preferences has to be further explored. This chapter presents how the relationship between context and preferences is specified. The chapter defines contextual

preferences and their use in deriving solutions for a given requirements problem. It also examines *i*) the relationship between context and requirements, and *ii*) the implication of having both contextual goals and contextual preferences to requirements variability. The chapter elaborates on those contextual preferences and presents an approach that weaves together the variability of both context and preferences with the variability of the requirements. Relationships between context variability and requirements, and relationships between context variability and preferences are simultaneously defined. Capturing and representing both contextual variability and preference variability alongside requirements broadens the scope of requirements variability and expounds the understanding of the problem-space oriented adaptability of software systems.

This chapter presents a framework for: *i*) specifying contextual goals and contextual preferences, and *ii*) utilising both contextual goals and contextual preferences to reason requirements variability. The framework is primarily built upon the existing notions of contextual goals impacting requirements variability (see Ali et al., 2010; C. M. Nguyen et al., 2018). The chapter introduces a *contextual goal model* that specifies relationships between goals and context variations. Likewise, a *contextual preference model* that enhances preferences with contextual information is presented, to capture the changing nature of stakeholders' preferences caused by context variation. Accordingly, a reasoning technique is defined, utilising the contextual goals and preferences to evaluate requirements variants and obtain the best solutions in a given situation. The approach presented in this chapter is essential for the analysis and design of software systems (including service-based systems) that are supposed to reflect both the stakeholders' rationale and adaptability to context changes.

The chapter is structured as follows. Section 4.1 elaborates on the problem statement and Section 4.2 presents both the running scenario and goal modelling style used in the chapter. Section 4.3 presents the specifications for contextual goals and contextual preferences that represent the relationship between context variability and goals. Section

4.3 also presents a context specification in hierarchical form, enabling the abstraction and decomposition of contexts. Section 4.4 demonstrates the roles of contextual goals and preferences in the requirements variability analysis. Section 4.5 presents the support tool and the translation of requirements specifications into the tool's input format, and Section 4.6 discusses the use of the tool to support modelling activities. Section 4.7 presents the application of the framework in various goal models. This section presents a case study that demonstrates an approach for identifying and fixing inconsistencies and modelling errors by detecting inconsistent contextual goals and preferences that become inapplicable and may lead to un-adoptable system implementations. The case study also offers a detailed account of the experiences in enhancing the requirements models and understanding their requirements variability domain by exploring the impact of contextual preferences upon potential solutions. Section 4.8 shows the evaluation of the automated analysis, and Section 4.9 concludes the chapter.

## 4.1 Preliminaries and Problem Statement

One of the empirical realisations of adaptability is software personalisation. Software personalisation, which has been widely studied from the human computer interface (HCI) point of view, is a process that changes the functionality, interface, information content, or distinctiveness of a software system to increase its personal relevance to an individual (Blom, 2000). It is typically seen as a problem of configuration or customisation; both can be system or user-initiated. From the perspective of requirements engineering (RE), which is the focus of this chapter, personalisation is one of the key motivations for requirements variability. Two issues are typically considered in the RE perspective (Liaskos et al., 2005). First, a software system is intended to adapt to the goals, preferences, abilities, and situation of a user. This requires software flexibility and multi-functionality to meet the ever-changing individual needs and desires of a user.

Second, it is expected that a software system has many users and those users are diverse to a certain degree. Hence, it would be impractical to design a unique software for each of them.

The personal medication assistant scenario in Section 4.2 illustrates the aforementioned concerns. Suppose the objective of that system were to promote self-confidence and a feeling of being independent among patients. A self-initiated intake would satisfy such a goal more than a robot-assisted medication. However, in some instances such as when a patient is tired or is having difficulty moving, the robot assistance should be activated. Likewise, when providing services to multiple patients, there may be differences in their individual needs due to differences in their health conditions. For example, a flashed message on a television (TV) screen is not a suitable way to give medication reminders to a visually impaired patient as it would be less noticeable. In this case, alarming the medicine dispenser may be a better alternative. Such a scenario describes the variability of requirements to consider, for achieving the intended purpose of the system according to the individual goals of users. From the lens of requirements variability, this chapter presents a way to facilitate the design of adaptable software systems that can accommodate the changing needs and differences of each intended user as well as possible. With the inevitable reliance of software systems on contextual information, recognising context influence is necessary to better understand requirements and enhance the usability of system functionalities in as many situations as possible.

To be adaptable to the volatile environment conditions, software designs must allow high variability in the behaviours they prescribe. This chapter aims to support the realisation of adaptability from the early phase of systems development, at the stakeholders' level of requirements engineering, as requirements variability essentially dictates the expected system adaptability behaviours. The variability in a solution must reflect the variability of the problem and the latter can be identified at the early requirements engineering phase. Software adaptability analysed from the perspective of requirements

has been advocated in several works (e.g., Liaskos et al., 2012; Ali et al., 2013; Dalpiaz, Giorgini & Mylopoulos, 2013; Dey & Lee, 2017; Angelopoulos, Papadopoulos, Souza & Mylopoulos, 2018), mainly through understanding requirements variability. This early consideration of variability at the level of stakeholders is assumed to bring cost-effective realisation of software adaptability. On the one hand, considering variability in terms of stakeholder goals, characteristics, preferences, and contexts, before actually designing a solution, may increase the chances that the resulting software system will feature the appropriate set of variation points and alternatives. On the other hand, although the variation points and alternatives must accommodate as much as possible the most likely ways that stakeholders fulfil their goals, irrelevant variabilities should be excluded. Decisions can be made early on variabilities that would rather increase the cost and complexity of the system (Liaskos, Lapouchnian, Yu, Yu & Mylopoulos, 2006b). Moreover, errors on adaptability representations have to be detected and fixed early on (i.e., at the requirements models). Uncorrected adaptability related errors, such as inconsistent context specifications and conflicting contextual goals, will be propagated across the next stages of the development process, thus increasing both the development cost and the risk of malfunctioning systems (Ali et al., 2013).

Addressing the adaptability of software systems from the perspective of requirements draws focus to the goal-oriented requirements variability analysis, in which goal modelling is used to explore: *i*) how to derive which requirements to achieve in a given situation or context, as well as which among the alternatives can be used to reach those requirements, and *ii*) how to select the best among the alternative options considering variability in both the contexts and stakeholder preferences. Substantial attention has been given to the representation and analysis of stakeholder goals (e.g., Rolland & Salinesi, 2005; Duran & Mussbacher, 2019), generally focusing on the requirements variability that originates from intentional variability and softgoal satisfaction. However, there is a need to further examine the relationships between goals and contexts, and

integrate context variability with requirements variability to establish early on, the influence of context to systems adaptability. It is also necessary to consider preference variability alongside requirements variability because preferences impact decisions about the alternative options of fulfilling goals. At the same time, context variability triggers preference variability, since stakeholders may have different priorities in relation to the fulfilment of their goals in different situations. Figure 4.1 illustrates the requirements-based outlook for adaptability. The contextual goals and contextual preferences, realised from contextual variability and preference variability, influence the intentional variabilities that fundamentally define requirements variability.

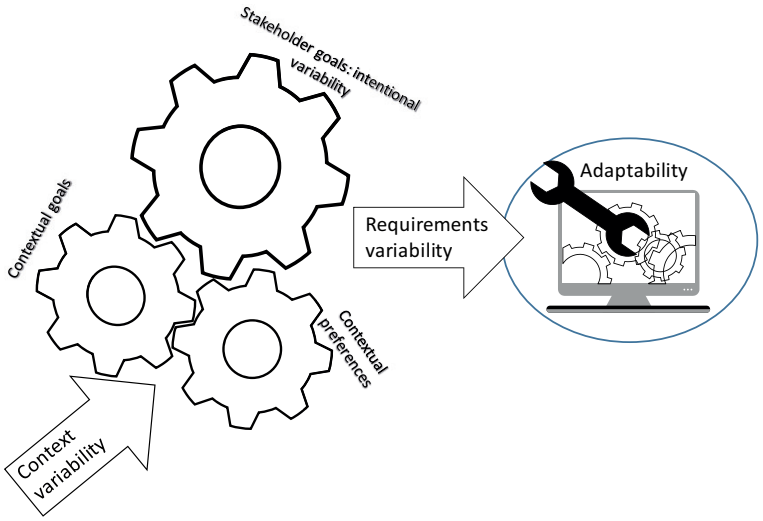



Figure 4.1: Adaptability from the perspective of requirements variability

The adaptability problem covered in this chapter is a typical choice problem in requirements variability and can be compared to those presented in the work of Henricksen, Indulska and Rakotonirainy (2006), Liaskos et al. (2011), and Ali et al. (2013). These works have recognised the impact of either context or preferences to requirements variability. This chapter, however, explores the *coexisting effect* of context and preferences to variability. The scope of preferences is expanded to not only the priority concerns with softgoals, but also with hardgoals. The chapter emphasises that context influences

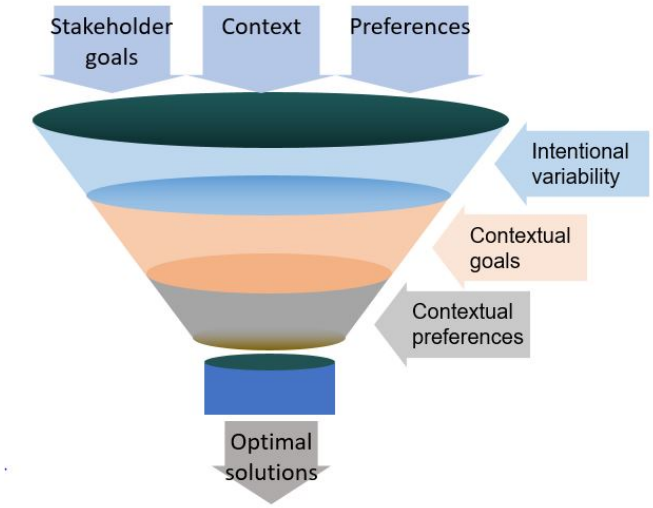



Figure 4.2: The stages of requirements variability analysis

not just the goals but also the preferences. Given a requirements problem defined by the stakeholders' goals, preferences, and relevant context information describing a usage situation, a *solution* that would satisfy such requirements problem is derived. The derivation process becomes a requirements variability analysis in three stages. The first stage utilises the intentional variabilities defined for the goals to generate the space of potential solutions. Considering context variability that may influence the goals, the second stage recognises contextual goals and generates solutions that are valid in certain contexts. The third stage applies the contextual preferences and identifies solutions that better satisfy priorities. A bird's-eye view of the adaptability problem is shown in Figure 4.2. The meta-model in Figure 4.3 illustrates relationships among the fundamental concepts underlying a contextual goal and contextual preference.

## 4.2 Running Scenario: Personal Medication Assistant

This section describes the requirements scenario used to facilitate subsequent discussions in the chapter, and gives an overview of the goal modelling approach adopted throughout this chapter.

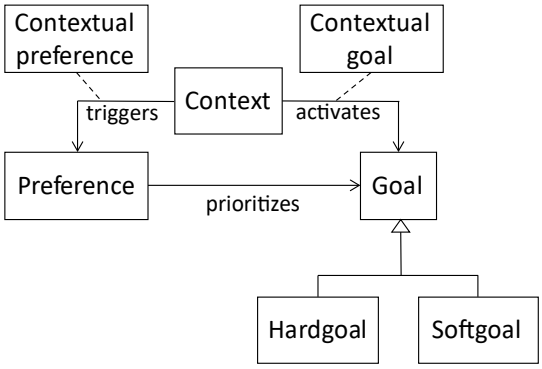



Figure 4.3: A conceptual diagram of a contextual goal and contextual preference

This section presents a requirements scenario for a personal medication assistant as described in the RAMCIP (robotic assistant for MCI patients) project (see Kostavelis et al., 2015). The personal medication assistant is an assistive medication system that facilitates patients' medication routine, both to ensure that a patient takes the needed medication, and to monitor proper medication or food supplement intake. This system is within the ambient assisted living domain which utilises smart objects and smart-home technologies to support elderly people or those with disabilities in their daily living activities. Some objectives and functions of the system include giving a reminder when it's time to take a medicine, giving a notice when an intake is missed or when a patient is leaving home and has to bring some pills, providing intake directions such as correct dosage, and giving warnings on attempts to take the wrong medicine. It also keeps a record of a patient's medicine intakes and monitors medication results and side-effects. To maintain peace of mind and a sense of assurance, intake information is conveyed to specified relatives. Similarly, the system informs external parties such as care-givers and physicians about problems such as consecutive intake misses or medication side-effects. In performing such functions, the system utilises appropriate services from objects within the smart home of a patient. For example, a reminder uses the display service of the kitchen TV screen when the patient is in the kitchen. The system also considers a patient's situation such as when one with a mild hearing impairment is in the backyard,

a reminder utilising an audio speaker service is adjusted to a higher volume to ensure it is noticed. If the patient is tired or unable to walk to the medicine cabinet, the medicine dispenser is fetched by a robot service. However, when an accompanying person (e.g., relative or caregiver) is present, the use of a robot should be restrained, as human assistance is preferred by the patient.

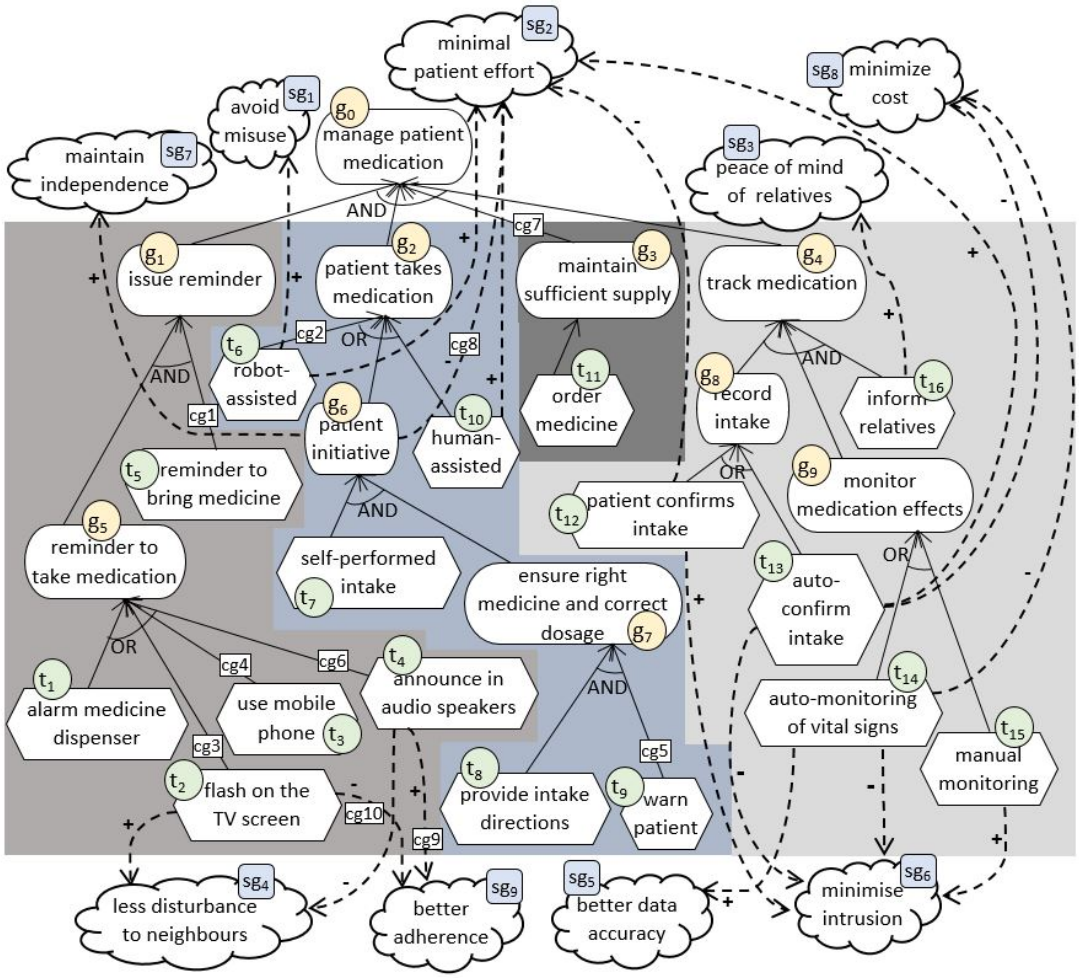



Figure 4.4: A partial goal model for the personal medication assistant.

Figure 4.4 shows a goal model that specifies a requirements fragment of the personal medication assistant. A subset of the i* goal modelling language originally proposed by E. S. Yu (1997) is adopted. Multiple variants of this language exist and efforts toward its standardisation have already been initiated (e.g., iStar 2.0 [Dalpiaz, Franch &

Horkoff, 2016]). The figure shows a typical goal model – an annotated AND/OR graph showing how higher-level goals are satisfied by lower-level ones (goal decomposition), and conversely, how lower-level goals contribute to the fulfilment of higher-level ones (goal abstraction) (see van Lamsweerde, 2009; Liaskos et al., 2011). The main elements of the goal model are *goals* that prescribe the intents (i.e., state of affairs or conditions) the system or actors of interest wish to satisfy. Goals are generally classified into hardgoals or softgoals. Although both can be further decomposed into some lower-level goals, variability concerns give more attention to hardgoal decomposition. A hardgoal (represented by the oval shape) is a mandatory goal that declaratively describes an intended system behaviour. A mandatory goal suggests that its fulfilment is a necessary step towards solving the main requirements problem defined by the root goal. A hardgoal has a clear-cut criterion for deciding whether or not it is satisfied; that is, satisfaction is through the leaf-level goals called *tasks* (depicted with hexagonal shapes), representing specific activities that can be performed by the system or actors, to operationalise and fulfil the goals. Relationships among goals and tasks are portrayed by *means-end links* ($\longrightarrow$) implying the means (i.e., subgoal or task) to fulfil a certain end (i.e., parent goal). Multiple means are associated with the *AND-/OR-decomposition links* ($\xrightarrow{AND}$/$\xrightarrow{OR}$). An AND-decomposed goal implies that the satisfaction/performance of all its children is necessary for its fulfilment, i.e., given a goal $g$ that is AND-decomposed into $g_1$,...,$g_n$, $g$ is satisfied iff $g_i$ is satisfied for all $i$. An OR-decomposed goal indicates that the satisfaction/performance of any among its children is sufficient for its fulfilment, i.e., when $g$ is OR-decomposed into $g_1$,...,$g_n$, $g$ is satisfied iff there exists an $i$ such that $g_i$ is satisfied. For example, the goal *track medication* ($g_4$) is satisfied by performing the tasks: *patient confirms intake* ($t_{12}$), *auto-monitoring of vital signs* ($t_{14}$), and *inform relatives* ($t_{16}$).

The softgoals (depicted with cloud shapes) indicate some system characteristics that are desired but not necessary to fulfil the root goal (Liaskos et al., 2011; Ernst

et al., 2010). Unlike hardgoals, most softgoals particularly quality goals, cannot be established in a clear-cut sense. For the personal medication assistant, it cannot be stated in a strict sense whether a certain system behaviour satisfies the goal *minimal patient effort* ($sg_2$). However, it may be stated that system behaviour contributes to the satisfaction of a softgoal to a certain degree. Relationships between hardgoals and softgoals are established by *contribution links*. The goal model utilises the *make* ($\xrightarrow{+}$) and *break* ($\xrightarrow{-}$) links, depicted with dashed arrow lines. The *make* link indicates that satisfying/performing the origin hardgoal/task satisfies the target softgoal. The *break* link indicates that satisfying/performing the origin hardgoal/task denies satisfying the target softgoal. In both cases, nothing is implied about the target if the origin is neither satisfied nor performed. Such interpretation of the contribution links is based on latter works (see Liaskos et al., 2011; Dalpiaz et al., 2016) that are built upon the original i* constructs. Some interpretations for the contribution links in the literature provide various levels of contribution to the satisfaction or denial of the target softgoals (e.g., a goal model used in business intelligence modelling defines four levels of contribution: strong and weak positive, and strong and weak negative (Horkoff et al., 2012)).

The diagrammatic formalism in the goal model can be an expressive equivalent to a propositional formula that defines the satisfaction of the root goal: $R \equiv T_g \wedge Q_g$, where each goal $g$ is associated with a propositional literal. $T_g$ denotes the AND/OR structure in terms of leaf level literals, i.e., tasks. $Q_g$ denotes the additional contribution links. Given two goals $g_1$ and $g_2$, the contribution links $g_1 \xrightarrow{+} g_2$ and $g_1 \xrightarrow{-} g_2$ are respectively written as implications $g_1 \rightarrow g_2$ and $g_1 \rightarrow \neg g_2$ in $Q_g$. In both $T_g$ and $Q_g$, a literal representing a non-leaf node is recursively replaced by its AND/OR decomposition until a clause composed of only leaf-level literals is reached. For example, for the goal $g_4$ in Figure 4.4: $T_{g4} \equiv (t_{12} \vee t_{13}) \wedge (t_{14} \vee t_{15}) \wedge t_{16}$, and $Q_{g4} \equiv (t_{12} \rightarrow \neg sg_2) \wedge (t_{12} \rightarrow sg_6) \wedge (t_{13} \rightarrow \neg sg_6) \wedge (t_{13} \rightarrow sg_2) \wedge (t_{13} \rightarrow \neg sg_8) \wedge (t_{14} \rightarrow sg_5) \wedge (t_{14} \rightarrow \neg sg_6) \wedge (t_{14} \rightarrow \neg sg_8) \wedge (t_{15} \rightarrow sg_6)$.

By considering the satisfaction of the AND/OR structure (by finding the truth assignments that satisfy the propositional formula $R$), the overall requirements problem posed by the root goal is satisfied by a set of tasks comprising a candidate solution. For instance, each set of tasks [$t_1$, $t_5$, $t_6$, $t_{11}$, $t_{12}$, $t_{14}$, $t_{16}$] and [$t_4$, $t_5$, $t_7$, $t_8$, $t_9$, $t_{11}$, $t_{13}$, $t_{15}$, $t_{16}$] is a candidate solution to the requirements problem defined by the root goal $g_0$.

Goal models have been used effectively to capture and analyse requirements variability (see Letier & Van Lamsweerde, 2004; Liaskos et al., 2006b; Ernst et al., 2010; Ali et al., 2013). On the one hand, goal models support a variability-intensive acquisition and decomposition of requirements. Even large numbers of alternative sets of low-level tasks, operations, and configurations that fulfil high-level stakeholder goals can be concisely represented. On the other hand, goal models facilitate reasoning about these alternative options. The alternatives that may arise at various stages of the requirements decomposition trigger the problem of choosing which of these options will be implemented or operationalised in a system.

Inspired by the work of Ali et al. (2013) and Stefanidis, Pitoura and Vassiliadis (2011), this chapter expands the modelling and reasoning capability of goal models to accommodate the variability in context. With the potential number of alternative means to meet every stakeholders' goal, a large space of solutions describing behavioural designs for a system-to-be is expected. Every OR-decomposition becomes the main *variability point* specifying the alternatives that can be part of a solution set. Operationalising such a large space of alternatives would be a daunting task, hence, it becomes practical to determine the suitable ones from which conforming design decisions can be derived. The suitability of a solution has three considerations. First, the applicability of each alternative hardgoal within a solution set can be context-dependent. Second, various stakeholders may have different prioritisations in relation to the alternative hardgoals or softgoals. Lastly, the context changes may motivate both goals and priorities to change accordingly. The requirements variability analysis needs to recognise the influence of

contexts on both goals and preferences. Therefore, it becomes necessary to represent contextual goals and contextual preferences in the requirements specifications.

## 4.3 Specifying Contextual Goals and Contextual Preferences

This section presents specifications for contexts, contextual goals, and contextual preferences. The contexts within which goals and preferences apply are modelled. Then for contextual goals, some goal model constructs are associated with contexts, defining *contextual variability points*. Defining contextual preferences mainly involves quantitative preference specification of goals (Liaskos et al., 2011) with contextual annotations.

### 4.3.1 Context specification

Context is the main element considered for software systems adaptability for which the framework in this chapter is intended. Context has been extensively defined and classified in the computing literature but there has been no consensus among researchers on its definition, classification, or scope. A survey of the various mechanisms for defining and classifying context was presented by Perera et al. (2014) and Alegre et al. (2016). Adopting the broad but often acknowledged definition by Abowd et al. (1999), *context* is any information used to characterise the situation of the system's operational environment that includes the system components and the users themselves.

The operational environment, where a system intends to satisfy goals, has *context entities* representing physical or conceptual objects in the environment, such as patients, weather, temperature, and devices. From these entities, properties of the entities, or relationships between the entities, *context elements* that might affect the goals are

defined. Each context element has a range of different values. A context element becomes a source of context variability, since the operational environment changes along with the change in the value of the context element.

**Definition 1** (Context dimension). Let $S$ be a system. The context dimension of $S$, $CD_S$, is a set of $n$ attributes, $CD_S = \{C_1, C_2, ..., C_n\}$, $n \geq 1$. Each attribute $C_i$, $i = 1, 2, ..., n$, is a context element.

Context dimension is a finite set of context elements relevant to the system (i.e., the context elements that influence the goals). Each context element can take an observable value to describe a particular contextual instance. Considering the personal medication assistant and assuming the relevant context elements: *patient-activity*, *patient-location*, *patient-illness*, *weather*, *body-condition*, and *accompanying-people*, the context dimension would be $CD_{PMA}$ = {*patient-activity*, *patient-location*, *patient-illness*, *weather*, *body-condition*, *accompanying-people*}. This context dimension can be further extended as additional relevant context elements are determined.

A context element $C$ has a domain of values $dom(C)$. To enable flexibility in defining context specifications, a hierarchical refinement of the domain values is utilised, eventually constructing a concept hierarchy (Stefanidis et al., 2011). This hierarchical refinement enables the context element values to be represented in different granularity. Figure 4.5 shows the hierarchical refinements of the domain values of some context elements identified for the personal medication assistant. A relationship among the domain values is represented by a directed edge from a child node to a parent node. A parent node abstracts its children nodes, and the children nodes are refinements of a parent node. Given a $dom(C)$, the most general or abstract node – the root node – is a value $All$ which abstracts all the values within $dom(C)$. Hence, $dom(C)$ is considered as an infinitary union $\bigcup_{i=1}^{\infty} A_i$, where $A_i$ is a set representing a parent node in $dom(C)$. For instance, $dom$(*patient-activity*) = *All* $\cup$ *busy* $\cup$ *taking-medicine*.

The hierarchical schema enables both context refinement to lower-level ones and

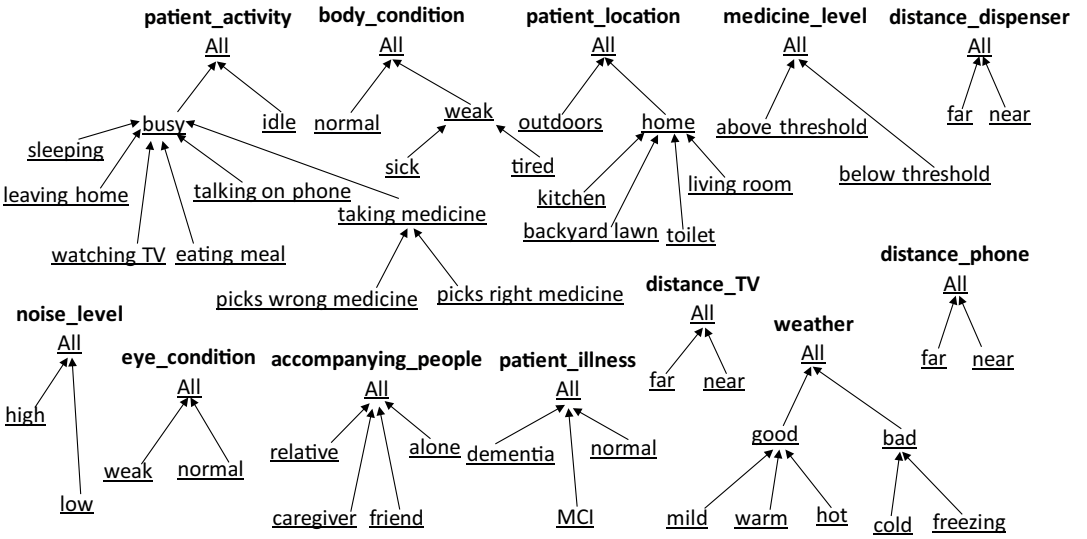



Figure 4.5: A hierarchical schema of context elements.

context abstraction to higher-level ones, producing appropriate context descriptions that can be more meaningful to a particular application, goal, or preference. For instance, a *robot-assisted* means of taking medication ($t_6$) requires the context *patient-location is at home* to hold. Although that same context may also apply to *flash a reminder on the TV screen* ($t_2$), a further detailed context is useful for $t_2$ to be effective, e.g., *patient-location is at the living room* where the TV can be easily noticed. Similar notions of modelling contexts with different levels of abstraction were utilised by Henricksen and Indulska (2004) and Lapouchnian and Mylopoulos (2009). Moreover, the context specification substantially conforms to a representational view of context (Dourish, 2004) where a context is a form of information that can be known, captured, and modelled. Hence, what counts as the (relevant) context for a system-to-be can be defined in advance, alongside requirements.

The context modelling involves contexts that characterise the physical tangible environment surrounding a system. Such contexts are vital in the domains that manifest interactions between the system and its environment, e.g., pervasive mobile systems, or smart/ambient systems. Capturing these contexts has now become adequately attainable

with the prevalent sensor and smart object technologies. Methods of capturing context, some of which are discussed by Perera et al. (2014), are however, beyond the scope of this chapter.

**Definition 2** (Context instance). Given a context dimension $CD_S$ = {$C_1$, $C_2$, ..., $C_n$}, a context instance is an $n$-tuple of the form ($c_1$, $c_2$, ..., $c_n$), $c_i \in dom(C_i)$, where $dom(C_i)$ is the domain of values for $C_i$, $1 \leq i \leq n$.

A context instance is the state of fact that describes the environment by assigning values to the elements of a context dimension. The one-to-one correspondence between the context dimension elements and context instance tuples forms a set of relations, i.e., ($C_1$, $c_1$), ($C_2$, $c_2$), ..., ($C_n$, $c_n$). Each relation ($C_i$, $c_i$) is an atomic proposition with the semantics '$C_i$ is $c_i$' or '$C_i$ = $c_i$', e.g., *patient-activity is busy*, or *body-condition is tired*. Hence, an $n$-tuple context instance implies a propositional conjunction of $n$ atomic propositions.

For the personal medication assistant (PMA), (*busy*, *outdoors*, *dementia*, *good*, *normal*, *alone*) and (*busy*, *kitchen*, *dementia*, *good*, *tired*, *caregiver*) are some of the context instances given that $CD_{PMA}$ = {*patient-activity*, *patient-location*, *patient-illness*, *weather*, *body-condition*, *accompanying-people*}. The space for context variability $W$ can now be defined, that is, the set of all possible context instances from the Cartesian product of the domains of the context elements, i.e., $W = dom(C_1) \times dom(C_2) \times ... \times dom(C_n)$.

**Definition 3** (Context descriptor). Let $CD_S$ = {$C_1$, $C_2$, ..., $C_n$} be a context dimension with $C_i$, $1 \leq i \leq n$, as its elements. A context descriptor $con(C_i)$ is an expression of the form $C_i \in \{c_1, ..., c_r\}$, where $c_q \in dom(C_i)$, $1 \leq q \leq r$.

A context descriptor specifies values of a context element. Suppose the context element *body-condition* has a context descriptor *body-condition* $\in$ {*tired*} or *body-condition* $\in$ {*tired*, *sick*}. The former specifies only one atomic proposition: *body-condition is tired*, while the latter specifies two: either *body-condition is tired* or

*body-condition is sick*.

A combined context descriptors of the elements in a context dimension is used to specify context instances. This combination, which is called $con$, is an expression $(con(C_1) \wedge ... \wedge con(C_n))$, where $con(C_i)$ is a context descriptor for a single element $C_i \in CD_S$, and there is at most one context descriptor for $C_i$. The use of the logical AND in $con$ implies that specifying a context instance requires the consideration of every context element within the context dimension. If the context descriptors of all context elements do not appear in $con$, the values of the missing elements are considered indifferent. Whenever $C_i$ is not found in $con$, a context descriptor $C_i \in \{All\}$ is implicitly included as part of $con$, where $All$ is a value that denotes any value from the domain $dom(C_i)$ of context element $C_i$.

Suppose given $CD_{PMA}$ = {*patient-activity*, *patient-location*, *patient-illness*, *weather*, *body-condition*, *accompanying-people*}, the (combined) context descriptor $con$ = (*patient-location* $\in$ {*indoor*} $\wedge$ *weather* $\in$ {*good*} $\wedge$ *body-condition* $\in$ {*tired*, *sick*}) specifies the context instance ($All$, $indoor$, $All$, $good$, $tired$, $All$) or ($All$, $indoor$, $All$, $good$, $sick$, $All$). The indifferent context elements are *patient-activity*, *patient-illness*, and *accompanying-people*. Moreover, a context descriptor can accommodate a negated value. For instance, *weather* $\in \{\neg freezing\}$ equivalently expresses either *weather* $\in$ {*cold*, *hot*, *mild*, *warm*} or *weather* $\in$ {*cold*, *good*}.

### 4.3.2 Contextual goal specification

A system's decision on what goals to reach and how to reach those goals is influenced by context, which makes up the properties of the operational environment that the system needs to observe. Likewise, the contributions to softgoals can also vary with context. A contextual goal specification annotates a goal or a contribution link with a context descriptor that explicitly defines the context instance when that goal or contribution link

is applicable.

**Definition 4** (Contextual goal). A contextual goal $cg$ is a pair (*Action*, *con*):

- *Action* is an expression of either a satisfaction of or a contribution to a goal model element. On the one hand, *Action* in the form *satisfy(g)* or *perform(t)* respectively denotes the satisfaction of goal $g$ or performance of task $t$. Both $g$ and $t$ are hardgoals. On the other hand, *Action* in the form *poscontrib(hg,sg)* or *negcontrib(hg,sg)* respectively denotes the positive or negative contribution of a hardgoal $hg$ to a softgoal $sg$.
- *con* is a (combined) context descriptor.

In Figure 4.4 on page 187, the small rectangular shapes represent the contextual goal specifications for those tagged goal model elements. The details of the contextual goal specifications are shown in Figure 4.6. The goal $g_1$: *issue reminder* is achieved through $g_5$: *reminder to take medication* and $t_5$: *reminder to bring medicine*. Despite $t_5$ being an AND-decomposition of $g_1$, a contextual goal specification $cg_1$ = (*perform*($t_5$), *patient-activity* $\in$ {*leaving-home*}) makes $t_5$ a contextual variability point. When *patient-activity* = *leaving-home* holds in the context instance, $t_5$ has to be performed. Otherwise, $t_5$ is not needed to achieve $g_1$. Meanwhile, $g_2$: *patient takes medication* is achieved through any means among $t_{10}$: *human assisted*, $t_6$: *robot assisted*, or $g_6$: *patient initiative*. Given this existing OR-decomposition variability, a contextual goal specification $cg_2$ = (*perform*($t_6$), *patient-location* $\in$ {*home*}) adds a contextual variability point at $t_6$. Hence, *patient-location* = *home* is necessary to hold in the context instance before adopting $t_6$ as an alternative. Otherwise $t_6$ cannot be an alternative for achieving $g_2$. Moreover, the contextual goal $cg_8$ = ($negcontrib(g_6, sg_2)$, *medicine-dispenser* $\in$ {*far*}) defines that $g_6$: *patient initiated medication* contributes negatively to $sg_2$: *minimal patient effort* only when *medicine-dispenser* = *far* holds in the context instance. This can be reasonable since the negative contribution of self-service medication takes place when the patient

has to exert some effort, moving towards the medicine location.

cg1 = (perform($t_5$), patient_activity ∈ {leaving_home})
cg2 = (perform($t_6$), patient_location ∈ {home})
cg3 = (perform($t_2$), distance_TV ∈ {near} ∧ patient_activity ∈ {¬sleeping})
cg4 = (perform($t_3$), distance_phone ∈ {near})
cg5 = (perform($t_9$), patient_activity ∈ {picks_wrong_medicine})
cg6 = (perform($t_4$), noise_level ∈ {low})
cg7 = (satisfy($g_3$), medicine_level ∈ {below_threshold})
cg8 = (negcontrib($g_6$, $sg_2$), distance_dispenser ∈ {far})
cg9 = (poscontrib($t_4$, $sg_9$), patient_location ∈ {backyard_lawn})
cg10 = (negcontrib($t_2$, $sg_9$), eye_condition ∈ {weak})

Figure 4.6: A goal catalogue

**Definition 5** (Goal catalogue). A goal catalogue $G$ is a set of contextual goals for a requirements goal model.

Figure 4.6 shows a goal catalogue that defines the contextual variability points in the personal medication assistant goal model. The contextual goals $cg_1$, $cg_5$, and $cg_7$, which are associated with some AND-decomposition of goals, indicate that a subgoal/subtask in an AND-decomposition might be needed only in a particular context. For example in $cg_7$, the goal $g_3$, which triggers the task of ordering medicine ($t_{11}$), is needed only when the medicine level is below a specified threshold. Even if $g_3$ is among the AND-decomposition of $g_0$, $g_3$ is not always mandatory to achieve $g_0$. Likewise, $cg_5$ defines that warning a patient ($t_9$) becomes necessary only when she picks the wrong medicine. Meanwhile, the contextual goals $cg_2$, $cg_3$, $cg_4$, and $cg_6$, which are associated with OR-decomposition, suggest that adoptability of a subgoal/subtask might require a valid context. For instance, $cg_4$ defines that a mobile phone can be used for issuing a reminder ($t_3$) when the phone is near the patient. Likewise for $cg_3$, flashing a reminder on the TV screen ($t_2$) can be adopted when the patient is near the TV and not sleeping. $cg_3$ specifies a goal that depends on the values of two context elements of which one contains a negated value, i.e., *distance-TV* ∈ {*near*} ∧ *patient-activity*

$\in \{\neg$*sleeping*$\}$. The contextual goals $cg_8$, $cg_9$, and $cg_{10}$ suggest that some positive or negative contributions of hardgoals to softgoals only apply in certain contexts. For instance, $cg_9$, a reminder announced through audio speakers ($t_4$) contributes positively to better adherence ($sg_9$) when the patient is in the backyard lawn, i.e., a medicine dispenser alarm, a reminder flashed on screen, or a text message may not be noticed on time, as the medium for these options are usually located inside the home. Likewise, in $cg_{10}$, a reminder flashed on a screen ($t_2$) negatively contributes to better adherence ($sg_{10}$) when the patient has weak eyesight. Such a deficiency makes the visual reminder hard to notice, hence, it may lead to missed medication intake.

Referring to the hierarchical schema in Figure. 4.5 on page 193, the context descriptor *patient-activity* $\in \{\neg$*sleeping*$\}$ has a similar connotation to *patient-activity* $\in \{$*leaving-home*, *watching-TV*, *eating-meal*, *talking-on-phone*, *taking-medicine*, *idle*$\}$. The use of a negated value simplifies a context descriptor that would have to enumerate multiple values. Furthermore, a context descriptor defines a context instance characterised by all elements in the context dimension. For example, the *patient-location* $\in \{$*home*$\}$ in $cg_2$ defines a context instance (*home*, *All*, *All*, ..., *All*), where the only relevant environment property is a patient's location.

### 4.3.3 Contextual preference specification

Preferences are used to characterise the alternative solutions to a requirements problem. Such characterisation facilitates the decision of selecting better solution sets from among the candidate variants. Preferences can be specified to the hardgoal alternatives or softgoals. A preference specification has two general approaches – a qualitative or quantitative preference (C. M. Nguyen et al., 2018). The qualitative approach directly specifies preferences between objects of concern, typically by using binary preference relations. The quantitative approach indirectly specifies preferences, using

scoring functions that associate a numeric score to the objects of concern. A contextual preference specification, which explicitly indicates the context instance of a preference, can apply to both approaches. This chapter uses a quantitative contextual preference model that annotates preferences with contexts and scoring components.

**Definition 6** (Contextual preference). A contextual preference $p$ is a triple (*Action*, *con*, *score*):

- *Action* is a satisfaction expression of a desired property in the solutions implied by the goal model. *Action* is in the form *satisfy(g)* or *perform(t)* respectively denoting the preference to the satisfaction of goal $g$ or performance of task $t$. A goal $g$ can be a hardgoal or softgoal.
- *con* is a (combined) context descriptor.
- *score* is a numerical value within a certain range, e.g., [0, $n$].

The *Action* and *con* parameters of a contextual preference are apparently similar to those of a contextual goal, that is, in terms of the individual definition of each parameter. However, the semantics of the combined parameters differ in each specification. In a contextual goal, *con* activates or deactivates an *Action*, i.e., an *Action* may (not) become a part of a valid solution. However, in a contextual preference, the *Action* is already part of a valid solution. Moreover, an *Action* in a contextual goal is associated with either AND- or OR-decomposition of a hardgoal, while in contextual preferences, an *Action* is associated with either a softgoal or an OR-decomposition of a hardgoal.

The added scoring component *score* indicates the priority level assigned to an *Action* considering the context instance specified by *con*. The specification employs a preference model that follows the quantitative preference framework of Agrawal and Wimmers (2000), which was also applied by Henricksen et al. (2006) and Stefanidis et al. (2011). However, with respect to generating those scores, the framework presented in this chapter focuses on modelling and reasoning preferences and does not bind to any

specific score elicitation approach. Various approaches in the requirements literature for eliciting quantitative priority values can be adopted. This chapter uses a range of score values that implies five levels of preference strength, i.e. 0 - 4. The score values 4, 3, 2, 1, and 0 respectively indicate *extreme preference*, *strong preference*, *moderate preference*, *slight preference*, and *indifference* to satisfy an *Action* in the context *con*. Such a five-level preference was introduced in the Analytic Hierarchy Process (AHP) (Karlsson & Ryan, 1997), and is one of the prominent approaches for eliciting priority scores. Although the scoring component can also represent the levels of negative preferences (Bistarelli, Pini, Rossi & Venable, 2005), only positive preferences are used in the examples for simplicity.

The contextual preference specification allows the expression of priorities through priority rankings of the desired properties of expected solutions, as well as considering the circumstances that hold for such priorities. Considering the goal model in Figure 4.4 on page 187, the contextual preference (*perform*(*robot-assisted* $t_6$), *patient-illness* $\in$ {*dementia*}, 4) expresses when the patient has *dementia*, performing the task *robot-assisted* for taking a medication is preferred with the highest priority score of 4. In contrast, (*perform*(*robot-assisted* $t_6$), *patient-illness* $\in$ {*normal*}, 0) indicates no level of interest in performing a robot-assisted medication when the patient has no illness. Since such contextual preference showing indifference is of no effect, its explicit specification can be omitted.

It may also be necessary to explicitly define *non-contextual preferences*, the preferences that hold regardless of the context elements' values. Such preferences are considered *default preferences*. For instance, (*satisfy*($sg_1$), *patient-illness* $\in$ *All*, 3), is a non-contextual preference, i.e., *patient-illness* $\in$ *All* defines the context instance (*All*, *All*, ..., *All*). This is semantically similar to a contextual preference with an empty context descriptor.

**Definition 7** (Preference catalogue). A preference catalogue $P$ is a set of contextual

or non-contextual preferences that applies to a particular requirements goal model.

A preference catalogue, through its set of contextual preferences, defines the priorities imposed over the hardgoal alternatives and softgoals in certain situations. A catalogue may contain the priorities of an individual stakeholder or a combination of priorities from various stakeholders. A higher score for a contextual preference indicates a higher level of importance of satisfying/performing the associated goal/task. This stands in comparison to the respective alternatives that either have contextual preferences with lower scores or those not mentioned at all. Figure 4.7 shows a preference catalogue for the personal medication assistant. The contextual preferences can be clustered based on the OR-decomposition they associate with (e.g., $A$, $B$, and $C$). Cluster $D$ contains the preferences over softgoals. The contextual preferences $p2$, $p3$, $p4$, $p5$, $p6$, and $p1$ are respectively associated with the three alternatives for satisfying goals $g_2$: $t_6$, $g_6$, and $t_{10}$. In $p5$, it becomes explicit that $g_6$ is preferred over $t_6$ and $t_{10}$ when the patient is near the medicine dispenser, while in $p3$, $t_6$ is most preferred when the following values in the context instance hold: {*alone*, *sleeping*, *tired*} or {*alone*, *sleeping*, *busy*}. The contextual preference $p1$ combines the three preferences for *perform*($t_6$), *perform*($t_{13}$), and *perform*($t_{14}$), and each one is similarly preferred with a score of 4 when the patient's illness is dementia. The symbol • is used as a separator among the joined *Action*s in this notation.

A similar notion is applied to the contextual preferences for softgoals, since softgoals may not always have equal importance. Different stakeholders in different contexts are interested in satisfying different subsets of softgoals, to which they also give different levels of importance (Liaskos et al., 2011). Referring to Figure 4.7, $p8$ defines an extreme preference for *minimal patient effort* $sg_2$ when the patient is sick. Likewise, $sg_6$ and $sg_7$ are extremely preferred in the situations defined in $p9$ and $p10$ respectively. The specifications $p11$, $p12$, and $p13$ define the non-contextual preferences for the softgoals, where such preferences hold regardless of the context. A non-contextual preference

**A**
p1 = (perform($t_6$) • perform($t_{13}$) • perform($t_{14}$), patient_illness ∈ {dementia}, 4)

**B**
p2 = (perform($t_6$), body_condition ∈ {tired, sick}, 2)
p3 = (perform($t_6$), accompanying_people ∈ {alone} ∧ patient_activity ∈ {sleeping} ∧ body_condition ∈ {tired, sick}, 4)
p4 = (perform($t_6$), accompanying_people ∈ {alone} ∧ patient_activity ∈ {busy}, 2)
p5 = (satisfy($g_6$), distance_dispenser ∈ {near}, 4)
p6 = (perform($t_{10}$), accompanying_people ∈ {caregiver, relative}, 3)

**C**
p7 = (perform($t_{15}$), patient_illness ∈ {normal}, 2)

**D**
p8 = (satisfy($sg_2$), body_condition ∈ {sick}, 4)
p9 = (satisfy($sg_6$), patient_illness ∈ {normal} ∧ body_condition ∈ {normal}, 4)
p10 = (satisfy($sg_7$), patient_illness ∈ {normal}, 4)
p11 = (satisfy($sg_1$), patient_illness ∈ {All}, 3)
p12 = (satisfy($sg_6$) • satisfy($sg_7$) • satisfy($sg_8$), patient_illness ∈ {All}, 2)
p13 = (satisfy($sg_2$) • satisfy($sg_3$) • satisfy($sg_4$) • satisfy($sg_5$), patient_illness ∈ {All}, 1)

Figure 4.7: A preference catalogue

becomes a default preference that may apply in any context instance.

The use of combined context descriptors allows the expression of preferences that depend on values of more than one context element. For instance, $p4$ specifies that a robot assisted intake ($t6$) is (*moderately*) preferred when *accompanying-people* = *alone* and *patient-activity* = *busy*. Moreover, multiple satisfaction expressions that are given similar preference scores in a particular situation can be combined in a single contextual preference notation (e.g., $p1$).

## 4.4 Requirements Variability in Varying Contexts and Preferences

A system implementation for the goal model in Figure 4.4 on page 187 may support all or a subset of the requirements variants. Selection among variants can be a design decision prior to system implementation, or a runtime decision (Ali et al., 2010). For instance, to minimise the overall development cost, the designer may decide to reduce the number of variants to be implemented. Otherwise, if the designer wants flexibility and high variability, more variants need to be implemented. Whether the decision is manually done by the designer prior to implementation or automatically performed by the system at runtime, a mechanism for requirements variability analysis is necessary to derive variants applicable in certain contexts and those that satisfy stakeholder preferences.

This section demonstrates the use of contextual goals and contextual preferences in the requirements variability analysis. Requirements variability is a derivation of solutions in which the inputs are: *i*) the relevant context instance describing a system usage situation, *ii*) the set of defined goals, *iii*) the contextual goals, and *iv*) the contextual preferences over goals. The output is a subset of the inputted goals that can be satisfied

or performed in the given context instance. Subsequently, each requirements variant emanating from this output describes a candidate solution to the main requirements problem.

### 4.4.1 Contextual situation

Contextual situation, i.e., *situation*, is the environment context for a system.

**Definition 8** (Contextual situation). Given a system $S$ and its context dimension $CD_S$, a contextual situation *situ(S)* is a context instance specified from $CD_S$ describing the implementation or runtime environment for $S$.

It is worth mentioning the difference between *con* and *situ*. Although both refer to context instances, *con* is particular to a contextual goal or preference, while *situ* is the context instance of the system. It is assumed that the context dimension $CD_{PMA}$ = {*patient-activity*, *body-condition*, *patient-location*, *medicine-level*, *distance-dispenser*, *noise-level*, *eye-condition*, *accompanying-people*, *patient-illness*, *distance-TV*, *weather*, *distance-phone*} for the system, for which requirements are described in Figure 4.4 on page 187. From this context dimension, context instances can be specified defining certain contextual situations such as: $a$ = (*sleeping*, *normal*, *home*, *above-threshold*, *far*, *high*, *weak*, *alone*, *normal*, *far*, *good*, *far*) or $b$ = (*idle*, *tired*, *living-room*, *below-threshold*, *far*, *low*, *weak*, *caregiver*, *dementia*, *near*, *good*, *far*) (see Figure 4.8). A situation refers to either case: a potential environment for a system-to-be, or the current operational environment. In the former case, the context instance is explicitly expressed to explore the suitability of various implementation designs. For example, the designer posits contextual situations that may correspond to some exploratory design possibilities such as the solutions to implement for a patient with *MCI* and *weak eyesight*, or for a patient who stays at *home* and is often *alone*. In the latter case, the context instance corresponds to the current context of a running system. The latter case assumes a

requirements model at runtime (Szvetits & Zdun, 2016; Bencomo, Götz & Song, 2019) that facilitates runtime adaptation, such as the goal-based service composition approach in the work of Cavallaro, Sawyer, Sykes, Bencomo and Issarny (2012). The advances in sensor and context-aware technologies enable the capture of runtime context instances, and various methods for capturing this (implicit) context are discussed extensively in the literature.

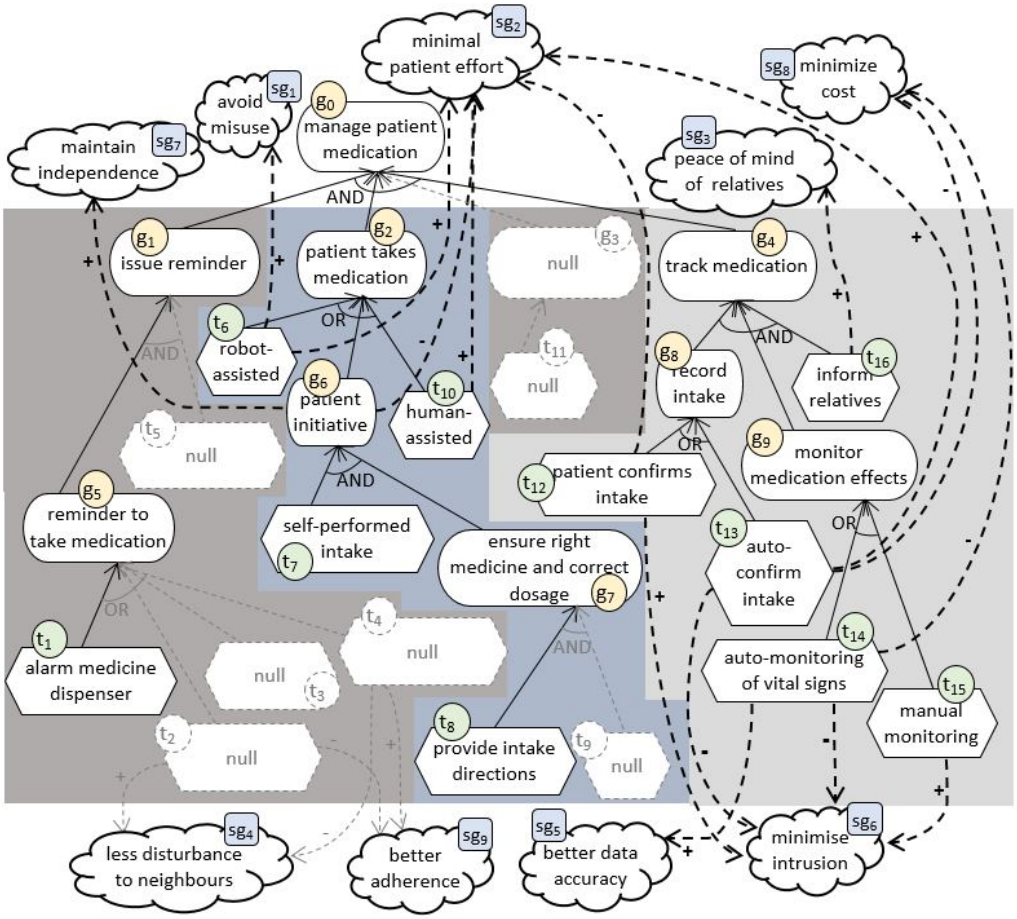
| *Contextual situation* | $CD_{PMA}$ | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | patient activity | body condition | patient location | medicine level | distance dispenser | noise level | eye condition | accompanying people | patient illness | distance TV | weather | distance phone |
| situation ***a*** | sleeping | normal | home | above threshold | far | high | weak | alone | normal | far | good | far |
| situation ***b*** | idle | tired | living room | below threshold | far | low | weak | caregiver | dementia | near | good | far |

Figure 4.8: Contextual situations for the personal medication assistant

A contextual situation may describe a single context state, i.e., each context element takes a detailed leaf-level value from its domain. An example is described in situation $b$ in Figure 4.8. However, there are cases when it is possible to specify a situation using general values only, e.g., when context values provided by sensors have limited accuracy. In such cases, a context element may take parent-node values from the hierarchical refinement. For example, in situation $a$ in Figure 4.8, the context element *patient-location* takes an abstract value *home* which is a less detailed value in the context hierarchy.

### 4.4.2 Requirements variability via contextual goals

Considering the contextual goals defining contextual variability points in a goal model, the valid requirements for a given situation can be derived. Hence, different goal model variants are derived in different situations.




**Contextual situation**: *patient_activity ∈ {sleeping} ∧ body_condition ∈ {normal} ∧ patient_location ∈ {home} ∧ medicine_level ∈ {above threshold} ∧ distance_dispenser ∈ {far} ∧ noise_level ∈ {high} ∧ eye_condition ∈ {weak} ∧ accompanying_people ∈ {alone} ∧ patient_illness ∈ {normal} ∧ distance_TV ∈ {far} ∧ weather ∈ {good} ∧ distance_phone ∈ {far}*

Figure 4.9: A goal model showing the requirements variants that are valid in a given situation

**Definition 9** (Cover relation between context instances). Given two context instances $cs_1$ = ($c_1^1$, ..., $c_n^1$) and $cs_2$ = ($c_1^2$, ..., $c_n^2$), $cs_2$ covers $cs_1$ if $\forall i$, $1 \leq i \leq n$, $c_i^2 = c_i^1$ or $c_i^2 = anc(c_i^1)$, where $anc(c)$ refers to an ancestor of $c$.

Given a goal catalogue $G$, the contextual goal specifications from $G$ that are relevant to a contextual situation can be identified. A contextual goal $cg$ is relevant when any context instance of $cg$ *covers* the contextual situation. A context instance of $cg$ must be the same or more general than that of the contextual situation, otherwise, $cg$ is non-relevant.

However, to derive requirements that are valid in a certain situation, the focus should not just be on the relevant contextual goals but also on the non-relevant ones. The following summarises the derivation process into four steps:

- **Step 1**. Identification of the non-relevant ones from those contextual goals that have *Action* of the form *satisfy()* or *perform()*.
- **Step 2**. A goal model element associated with the non-relevant contextual goal becomes void (deactivated). The void element $e$ is considered non-existent in the goal model and can be discarded in that given situation.
- **Step 3**. Any subgoal, decomposition, or contribution link associated with $e$ becomes void as well.
- **Step 4**. Further identification of the non-relevant contextual goals that have *Action* of the form *poscontrib()* or *negcontrib()*. A contribution link associated with those contextual goals is considered void.

**Example 1**. Figure 4.9 shows a variation of the goal model in Figure 4.4 on page 187 when situation $a$ in Figure 4.8 holds. The densely coloured dotted-objects represent the goal model elements that are not applicable in situation $a$. Considering the goal catalogue in Figure 4.6 on page 197, the aforementioned steps are followed to derive the valid requirements as shown by the goal model. (*Step 1*) Among contextual goals

that have *Action* of the form *satisfy()* or *perform()*, only $cg_2$ is relevant. A robot-assisted intake ($t_6$) is a valid option for taking medication since situation $a$ specifies *patient-location = home*. (*Step 2*) The goals/tasks $t_5$ (reminder to bring medicine), $t_2$ (flash on the TV screen), $t_3$ (use mobile phone), $t_9$ (warn patient), $t_4$ (announce in audio speakers), and $g_3$ (maintain sufficient supply), which respectively associate with the non-relevant contextual goals $cg_1$, $cg_3$, $cg_4$, $cg_5$, $cg_6$, and $cg_7$, are discarded. The context instance needed to adopt or achieve each goal/task does not cover situation $a$. For example, the use of a mobile phone to issue a reminder ($t_3$) is not a valid option since situation $a$ specifies that the patient is *far* from the mobile phone. (*Step 3*) The task $t_{11}$ (order medicine) becomes void with its parent goal $g_3$. Likewise, the contribution links from $t_2$ and $t_4$ are discarded. (*Step 4*) Furthermore, among the contextual goals involving contribution links, only $cg_8$ is relevant, and $cg_9$ and $cg_{10}$ are non-relevant. Although the context instance of $cg_{10}$ covers that of situation $a$, which would make $cg_{10}$ relevant, the associated contribution link becomes void with $t_2$.

**Example 2**. If, for the goal model in Figure 4.9, the situation changes to $b$ as described in Figure 4.8, the contextual goals $cg_2$, $cg_3$, $cg_6$, and $cg_7$, become relevant. Hence, $t_6$ (robot-assisted), $t_2$ (flash on the TV screen), $t_4$ (announce in audio speakers), and $g_3$ (maintain sufficient supply) are (re)activated, including the contribution links, subgoals, or decomposition associated with these goals/tasks. As to contextual goals associated with contribution links, $cg_8$ and $cg_{10}$ become relevant, while $cg_9$ remains non-relevant. Therefore, only $t_5$ (reminder to bring medicine), $t_3$ (use mobile phone), $t_9$ (warn patient), and the contribution link from $t_4$ to $sg_9$ (better adherence) are discarded.

The activated elements of a goal model may vary as the situation changes. Different requirements variants are valid in different situations. Consequently, the candidate solutions to achieve the main requirements problem may also vary for each situation. For instance, the set of tasks [$t_1$, $t_7$, $t_8$, $t_{12}$, $t_{14}$, $t_{16}$] is a candidate solution for the requirements problem in Figure 4.9. However, when the situation changes to the one

described in Example 2, that solution is no longer valid because the solution does not satisfy the AND/OR structure of the goal model variation. Task $t_{11}$ has to be included in that set of tasks to be a candidate solution for the resulting goal model variation, i.e., [$t_1$, $t_7$, $t_8$, $t_{11}$, $t_{12}$, $t_{14}$, $t_{16}$].

### 4.4.3 Requirements variability via contextual preferences

As demonstrated in Section 4.4.2, there can be multiple requirements variants in a given contextual situation. Each variant defines an alternative solution to the main requirements problem. The contextual preferences can be utilised in the decision on which solution to adopt of the several alternatives. The degree to which a candidate solution satisfies the contextual preferences, is measured.

Given a preference catalogue $P$, the contextual preference specifications from $P$ relevant to a given situation are identified. A contextual preference $p$ is relevant when any context instance of $p$ covers the contextual situation.

**Example 3**. Given the goal model and contextual situation in Figure 4.9 on page 206, the relevant contextual preferences are identified from the preference catalogue in Figure 4.7 on page 202. Preferences $p4$ and $p7$ express priorities for the hardgoals $t_6$ and $t_{15}$ respectively. Preferences $p9$, $p10$, $p11$, $p12$, and $p13$ express priorities for the softgoals $sg_1$, $sg_2$, ..., $sg_8$, where $p11$, $p12$, and $p13$ are the non-contextual preferences.

The degree by which a solution satisfies the relevant preferences is calculated. The *softgoal preference score* and *hardgoal preference score* of each solution are calculated. Afterwards, the scores are combined to derive an overall *preference satisfaction degree* for each solution.

### Softgoal preferences

The softgoal preference score is calculated by considering the contribution links from a solution to the preferred softgoals. A quantitative estimation approach is used for assessing the positive or negative contribution of the candidate solutions to the softgoals. This approach was introduced in the NFR framework (Mylopoulos et al., 1992), and further reused by van Lamsweerde (2009) and Ali et al. (2010). The approach considers every contribution link as evidence of the positive or negative satisfaction of a preferred softgoal.

**Definition 10** (Softgoal preference score). A softgoal preference score *sps(sol)* is the score derived for a solution *sol* with respect to satisfying the preferred softgoals, *sps(sol)* = *contrib*(*sol*, $sg_1$) + ... + *contrib*(*sol*, $sg_n$), where:

- *contrib*(*sol*, $sg_i$) = *percentPos*(*sol*, $sg_i$) * *score*($sg_i$) - *percentNeg*(*sol*, $sg_i$) * *score*($sg_i$), is the contribution score of the solution *sol* to $sg_i$,
- each $sg_i$, $1 \leq i \leq n$, is a softgoal associated to a non-contextual/contextual preference $p$, and for any two softgoals $sg_i$, $sg_k$, $i \neq k$.

The function *percentPos*(*sol*, $sg_i$) refers to the percentage of the positive contributions ($\xrightarrow{+}$) with respect to the total number of contributions from *sol* to $sg_i$, while *percentNeg*(*sol*, $sg_i$) refers to the percentage of the negative contributions ($\xrightarrow{-}$) with respect to the total number of contributions from *sol* to $sg_i$. These functions are used to normalise the different number of contribution links upon softgoals. The function *score*($sg_i$) is the priority degree defined for $sg_i$ in a contextual preference. Where $sg_i$ appears in multiple relevant contextual preferences, the highest defined score for $sg_i$ from among the relevant preferences is chosen for *score*($sg_i$).

**Example 4**. Table 4.1 shows the derived softgoal preference score of each candidate solution of the requirements problem illustrated in Figure 4.9 on page 206. The headers of the middle columns show the preferred softgoals and their individual priority scores

Table 4.1: The softgoal preference scores

| Candidate solution ($sol$) | Preferred softgoals | | | | | | | *sps(sol)* |
|---|---|---|---|---|---|---|---|---|
| | $sg_1$, 3 | $sg_2$, 1 | $sg_3$, 1 | $sg_5$, 1 | $sg_6$, 4 | $sg_7$, 4 | $sg_8$, 2 | |
| a: [$t_1$, $t_7$, $t_8$, $t_{12}$, $t_{14}$, $t_{16}$] | | -1.0 | 1.0 | 1.0 | 0.0 | 4.0 | -2.0 | 3.0 |
| b: [$t_1$, $t_7$, $t_8$, $t_{12}$, $t_{15}$, $t_{16}$] | | -1.0 | 1.0 | | 4.0 | 4.0 | | 8.0 |
| c: [$t_1$, $t_7$, $t_8$, $t_{13}$, $t_{14}$, $t_{16}$] | | 0.0 | 1.0 | 1.0 | -4.0 | 4.0 | -2.0 | 0.0 |
| d: [$t_1$, $t_7$, $t_8$, $t_{13}$, $t_{15}$, $t_{16}$] | | 0.0 | 1.0 | | 0.0 | 4.0 | -2.0 | 3.0 |
| e: [$t_1$, $t_6$, $t_{12}$, $t_{14}$, $t_{16}$] | 3.0 | 0.0 | 1.0 | 1.0 | 0.0 | | -2.0 | 3.0 |
| f: [$t_1$, $t_6$, $t_{12}$, $t_{15}$, $t_{16}$] | 3.0 | 0.0 | 1.0 | | 4.0 | | | 8.0 |
| g: [$t_1$, $t_6$, $t_{13}$, $t_{14}$, $t_{16}$] | 3.0 | 1.0 | 1.0 | 1.0 | -4.0 | | -2.0 | 0.0 |
| h: [$t_1$, $t_6$, $t_{13}$, $t_{15}$, $t_{16}$] | 3.0 | 1.0 | 1.0 | | 0.0 | | -2.0 | 3.0 |
| i: [$t_1$, $t_{10}$, $t_{12}$, $t_{14}$, $t_{16}$] | | 0.0 | 1.0 | 1.0 | 0.0 | | -2.0 | 0.0 |
| j: [$t_1$, $t_{10}$, $t_{12}$, $t_{15}$, $t_{16}$] | | 0.0 | 1.0 | | 4.0 | | | 5.0 |
| k: [$t_1$, $t_{10}$, $t_{13}$, $t_{14}$, $t_{16}$] | | 1.0 | 1.0 | 1.0 | -4.0 | | -2.0 | -3.0 |
| l: [$t_1$, $t_{10}$, $t_{13}$, $t_{15}$, $t_{16}$] | | 1.0 | 1.0 | | 0.0 | | -2.0 | 0.0 |

as specified in the preference catalogue in Figure 4.7 on page 202. The softgoals $sg_1$, $sg_2$, ..., $sg_8$, are found in the relevant contextual preferences (see Example 3). Both $sg_6$ and $sg_7$ appear in two contextual preferences: $sg_6$ is in $p9$ and $p12$, while $sg_7$ is in $p10$ and $p12$. In this case, the higher priority score between those respective preferences is chosen for each softgoal. That is, the priority score '4' in either $p9$ or $p10$ overrides the lower score '2' specified by the default preference $p12$. A table cell associated with a candidate solution *sol* and a softgoal $sg$ captures the *contrib*(*sol*, $sg$), which is the estimated preference score of *sol* with respect to $sg$. An empty cell denotes no contribution links from *sol* to $sg$. The last column shows the total preference score of each candidate solution, i.e., *sps(sol)*. For instance, *sps(a)* = 3.0 is the sum of the preference scores of solution $a$: [$t_1$, $t_7$, $t_8$, $t_{12}$, $t_{14}$, $t_{16}$] with respect to each preferred softgoal.

A solution's positive (resp. negative) contribution to a softgoal is significant only if there is no negative (resp. positive) contribution that negates it coming from the same solution. For instance, the solution $a$: [$t_1$, $t_7$, $t_8$, $t_{12}$, $t_{14}$, $t_{16}$] has one positive contribution to $sg_6$ (from $t_{12}$), but this is negated by the negative contribution from $t_{14}$. Hence, the computed score of solution $a$ for $sg_6$ is 0.

### Hardgoal preferences

The relevant contextual preferences over some hardgoals in the goal model are considered to derive the hardgoal preference score of each candidate solution.

**Definition 11** (Hardgoal preference score). A hardgoal preference score *hps(sol)* is the score derived for a solution *sol* with respect to the contextual preferences over the goal alternatives, *hps(sol)* = *pref*(*sol*, $hg_1$) + ... + *pref*(*sol*, $hg_n$), where:

- *pref*(*sol*, $hg_i$) = *score*($hg_i$), is the preference score of the solution *sol* to the alternative hardgoal $hg_i$,
- each $hg_i$, $1 \leq i \leq n$, is a hardgoal associated to a contextual preference $p$, and for any two hardgoals $hg_i$, $hg_k$, $i \neq k$.

The function *score*($hg_i$) is the priority degree defined for $hg_i$ in a contextual preference. Where $hg_i$ appears in multiple relevant contextual preferences, the highest defined score for $hg_i$ from among the relevant preferences is chosen for *score*($hg_i$). For instance, contextual preference $p3$ and $p4$ in the preference catalogue in Figure 4.7 on page 202 both define preferences for performing $t_6$ (i.e., the priority scores '4' and '2' are defined respectively). When both $p3$ and $p4$ become relevant in a given situation, *score*($t_6$) = 4, which is the higher score of the two.

**Example 5**. Using the same goal model in Figure 4.9 and contextual preferences in Figure 4.7, Table 4.2 shows the derived hardgoal preference score of each candidate solution. The middle column lists the *pref(sol, hg)* – the preference score of each preferred hardgoal $hg$ that is satisfied in a particular solution $sol$. The last column totals the preference scores deriving the respective *hps(sol)*. For instance, *hps(f)* = 4.0 adds the preference scores associated to the preferred hardgoals, $t_6$: 2, $t_{15}$: 2, that are satisfied in the solution $f$: [$t_1$, $t_6$, $t_{12}$, $t_{15}$, $t_{16}$].

Table 4.2: The hardgoal preference scores

| Candidate solution (*sol*) | Hardgoal preferences | *hps(sol)* |
|---|---|---|
| a: [$t_1$, $t_7$, $t_8$, $t_{12}$, $t_{14}$, $t_{16}$] | | 0.0 |
| b: [$t_1$, $t_7$, $t_8$, $t_{12}$, $t_{15}$, $t_{16}$] | $t_{15}$: 2 | 2.0 |
| c: [$t_1$, $t_7$, $t_8$, $t_{13}$, $t_{14}$, $t_{16}$] | | 0.0 |
| d: [$t_1$, $t_7$, $t_8$, $t_{13}$, $t_{15}$, $t_{16}$] | $t_{15}$: 2 | 2.0 |
| e: [$t_1$, $t_6$, $t_{12}$, $t_{14}$, $t_{16}$] | $t_6$: 2 | 2.0 |
| f: [$t_1$, $t_6$, $t_{12}$, $t_{15}$, $t_{16}$] | $t_6$: 2, $t_{15}$: 2 | 4.0 |
| g: [$t_1$, $t_6$, $t_{13}$, $t_{14}$, $t_{16}$] | $t_6$: 2 | 2.0 |
| h: [$t_1$, $t_6$, $t_{13}$, $t_{15}$, $t_{16}$] | $t_6$: 2, $t_{15}$: 2 | 4.0 |
| i: [$t_1$, $t_{10}$, $t_{12}$, $t_{14}$, $t_{16}$] | | 0.0 |
| j: [$t_1$, $t_{10}$, $t_{12}$, $t_{15}$, $t_{16}$] | $t_{15}$: 2 | 2.0 |
| k: [$t_1$, $t_{10}$, $t_{13}$, $t_{14}$, $t_{16}$] | | 0.0 |
| l: [$t_1$, $t_{10}$, $t_{13}$, $t_{15}$, $t_{16}$] | $t_{15}$: 2 | 2.0 |

**Preference satisfaction degree**

Each solution satisfies the relevant contextual preference specifications to a different degree. By considering both the softgoal and hardgoal preference scores, the preference satisfaction degree of each candidate solution is derived.

**Definition 12** (Preference satisfaction degree). Given a candidate solution *sol*, a preference satisfaction degree *psd(sol)* is the sum of the softgoal and hardgoal preference scores of *sol*, *psd(sol)* = *sps(sol)* + *hps(sol)*.

**Example 6**. Table 4.3 shows the candidate solutions of the requirements problem in Figure 4.9 with their preference satisfaction degrees. The sum of the derived *sps* and *hps* of each solution shown in Table 4.1 and 4.2 is computed. The solutions are arranged from the highest to lowest $psd$.

## 4.5 DLV-ReqVar: Automated Support Tool

A powerful method for declarative knowledge representation and reasoning provided by Answer Set Programming (ASP) (Leone & Ricca, 2015) is utilised to automate both the generation of valid solutions and calculation of the preference satisfaction degrees

Table 4.3: The ranked solutions based on their preference satisfaction degrees

| | Candidate solution (*sol*) | *psd(sol)* |
|---|---|---|
| 1 | f: [$t_1$, $t_6$, $t_{12}$, $t_{15}$, $t_{16}$] | 12.0 |
| 2 | b: [$t_1$, $t_7$, $t_8$, $t_{12}$, $t_{15}$, $t_{16}$] | 10.0 |
| 3 | h: [$t_1$, $t_6$, $t_{13}$, $t_{15}$, $t_{16}$] | 7.0 |
| 4 | j: [$t_1$, $t_{10}$, $t_{12}$, $t_{15}$, $t_{16}$] | 7.0 |
| 5 | d: [$t_1$, $t_7$, $t_8$, $t_{13}$, $t_{15}$, $t_{16}$] | 5.0 |
| 6 | e: [$t_1$, $t_6$, $t_{12}$, $t_{14}$, $t_{16}$] | 5.0 |
| 7 | a: [$t_1$, $t_7$, $t_8$, $t_{12}$, $t_{14}$, $t_{16}$] | 3.0 |
| 8 | g: [$t_1$, $t_6$, $t_{13}$, $t_{14}$, $t_{16}$] | 2.0 |
| 9 | l: [$t_1$, $t_{10}$, $t_{13}$, $t_{15}$, $t_{16}$] | 2.0 |
| 10 | c: [$t_1$, $t_7$, $t_8$, $t_{13}$, $t_{14}$, $t_{16}$] | 0.0 |
| 11 | i: [$t_1$, $t_{10}$, $t_{12}$, $t_{14}$, $t_{16}$] | 0.0 |
| 12 | k: [$t_1$, $t_{10}$, $t_{13}$, $t_{14}$, $t_{16}$] | -3.0 |

of those solutions. A prototype tool called *DLV-ReqVar* is developed, which extends the reasoning mechanisms of DLV[3] – a state-of-the-art ASP implementation. DLV is a powerful formalism for knowledge representation and reasoning. It is widely used in research and industry, and it can handle all kinds of non-monotonic reasoning, including diagnosis and planning (Alviano et al., 2011). The DLV Java wrapper (Ricca, 2003) is specifically utilised to build the tool. The reasoning mechanisms of DLV-ReqVar presume the translation of the goal model constructs, contextual situations, contextual goal specifications, and contextual preference specifications into a disjunctive logic program appropriate for a DLV input file. DLV-ReqVar returns the derived alternative solutions to the goal model problem, and those solutions are ranked based on their preference satisfaction degrees. The DLV-ReqVar has two main components: the *checkConsistency* and *analyseSolution* modules. The pseudocodes for the two modules are given in Algorithm 1 and 2.

DLV, which stands for *datalog with disjunction*, is a declarative language that utilises the logical disjunction in its specifications. Its basic specifications consist of: *i*) an *atom* (i.e., *proposition*) expressing an atomic fact, *ii*) a *predicate* which is an atom with

[3]http://www.dlvsystem.com/

**Algorithm 1** pseudocode for the checkConsistency module

**Require:** $R$, $G_R$ and $P_R$, where $R$ is a goal model, $G_R$ is the contextual goal catalogue for $R$, and $P_R$ is the contextual preference catalogue for $R$

1: **for all** alternative solutions $sol$ in $R$ **do**
2: checkSAT($\theta_{sol}$) /* the checkSAT function defined in Algorithm 3 passing $\theta_{sol}$ which is a propositional formula of the conjunction of all context descriptors from both contextual goals and contextual preferences associated to $sol$ */
3: **end for**
4: **return** list of solutions with inconsistency

**Algorithm 2** pseudocode for the analyseSolution module

**Require:** $R^G$, $P_R$, $situ$, where $R^G$ is a contextual goal model (i.e., contextual goals are specified to $R$), $P_R$ is the contextual preference catalogue for $R$, and $situ$ is the contextual situation.

1: given $situ$, generate a valid requirements variant $\chi \subset R^G$
2: **for all** alternative solutions $sol$ in $\chi$ **do**
3: get the softgoals $sg$ that are linked to $sol$
4: **for all** $sg_i \in sg$ **do**
5: store the preference score
6: count the positive and negative contribution links of $sol$
7: **end for**
8: compute the softgoal preference score $sps(sol)$
9: get the preferred hardgoals $hg$ in $sol$
10: **for all** $hg_i \in hg$ **do**
11: store the preference score
12: **end for**
13: compute the hardgoal preference score $hps(sol)$
14: compute the preference satisfaction degree $psd(sol)$
15: **end for**
16: **return** the top-k solutions

arguments, and *iii*) a set of inference rules which are basically in the form of *head :- body*. The symbol ":-" is equivalent to the logical representation '*implies*' that points left, i.e., body *implies* head (*head* ← *body*). The *head* can be an atom, predicate, or disjunction of atoms and predicates separated by the "∨" symbol. Likewise, the *body* can be an atom, predicate, or conjunction of atoms and predicates separated by the "," symbol. This section describes the details of translating the requirements variability specifications into DLV-based specifications.

### 4.5.1 Translating goal models and contextual goals

This subsection shows how the visual representations of goals and contextual goals are translated into the input format of the DLV-ReqVar. Figure 4.10 shows guidelines on how the goal model constructs are mapped on to DLV specifications. The left portion of each frame contains the goal model constructs and related contextual goal specifications, while the right portion contains the corresponding DLV code.

Transforming an AND-decomposition is shown in Frame A. In the DLV translations, it is maintained that the goals and context instance labels start with lowercase letters, because atoms with leading uppercase letters represent variables. The non-contextual decomposition $g1$ is represented by the rule in Line 1, which states that if $g0$ is achieved, then $g1$ has to be achieved. This rule can be written as *achieve*($g1$) :- *achieve*($g0$), however, the binary predicate *achieveFrom*($g1$, $g0$) replaces the unary predicate *achieve*($g1$) to reflect the goal tree structure. The rules in Lines 2-4 represent the goal decomposition $g2$ which is associated with the contextual goal $cg1$. Line 2 states that if $g0$ has to be achieved, either $g2$ must be achieved or the context instance defined by $cg1$ must be false. The context instance of $cg1$ is false when $\neg(cntx1(val1) \wedge cntx2(val2))$. This negated instance is equivalent to $\neg cntx1(val1) \vee \neg cntx2(val2)$, which is the one used in Line 2. Line 3 and 4 indicate that if $g2$ must be achieved, both $cntx1(val1)$ and $cntx2(val2)$

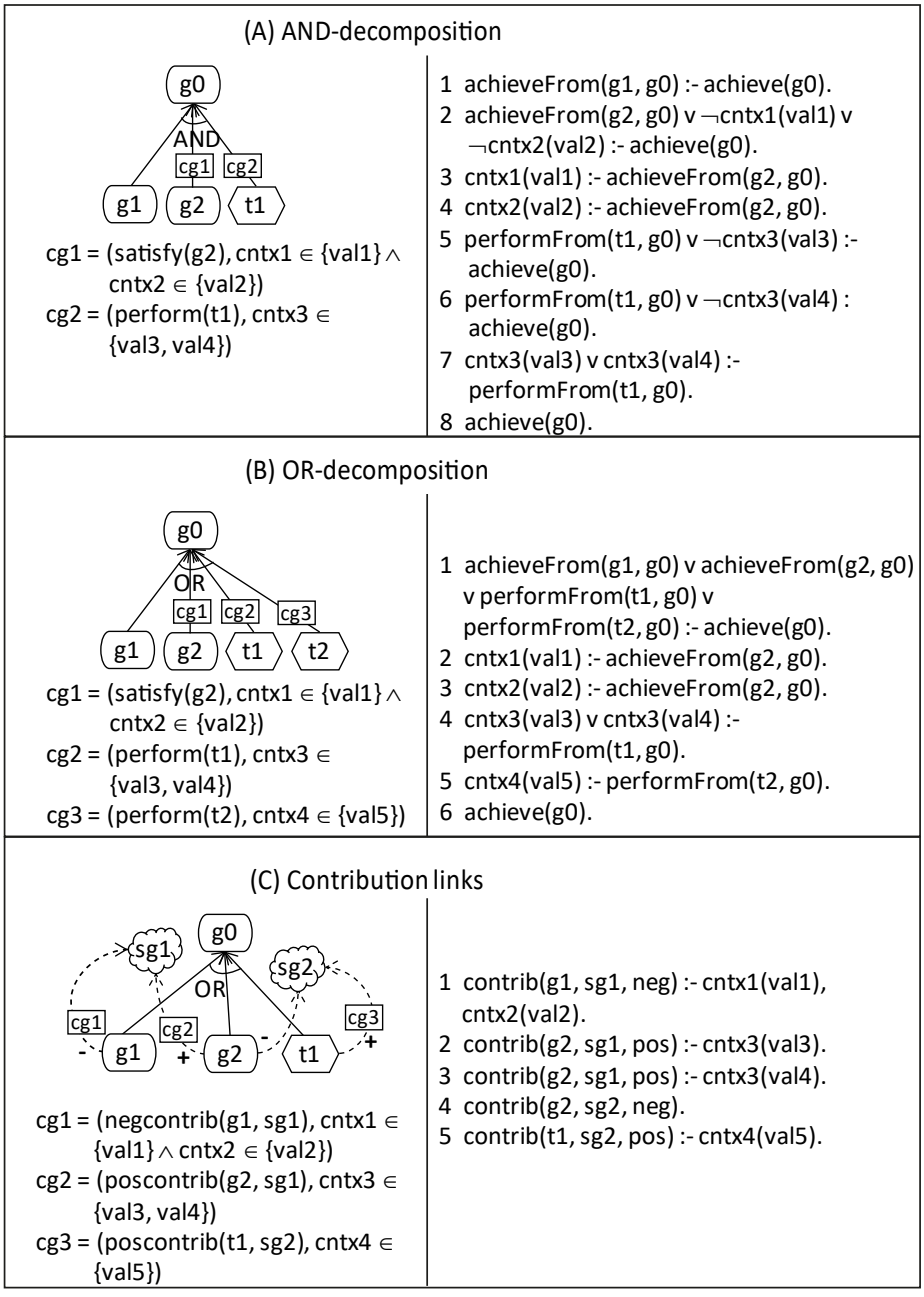



Figure 4.10: The DLV translations for goal model elements and contextual goals

must be true. Lines 5-7 represent the goal decomposition $t1$ which associates with $cg2$. The rules in Line 5 and 6 state that if $g0$ has to be achieved, then either $t1$ must be performed or the context instance defined by $cg2$ must be false. The context instance of $cg2$ is false when $\neg(cntx3(val3) \vee cntx4(val4))$. This negated instance becomes $\neg cntx3(val3) \wedge \neg cntx4(val4)$, which are the ones used in those respective inference rules. Line 7 says that if $t1$ must be performed, either $cntx3(val3)$ or $cntx4(val4)$ must be true. The predicate in Line 8 is the input for DLV to start deriving solutions: it states that $g0$ should be achieved.

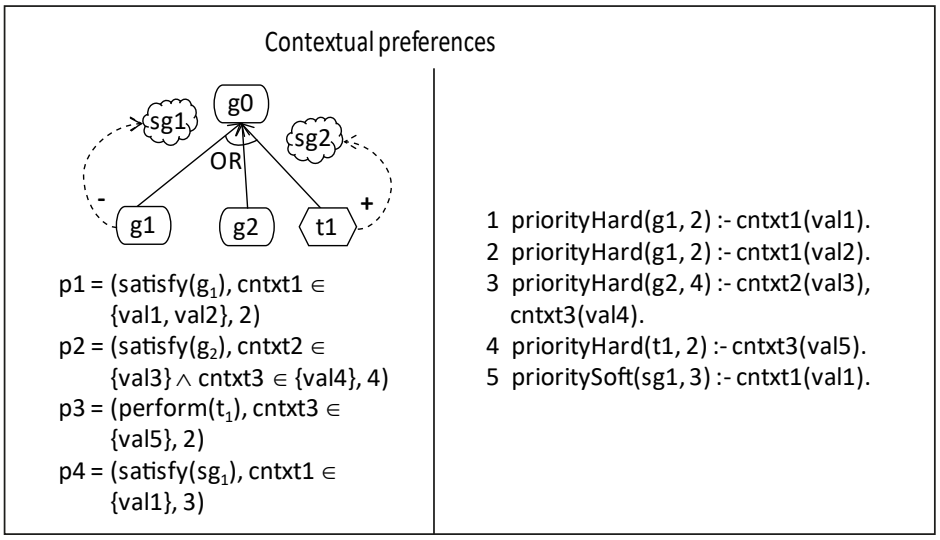



Figure 4.11: The DLV translations for contextual preferences

The OR-decomposition, as shown in Frame B, is translated into two types of rule specifications. One type specifies the decomposition shown in Line 1, and the other specifies contextual goal specifications shown in Lines 2-5. Line 1 states that if $g0$ must be achieved, then one among $g1$, $g2$, $t1$, and $t2$ must be achieved. Lines 2 and 3 represent $cg1$, stating that if $g2$ has to be achieved, then $cntx1(val1)$ and $cntx2(val2)$ must be true. Line 4 represents $cg2$, stating that if $t1$ has to be performed, then either $cntx3(val3)$ or $cntx3(val4)$ must be true. Line 5 represents $cg3$, stating that if $t2$ has to be performed, then $cntx4(val5)$ must be true. The predicate in Line 6 states that $g0$

should be achieved.

Frame C shows the translation of contribution links. Line 1 translates the contextual contribution link associated to $cg1$ stating the negative contribution from $g1$ to $sg1$ if both $cntx1(val1)$ and $cntx2(val2)$ holds. The contribution link described by $cg2$ translates to Lines 2 and 3 stating there is a positive contribution from $g1$ to $sg1$ when either $cntx3(val3)$ or $cntx3(val4)$ holds. Line 5 translates $cg3$ stating the positive contribution from $t1$ to $sg2$ when $cntx4(val5)$ holds. The predicate in Line 4 represents the non-contextual negative contribution from $g2$ to $sg2$.

### 4.5.2 Translating contextual preferences

Figure 4.11 shows the translation of contextual preferences into DLV rules. Lines 1-2 translate $p1$ stating that if either $cntxt1(val1)$ or $cntxt1(val2)$ is true, then $g1$ must be preferred with a priority score '2'. Line 3 translates $p2$ stating that if $cntxt2(val3)$ and $cntxt3(val4)$ are both true, then $g2$ is preferred with the score '4'. Lines 4 and 5 translate $p3$ and $p4$ respectively.

Defining the semantics of the goal model constructs and contextual specifications in basic DLV could allow any automated DLV-based constraint solver to render the valid solutions to a goal-modelled requirements problem. Any efficient DLV-compliant reasoning tool that emerges in the future could also be used.

### 4.5.3 Auxiliary rules

The tool uses a number of auxiliary rules to pre-process the inputs before ranking the valid candidate solutions. The sets of rules are shown in Figure 4.12. Each rule is applied to every valid solution to generate predicates to be used in the computation of preference scores. The rules in $A$ generate predicates containing the preferred hardgoals and their priority scores. These predicates are used to compute the hardgoal preference

scores. Meanwhile, the predicates generated by the rules in $B$, $C$, $D$, and $E$ are used to compute the softgoal preference scores. The rules in $B$ and $C$ identify the preferred softgoals to which a solution has a negative or positive contribution. The generated predicates contain the preferred softgoals and their priority scores. The rules in $D$ and $E$ generate ternary predicates that differentiate between goal elements associated with positive contributions, from those associated with negative contributions. This differentiation is needed since DLV does not support negative values. The rules in $F$ define unary predicates from their counterpart binary predicates. The unary predicates are useful in the other rules.

| | |
|---|---|
| A | hardPriority(T, Priority):- priorityHard(T, Priority), #int(Priority), Priority!=0, perform(T).<br>hardPriority(G, Priority):- priorityHard(G, Priority), #int(Priority), Priority!=0, achieve(G). |
| B | posContribTo(T, S, Priority) :- prioritySoft(S, Priority), Priority!=0, contrib(T, S, pos), perform(T).<br>posContribTo(G, S, Priority) :- prioritySoft(S, Priority), Priority!=0, contrib(G, S, pos), achieve(G). |
| C | negContribTo(T, S, Priority) :- prioritySoft(S, Priority), Priority!=0, contrib(T, S, neg), perform(T).<br>negContribTo(G, S, Priority) :- prioritySoft(S, Priority), Priority!=0, contrib(G, S, neg), achieve(G). |
| D | softPriority(S, Priority):- prioritySoft(S, Priority), Priority!=0, contrib(T, S, pos), perform(T).<br>softPriority(S, Priority):- prioritySoft(S, Priority), Priority!=0, contrib(G, S, pos), achieve(G). |
| E | softPriority(S, Priority):- prioritySoft(S, Priority), Priority!=0, contrib(T, S, neg), perform(T).<br>softPriority(S, Priority):- prioritySoft(S, Priority), Priority!=0, contrib(G, S, neg), achieve(G). |
| F | achieve(G) :- achieveFrom(G,_).<br>perform(T) :- performFrom(T,_). |

Figure 4.12: The pre-processing rules for the DLV-ReqVar

## 4.6 Reasoning about Contextual Goals and Contextual Preferences

This section discusses the reasoning activities involving contextual goals and contextual preferences. It highlights concerns about consistency analysis, selecting among valid alternatives at a variability point, and the iterative preference specification.

### 4.6.1 Verifying consistency of contextual goals

Early detection of modelling errors supports a well-specified requirements model that would result in an efficient development process and correctly implemented system. A consistency analysis (Ali et al., 2013) can be performed to determine whether the contexts (i.e., context instances) of the contextual goals are consistent. This analysis falls into two categories. *First*, the context of each contextual goal must be consistent. In particular, contextual goals involving more than one context element are verified. For instance, a contextual goal with the action: *flash a reminder on the TV screen*, and context descriptor: *distance-TV* $\in$ {*near*} $\wedge$ *patient-location* $\in$ {*outdoors*}, is inapplicable. Such a contextual goal is unsatisfiable since for a patient to be near the TV implies a location of being at home, and not outdoors. This kind of error has to be fixed, otherwise such goal could never be performed. *Second*, the context of each requirements variant has to be verified. The context of a requirements variant is a formula, i.e., *the conjunction of the context instances of all contextual goals in that variant*. An unadoptable variant due to unsatisfiability in context can be fixed early. In both cases, fixing errors can save costs since unadoptable variants may lead to functionality implementations that are never used or incorrectly specified.

Consistency analysis needs the specification of all possible contradictions among context variables. By context variable, it means a single value assignment for a context

element, e.g., *patient-activity=idle*. The specification of possible contradictions involves formulating *logical relations* based on domain assumptions or the hierarchical schema of context elements. Logical relations (implications) are formulated between context variables to explicitly specify the contradictions, e.g., *body-condition=normal* $\rightarrow$ *body-condition=*$\neg$*weak*, or *patient-activity=watching-TV* $\rightarrow$ *patient-location=*$\neg$*outdoors*. As it would be cumbersome to list the possible contradictions for all the context elements' values, such logical relations are formulated based on the context variables found in the goal catalogue. This avoids defining unnecessary logical relations, and only those useful for detecting inconsistencies in the current requirements model are defined.

The checkSAT algorithm shown in Algorithm 3 is applied to verify the consistency of a formula. The SAT-solver[4] is utilised to check if there exists a satisfiability in the truth values of the conjunction of *i*) the formula describing the context of a requirements variant and *ii*) the specified logical relations.

**Algorithm 3** checkSAT algorithm

```
1: Input: context θ
2: Output: ⊥(⊤) if θ is inconsistent(consistent)
3: ϵ := getLogicalRelations(θ)
4: if isSatisfiable(ϵ ∧ θ) then
5:     return ⊤
6: else
7:     return ⊥
8: end if
```

The context inconsistency of a requirements variant does not always indicate a modelling error. Although a requirements variant with context inconsistency may never be applicable, its contextual goals could also belong to other variants with satisfiable contexts. Hence, the designer must decide whether to fix or allow an inconsistency. Two cases of context inconsistencies in the requirements variants are illustrated depending on where the contradicting contexts occur.

[4]http://www.sat4j.org/

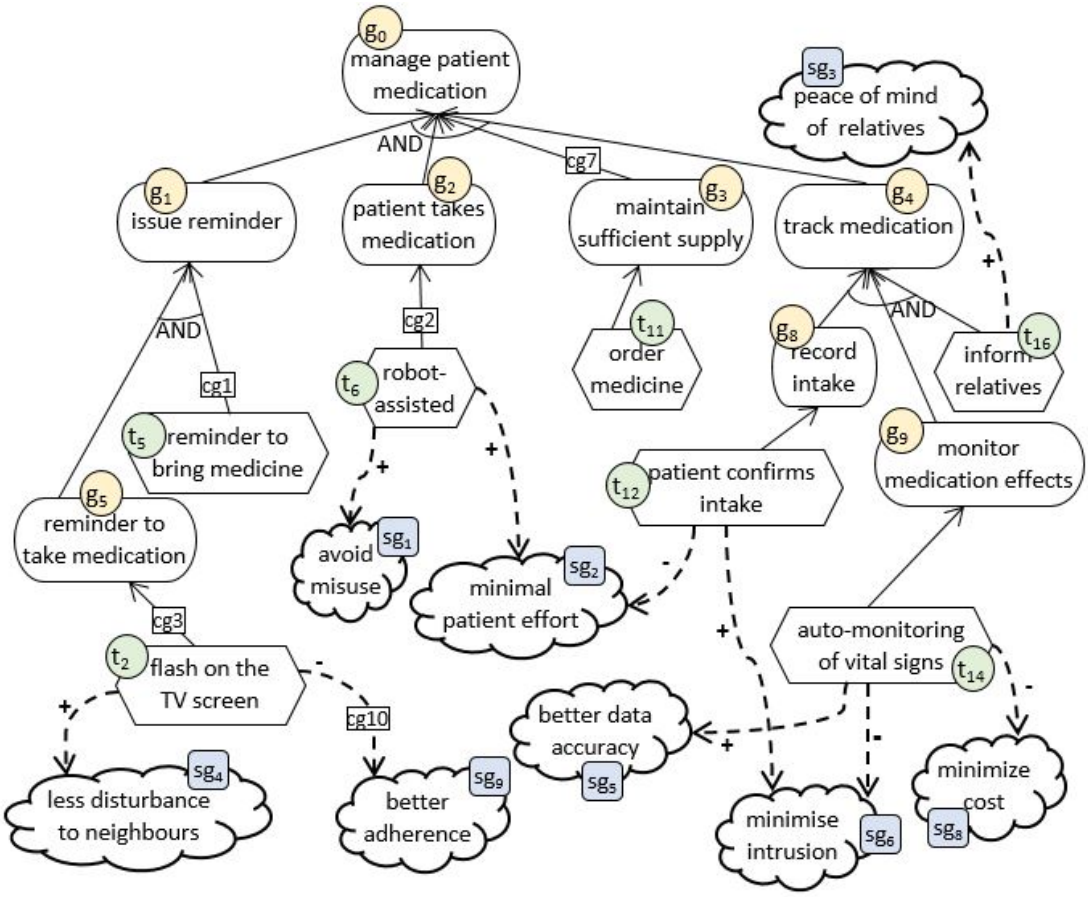
**Case 1** (Vertical inconsistency). Considering the goal decomposition structure, the context defined for an ancestor goal is inconsistent with the context of its descendant goal. This type of inconsistency is a modelling error and has to be fixed.

**Example 7**. Figure 4.13 shows an adoptable requirements variant of the personal medication assistant (i.e., the context of the variant is consistent). However, suppose a contextual goal $cg_{11}$ = (*satisfy*($g_1$), *patient-location* $\in$ {*outdoors*}) is defined. With the logical relations (*patient-activity=leaving-home* $\rightarrow$ *patient-location=home*), (*patient-location=home* $\rightarrow$ *patient-location=*$\neg$*outdoors*), and (*distance-TV=near* $\rightarrow$ *patient-location=home*), both contextual goals $cg_1$ and $cg_3$ are now unsatisfiable. Consequently, the variant in Figure 4.13, or any other variant that includes either $t_2$ or $t_5$ becomes unadoptable. This contradiction illustrates an inconsistency among the goals within the ancestor/descendant relationship. Fixing such an inconsistency becomes necessary, otherwise implementing these (contradicting) contextual goals is useless since they can never be adopted.

**Case 2** (Horizontal inconsistency). Given a requirements variant, the contexts between any two contextual goals are inconsistent, if a goal is neither an ancestor nor descendant of the other in the goal decomposition structure. This inconsistency may not be a modelling error and can be allowed by the designer.

**Example 8**. Suppose there is a contextual goal $cg_{12}$ = (*perform*($t_{14}$), *patient-location* $\in$ {*outdoors*}). Using the same logical relations defined in Example 7, the variant in Figure 4.13 becomes unadoptable since the context of $cg_{12}$ contradicts those of $cg_1$, $cg_2$, and $cg_3$. However, correcting the inconsistency is not necessary in this case. Although the variant becomes unadoptable, these contextual goals could appear in other variants with satisfiable contexts. Therefore, implementing the associated goals is not necessarily useless. A contextual goal is adoptable if it appears in at least a requirements variant with a satisfiable context.

Overall, a context inconsistency in a single contextual goal or a vertical inconsistency




The context of this variant: *cg1 (patient_activity=leaving_home) Λ cg2 (patient_location=home) Λ cg3 (distance_TV=near Λ patient_activity=¬sleeping) Λ cg7 (medicine_level=below_threshold) Λ cg10 (eye_condition=weak)*

Figure 4.13: A goal model variant and its context

in a requirements variant needs to be corrected. For example, if the context of a parent goal is inconsistent with the context of its decomposition goal, the designer can modify the associated contextual goals, or modify the goal model by removing the unadoptable decomposition. Using the DLV-ReqVar tool, gradual encoding and verifying of the goal model is beneficial. Fragments of DLV translations representing clusters of the goal model can be independently encoded and verified. As inconsistencies can be propagated to multiple variants, these are addressed early in the analysis (e.g., at the per fragment level), so that only a few contextual goals and variants are involved. The clusters are then intermittently combined following an iterative verification and modification process until the whole goal model is considered.

It is also necessary to verify the consistency of contextual preferences. Similar to the contextual goals, the consistency of a contextual preference involving more than one context element must be ensured. In addition, the context of a contextual preference associated to a hardgoal *hg* has to be consistent with the context of any contextual goal associated with *hg* and the ancestors of *hg*. Otherwise, leaving an inconsistent contextual preference as it is, makes it inapplicable.

**Example 9**. Refer to the goal model in Fig. 4.4. Suppose there is a contextual goal $cg_{13}$ = (*satisfy*($g_9$), *body-condition* ∈ {*sick*}) that would require monitoring of medication effects only when the patient is sick. A contextual preference $p14$ = (*perform*($t_{15}$), *body-condition* ∈ {*normal*}, 3) is inconsistent. Hence, $p14$ will never be used because it applies when *body-condition=normal* but $t_{15}$ is a decomposition of $g_9$ which becomes valid only when *body-condition=sick*.

### 4.6.2 Deriving preferred solutions

Considering the contextual situation and valid requirements variants in Figure 4.9 on page 206, and the applicable contextual preferences from Figure 4.7 on page 202, the

derived optimal variant defining the solution $f$: [$t_1$, $t_6$, $t_{12}$, $t_{15}$, $t_{16}$], which has a *psd score* = 12.0, is examined. The tasks $t_1$ and $t_{16}$ are required to be part of every valid solution. The goal model presents three variability points, $g_2$, $g_8$, and $g_9$, from which the tasks $t_6$, $t_{12}$, and $t_{15}$ respectively, are alternatives. These three alternatives best satisfy the preferred hardgoals and softgoals. On the one hand, two hardgoal preferences, $p4$ and $p7$, are satisfied through $t_6$ and $t_{15}$, respectively. Among the alternatives $t_6$ (*robot assisted*), $g_6$ (*patient initiative*), and $t_{10}$ (*human assisted*) to achieve $g2$ (*patient takes medication*), $t_6$ has a preference score 2.0 since the patient is alone and busy as indicated by the given situation. Likewise, between the alternatives $t_{14}$ (*auto-monitoring*) and $t_{15}$ (*manual monitoring*) to achieve $g_9$ (*monitor medication effects*), $t_{15}$ has a preference score 2.0 since the given situation indicates a patient with no illness. The hardgoal preference score of this variant $f$ totals to 4.0.

On the other hand, these three alternatives (i.e., $t_6$, $t_{12}$, and $t_{15}$), give the highest satisfaction score considering the applicable softgoal preferences ($p9$, $p10$, ..., $p13$). Preferences $p9$, $p11$, and $p12$ are satisfied through the positive contributions of both $t_{12}$ and $t_{15}$ to $sg_6$ (*minimise intrusion*, 4.0), and $t_6$ to $sg_1$ (*avoid misuse*, 3.0). Although $t_6$ also gives a positive contribution to $sg_2$ (*minimal patient effort*, 1.0), that is negated by the negative contribution from $t_{12}$ to $sg_2$. The numbers alongside the softgoals specify the preference scores. The approach derives a variant's preference satisfaction for a softgoal, based on the combined percentage of positive and negative contributions of the whole variant to that softgoal. The three alternatives give a softgoal preference score 7.0 which is added to the score derived from the contribution of $t_{16}$ to $sg_3$ (*peace of mind of relatives*, 1). Hence, the softgoal preference score of the whole variant $f$ totals to 8.0.

Even when each of the three alternatives is compared to a counterpart within its own decomposition cluster, such an alternative would stand out in terms of satisfying the contextual preferences. For instance, among the $g_2$ alternatives ($t_6$, $g_6$, and $t_{10}$),

$t_6$ contributes to both $sg_1$ (*avoid misuse*, 3) and $sg_2$ (*minimal patient effort*, 1), $g_6$ contributes to $sg_7$ (*maintain independence*, 4) but negatively contributes to $sg_2$, and $t_{10}$ contributes to $sg_2$. By score summation, $t_6$ = 4, $g_6$ = 3, and $t_{10}$ = 1. Here, $t_6$ has the highest score even without a hardgoal preference.

### 4.6.3 Gradual specification of contextual preferences

The capability of the aforementioned analysis approach to automatically derive compositions of operationalising tasks can be used to facilitate the gradual specification and refinement of both the goal model and contextual preferences. Observing the derived solutions may lead to revisions of the goal model elements, goal decomposition, and contribution links. Stakeholders may modify the contextual preferences such as adding/removing constraints, or adjusting preference scores. Additional preferences may be formulated, likewise, existing ones may be discarded. Hence, stakeholders can explore different solutions and the variations of those solutions under different situations, which would consequently affect their prioritisations over goals.

This process is illustrated using the personal medication assistant goal model in Figure 4.9 on page 206. Suppose the system is intended for users with no dementia. The users, who can live independently, are residents of a home-care facility. Without any specified contextual preferences wherein all the softgoals are equally preferred, the tool returns the following top solution: [$t_1$, $t_6$, $t_{13}$, $t_{14}$, $t_{16}$].

The designers start with specifying initial softgoal preferences for *avoiding misuse* and *ensuring patient comfort*: $p11$ = (*satisfy*($sg_1$), *patient-illness* $\in$ {*normal*}, 4), $p8$ = (*satisfy*($sg_2$), *patient-illness* $\in$ {*normal*}, 4). The tool returns two top solutions – those that contain both $t_6$ (*robot-assisted*) and $t_{13}$ (*auto-confirm intake*): [$t_1$, $t_6$, $t_{13}$, $t_{14}$, $t_{16}$] and [$t_1$, $t_6$, $t_{13}$, $t_{15}$, $t_{16}$].

However, suppose the home-care management wishes to encourage self-dependence

among its residents. This implies a higher preference for self-help medication intake, and as much as possible, minimising the use of robot assistance. The preference for *maintaining independence* is therefore specified: $p10$ = (*satisfy*($sg_7$), *patient-illness* $\in$ {*normal*}, 4). The analysis returns the same top-2 solutions, despite the expectation for solutions containing tasks under $g_6$ ($t_7$ and $t_8$). The main reason, which the designers have noticed, is the positive contribution of $g_6$ to $sg_7$ and the negative contribution of $g_6$ to $sg_2$; they cancel out each other. Hence, the suggestion to add a negative contribution link from $t_6$ to $sg_7$ is raised, but the designers opt not to include such a link, since *robot-assistance* does not totally deny a user's independence. Besides, even the context specification may still be modified since additional context elements or refinements can be realised. For instance, the stakeholders may raise an issue about residents who often get sick. Such concern leads to defining the context element *body-condition*. A revision is made to the constraint in $p_8$ so that *ensuring patient comfort* is preferred when a resident is sick: $p8$ = (*satisfy*($sg_2$), *body-condition* $\in$ {*sick*}, 4). The designers adjust the preference score and constraint for *avoiding misuse*: $p11$ = (*satisfy*($sg_1$), *patient-illness* $\in$ {*All*}, 3). Such a specification makes $p11$ a default preference. Now, the tool returns two top solutions: [$t_1$, $t_7$, $t_8$, $t_{12}$, $t_{15}$, $t_{16}$] and [$t_1$, $t_6$, $t_{12}$, $t_{15}$, $t_{16}$].

The designers are now satisfied with the result but a preference for *minimising intrusion* is added to reflect the users' concerns about privacy: $p9$ = (*satisfy*($sg_6$), *patient-illness* $\in$ {*normal*} $\wedge$ *body-condition* $\in$ {*normal*}, 4). This preference implies that auto-monitoring of medication effects and auto-confirmation of medicine intake, which would need tracking devices, are less preferred when the resident is neither demented nor sick. An additional two default preferences are also specified based on how the designers rank the importance of all the softgoals: $p12$ = (*satisfy*($sg_6$) $\bullet$ *satisfy*($sg_7$) $\bullet$ *satisfy*($sg_8$), *patient-illness* $\in$ *All*, 2) and $p13$ = (*satisfy*($sg_2$) $\bullet$ *satisfy*($sg_3$) $\bullet$ *satisfy*($sg_4$) $\bullet$ *satisfy*($sg_5$), *patient-illness* $\in$ *All*, 1). Even with these additional preferences, the same two top solutions are returned.

Furthermore, designers can specify hardgoal preferences to directly imply priorities among the alternatives per variability point. Such direct specification could firmly establish prioritisations among the alternatives in a given situation. For instance, the strongest preference to *robot-assistance*, *auto-monitoring*, and *auto-confirmation* is specified for a resident with dementia: $p1$ = (*perform*($t_6$) • *perform*($t_{13}$) • *perform*($t_{14}$), *patient-illness* ∈ {*dementia*}, 4). However, when a resident has no dementia, a *fair* preference for *manual monitoring* is specified: $p7$ = (*perform*($t_{15}$), *patient-illness* ∈ {*normal*}, 2). At every variability point, the designers can repetitively specify the preferences for the alternatives considering different perceived contexts until the preferences are settled. Considering the alternatives for $g_2$, *robot-assistance* is initially preferred only for demented patients. The only context element that has been considered is *patient-illness*. But what about other context elements such as *body-condition*, *patient-activity*, and *accompanying-people*? Are there any hardgoal preferences for the other alternatives? Answering such questions result in specifying the additional preferences such as $p2$, $p3$, ..., $p6$, presented in Figure 4.7 on page 202.

The practice of a gradual preference refinement supports an iterative process for contextual preference acquisition. Starting with some initial preferences, designers can explore potential solutions while continuously refining the preference specifications. The variations in solutions resulting from the insertion of different preferences can be observed. Overall, the iterative preference specification would help designers *i*) enhance the preference model, goal model, and context model, and *ii*) understand more profoundly the domain of the requirements being modelled.

## 4.7 Case Study

With the aid of the DLV-ReqVar tool, the goal models of three different application domains were explored by applying the approach presented in this chapter. The first

goal model defined a comprehensive requirements scenario of the personal medication assistant, an extended version of the one being used in the preceding discussions. The other two were the contextual goal models of a smart-home caregiving system for dementia patients, and a museum-guide system for managing museum visitors. These requirements scenarios represent real-world cases that have been previously used in the literature. The requirements scenarios of the latter goal models were described in the work of Ali et al. (2013) and Ali et al. (2010), respectively. This section presents the experiences of applying the approach used to refine the three goal models, and addresses the following questions:

**Q1**. Is the approach able to identify inconsistencies among contextual goals and contextual preferences? This question concerns the use of the approach in supporting designers to detect and fix inconsistencies, thus maintaining the correctness of the goal model, contextual goals, contextual preferences, and context specifications.

**Q2**. Does the approach enable the refinement of preferences? It is interesting to know how the approach can help designers refine the contextual preferences, so that given a situation, the most appropriate alternative in every variability point is selected as part of the generated solution.

**Q3**. Are contextual preferences relevant to the goal model analysis? To justify the significant effort in specifying contextual preferences, the applicability and relevancy of contextual preferences in the reasoning process are explored. It is interesting to assess if their impact on the analysis is worth the effort of specification. Likewise, it is also important to know if there are significant differences between solutions applying contextual preferences and the solutions derived when there are no preferences, or when the preferences do not depend on context.

Despite the lack of third party involvement in this usability evaluation, the case study demonstrates a systematic evaluation approach. It describes in detail *i*) the goals of the evaluation, *ii*) the activities carried out for the evaluation, *iii*) the evaluation

Table 4.4: The results of applying the approach to the personal medication assistant

| | Goal model size | | | | | Context specification | | | Contextual variability | | Solution validation | | Number of alterations | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | NG | NT | NSG | NCL | NS | NCD | NCV | NLR | NCG | NCP | NI | NU | GM | CG | CP | CS |
| Cluster 1 | (3) 4 | (9) 10 | (2) 2 | (4) 8 | (24) 28 | (12) 12 | (46) 50 | (30) 40 | (14) 13 | (8) 13 | (3) 1 | 7 | 7 | 4 | 8 | 14 |
| Cluster 2 | (6) 5 | (5) 8 | (4) 4 | (9) 9 | (7) 10 | (12) 12 | (50) 50 | (40) 45 | (4) 7 | (6) 7 | (2) 0 | 3 | 6 | 3 | 4 | 7 |
| Cluster 3 | (5) 6 | (6) 9 | (4) 6 | (5) 9 | (2) 10 | (12) 13 | (50) 54 | (45) 45 | (0) 3 | (11) 9 | (3) 1 | 5 | 12 | 2 | 6 | 5 |
| Cluster 4 | (1) 1 | (1) 1 | (0) 0 | (0) 0 | (1) 1 | (13) 13 | (54) 58 | (45) 46 | (0) 1 | (0) 0 | (2) 0 | 0 | 0 | 1 | 0 | 5 |
| Cluster 1+2 | (9) 9 | (18) 18 | (6) 6 | (17) 17 | (280) 280 | (13) 13 | (58) 61 | (46) 49 | (20) 20 | (20) 22 | (1) 1 | 1 | 1 | 0 | 2 | 6 |
| Cluster 3+4 | (7) 8 | (10) 10 | (6) 6 | (9) 9 | (20) 20 | (13) 13 | (61) 61 | (49) 49 | (4) 4 | (10) 10 | (1) 1 | 1 | 0 | 0 | 1 | 1 |
| All clusters | (17) 17 | (28) 28 | (11) 11 | (26) 26 | (5,600) 5,600 | (13) 14 | (61) 61 | (49) 51 | (24) 24 | (32) 34 | (2) 2 | 0 | 0 | 0 | 1 | 1 |
| | | | | | | | | | | | | Total: | **26** | **10** | **22** | **39** |

*the parenthesised values are taken prior to any alteration of the initial model or specification

NG: number of goals
NT: number of tasks
NSG: number of softgoals
NCL: number of contribution links
NS: number of solutions

NCD: number of context dimensions
NCV: number of context variables
NLR: number of logical relations

NCG: number of contextual goals
NCP: number of preferences

NI: number of inconsistencies
NU: number of unexpected top solutions

GM: goal models
CG: contextual goals
CP: contextual preferences
CS: context specifications

Table 4.5: The results of applying the approach to the museum-guide system

| | Goal model size | | | | | Context specification | | | Contextual variability | | Solution validation | | Number of alterations | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | NG | NT | NSG | NCL | NS | NCD | NCV | NLR | NCG | NCP | NI | NU | GM | CG | CP | CS |
| Cluster 1 | (3) 2 | (3) 3 | (0) 0 | (0) 0 | (1) 2 | (8) 8 | (18) 18 | (20) 21 | (4) 3 | (2) 2 | (0) 0 | 2 | 3 | 1 | 4 | 2 |
| Cluster 2 | (4) 4 | (4) 4 | (0) 0 | (0) 0 | (2) 2 | (8) 8 | (18) 18 | (21) 23 | (2) 0 | (1) 1 | (1) 0 | 3 | 2 | 2 | 5 | 2 |
| Cluster 3 | (3) 3 | (4) 4 | (0) 0 | (0) 0 | (4) 4 | (8) 9 | (18) 18 | (23) 23 | (0) 0 | (4) 3 | (2) 0 | 3 | 0 | 0 | 3 | 2 |
| Cluster 4 | (3) 3 | (3) 3 | (1) 1 | (3) 3 | (3) 3 | (9) 10 | (18) 19 | (23) 23 | (5) 3 | (2) 2 | (2) 1 | 1 | 0 | 3 | 1 | 4 |
| Cluster 5 | (3) 3 | (4) 4 | (2) 2 | (4) 4 | (4) 4 | (10) 10 | (19) 21 | (23) 27 | (2) 2 | (2) 2 | (1) 0 | 0 | 1 | 1 | 1 | 6 |
| Cluster 6 | (8) 9 | (16) 10 | (0) 0 | (0) 0 | (16) 24 | (10) 10 | (21) 21 | (27) 27 | (4) 4 | (3) 2 | (2) 0 | 2 | 7 | 2 | 3 | 1 |
| Cluster 7 | (3) 3 | (5) 5 | (3) 3 | (8) 8 | (2) 2 | (10) 10 | (21) 22 | (27) 33 | (2) 2 | (2) 2 | (1) 0 | 1 | 2 | 2 | 3 | 6 |
| Cluster 4+5 | (7) 7 | (7) 7 | (3) 3 | (7) 7 | (9) 9 | (10) 10 | (22) 22 | (33) 33 | (5) 5 | (4) 5 | (1) 1 | 0 | 2 | 0 | 0 | 1 |
| Cluster 2+3+4+5 | (15) 14 | (15) 15 | (3) 3 | (7) 7 | (15) 15 | (10) 10 | (22) 22 | (33) 33 | (5) 8 | (9) 9 | (1) 1 | 1 | 1 | 3 | 1 | 0 |
| All clusters | (28) 29 | (33) 33 | (5) 5 | (15) 15 | (6,912) 6,912 | (10) 10 | (22) 22 | (33) 33 | (17) 18 | (15) 15 | (1) 1 | 0 | 0 | 1 | 2 | 0 |
| | | | | | | | | | | | | Total: | **18** | **15** | **23** | **24** |

*the parenthesised values are taken prior to any alteration of the initial model or specification

NG: number of goals
NT: number of tasks
NSG: number of softgoals
NCL: number of contribution links
NS: number of solutions

NCD: number of context dimensions
NCV: number of context variables
NLR: number of logical relations

NCG: number of contextual goals
NCP: number of preferences

NI: number of inconsistencies
NU: number of unexpected top solutions

GM: goal models
CG: contextual goals
CP: contextual preferences
CS: context specifications

results, and *iv*) the details of both data recording and collection of the various constructs that were observed and measured.

### 4.7.1 Answering Q1 and Q2

To answer both questions, four steps of activities for each goal model are performed:

- Step 1, building the initial goal model. This step also defines the associated

Table 4.6: The results of applying the approach to the caregiving system

| | Goal model size | | | | | Context specification | | | Contextual variability | | Solution validation | | Number of alterations | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | NG | NT | NSG | NCL | NS | NCD | NCV | NLR | NCG | NCP | NI | NU | GM | CG | CP | CS |
| Cluster 1 | (6) 6 | (8) 7 | (1) 1 | (2) 2 | (6) 6 | (8) 8 | (18) 18 | (10) 10 | (5) 5 | (2) 2 | (1) 0 | 4 | 1 | 3 | 5 | 3 |
| Cluster 2 | (1) 1 | (2) 2 | (2) 2 | (4) 4 | (2) 2 | (8) 9 | (18) 20 | (10) 14 | (3) 3 | (4) 3 | (0) 0 | 2 | 0 | 1 | 3 | 7 |
| Cluster 3 | (7) 6 | (6) 6 | (0) 0 | (0) 0 | (8) 8 | (9) 10 | (20) 20 | (14) 15 | (3) 2 | (2) 2 | (1) 0 | 0 | 3 | 3 | 2 | 2 |
| Cluster 1+2+3 | (15) 15 | (15) 15 | (3) 3 | (6) 6 | (168) 168 | (10) 10 | (20) 21 | (15) 15 | (12) 12 | (7) 7 | (0) 0 | 0 | 0 | 0 | 1 | 1 |
| | | | | | | | | | | | | Total: | **4** | **7** | **11** | **13** |

*the parenthesised values are taken prior to any alteration of the initial model or specification

NG: number of goals
NT: number of tasks
NSG: number of softgoals
NCL: number of contribution links
NS: number of solutions

NCD: number of context dimensions
NCV: number of context variables
NLR: number of logical relations

NCG: number of contextual goals
NCP: number of preferences

NI: number of inconsistencies
NU: number of unexpected top solutions

GM: goal models
CG: contextual goals
CP: contextual preferences
CS: context specifications

requirements specifications: contextual goals, contextual preferences, and context specifications.

- Step 2, defining the logical relations and implications among the context specifications. The defined logical relations are transformed into their conjunctive normal form (CNF)[5]. Then for each contextual goal, the associated context descriptor is transformed into its CNF. Likewise, a CNF transformation is performed for the context descriptor of each contextual preference. At this stage, the inconsistency is checked within individual context descriptors involving multiple context elements.
- Step 3, subdividing the goal model into smaller clusters. For each cluster, the following tasks are performed:
  - Task 1, transforming the goal model cluster, including the associated contextual goals and contextual preferences, into a DLV input format.
  - Task 2, running the tool to check for contextual goal and contextual preference inconsistencies.
  - Task 3, performing necessary alterations to the goal model and requirements specifications to address a detected inconsistency. Repeat Tasks 2 and 3 until all inconsistencies are addressed.
  - Task 4, running the tool to generate alternative solutions and ranking them using their derived preference satisfaction degrees in various contextual situations.
  - Task 5, performing the necessary alterations to the goal model and requirements specifications. When there is any change made to a contextual goal or contextual preference (i.e., that includes adding a new one or removing an existing one), perform Tasks 2 and 3. Tasks 4 and 5 are repeated until the designers are satisfied with the generated top solutions in all the given

[5]https://doi.org/10.1007/978-0-387-30164-8_158

contextual situations.

- Step 4, combining the individual clusters and performing Tasks 2, 3, 4, and 5. This step performs the iterative cluster integration, consistency check, and solution validation until all clusters are combined into a unified goal model.

The clustering of the goal models is straightforward since any non-leaf goal can be the root node of a cluster, and all descendant nodes and the associated contribution links, softgoals, contextual goals, and contextual preferences will belong to that cluster. For example, a cluster from the goal model in Figure 4.13 on page 224 can be composed of $g_2$ as the root node, all descendant goals and tasks of $g_2$ (i.e., $t_6$), and all the associated softgoals and contextual goals (i.e., $sg_1$, $sg_2$, and $cg_2$).

The approach is examined whether it is helpful in both *i*) detecting inconsistencies that have been overlooked in the initial specifications and *ii*) refining contextual preferences to produce the expected solutions. Hence, the initial models and specifications are ensured to be free from inconsistencies as much as possible. The initial contextual preference specifications should also comprehensively capture the possible priorities in the application domain. To this end, significant efforts have been exerted in building the initial goal models and specifications. Unlike the personal medication assistant goal model which is built from scratch, the baselines of the initial goal models for the smart-home system and museum-guide system are adopted from the pre-built models found in the work of Ali et al. (2013) and Ali et al. (2010), respectively. Since initial goal models already exist for the latter systems, more focus is given to the specification of their respective contexts, contextual goals, and contextual preferences.

Tables 4.4, 4.5, and 4.6 summarise the results of the iterative validation and enhancement of the goal models from the initial ones. The tables report the properties of both the initial and final refined goal models at each cluster, as well as with the combined clusters. The columns under the goal model size describe the number of

goals (NG), tasks (NT), softgoals (NSG), contribution links (NCL), and solutions (NS). The context specification columns describe the number of context dimensions (NCD), context variables (NCV), and logical relations (NLR). For the context specification columns, each row considers the total number for the whole goal model. The contextual variability columns describe the number of contextual goals (NCG) and contextual preferences (NCP). It is also reported in the solution validation columns, the number of inconsistencies (NI) detected and the number of unexpected top solutions (NU) encountered. The last set of columns describes the number of changes made to the goal model (GM), contextual goals (CG), contextual preferences (CP), and context specification (CS). Such changes were made to the initial model and specifications while applying the approach.

Changes made to the goal models and contextual goals, triggered by applying the approach, are counted. A change in the goal model can be the removal, addition, or modification of a component or set of adjacent components resulting from a detected inconsistency or goal model construct error. A change in the contextual goals involves removing or modifying an existing contextual goal, or adding new ones. A change in the goal model may result in a change to the contextual goals or contextual preferences. For instance, removing a goal model element discards the associated contextual goal or preference. Likewise, adding a goal model element, e.g., a decomposition goal or contribution link, may lead to specifying new contextual goals or preferences. Most changes to the goal models and contextual goals occur during the DLV transformation and solution validation at the cluster level.

Similarly, the changes made to the contextual preferences and context specifications are counted. A change in the contextual preferences involves removing or modifying an existing contextual preference, or adding new ones. Such changes are caused by a detected inconsistency, or an unexpected top solution resulting from the iterative solution validation. The majority of the contextual preference modifications result from

the iterative solution validation at the cluster level. Meanwhile, changes in the contextual specifications involve removing or modifying existing context specifications (including the logical relations), or adding new ones. A change in the context specification may trigger a change to a contextual goal or contextual preference associated with that context specification. Reciprocally, there are changes in the contextual goals and contextual preferences effecting necessary changes in the context specifications.

The results indicate a significant number of alterations performed to refine the requirements specifications while applying the approach. Although alterations have started at the DLV-transformation task (Task 1), more number of alterations are triggered during the consistency check and validation tasks. On the one hand, performing Tasks 2 and 3 has led to the discovery of several inconsistencies among the contextual goals and preferences. The NI columns report the number of conflicting pairs of contextual specifications within a cluster. It is noted that in performing Step 2, no inconsistency has been encountered within any context descriptor, maybe because such an inconsistency would be noticeable and corrected in the initial specifications. Notice also in Table 4.4 that the two horizontal inconsistencies remain in the model. On the other hand, performing Tasks 4 and 5 has enabled the refinement of particularly, the contextual preferences. The NU columns report the total count of instances that the derived top solutions are not as expected for a given contextual situation. For instance, when the situation says *patient-illness* is *dementia* and *body-condition* is *sick*, a top solution containing the task *patient's own initiative of taking medication* is unexpected. In this situation, either a *human-assisted* or *robot-assisted* medication would be expected. These unexpected top solutions have prompted further tuning of the contextual preferences until the expected ones are derived. Most inconsistencies and unexpected top solutions appear at the cluster level, which is similarly observed with the number of alterations. The initial goal model and requirements specification at the cluster level come from the unrefined versions of both. It is understood that for a

combined cluster, its initial model and specifications are already the refined individual clusters. Hence, more needed refinements are expected with the individual clusters than with the combined ones.

The iterative cluster-based approach indeed has been useful to the specification, analysis, and refinement of the goal models and requirements specifications. But despite the potential effectiveness of the approach, the following difficulties were encountered. First, the manual transformation of the goal models and related requirements specifications into the correct DLV format took a significant amount of time. For instance, it took about an hour to transform the initial goal model of Cluster 1 of the museum-guide system (see Table 4.5), i.e., including the initial specifications of both context and contextual variability. However, each of the succeeding clusters of the museum-guide system took only about 15 minutes, since transforming the initial specifications for context and contextual variability was already done in Cluster 1. Familiarity with the transformation rules would reduce the time spent and transformation errors committed. Second, both the manual definition of logical relations among the context variables and the transformation of these logical relations to CNF were error-prone. Likewise, mistakes were often made on the context specification alterations since an alteration involves both updating the affected logical relations and the CNF transformations. Carefully defining a complete list of logical relations is essential in detecting inconsistencies. Lastly, assigning priority scores to the contextual preferences needed a significant amount of decision making. The priority scores that designers assign to each contextual preference are based on a collective thorough understanding of the application domain. Since a systematic approach of eliciting priority scores for the contextual preferences has yet to be included, the difficulty of specifying these scores increases as the number of contextual preferences grows.

### 4.7.2 Answering Q3

This subsection discusses the experiences with the aforementioned goal models demonstrating how the need to express preferences emerges from the variability points. Despite the presence of contextual goals, alternative selection problems still arise in the OR-decomposition variability points. Such problems are addressed using contextual preferences. Indeed, contextual requirements variability is maximised by the contextual preferences. In the personal medication assistant, four alternatives of issuing medication reminders to a patient are modelled: *ringing the medicine dispenser*, *flashing the reminder on a TV screen*, *using a mobile phone*, and *announcing on audio speakers*. There are specified situations when an alternative becomes adoptable (i.e., through the contextual goals). The problem occurs when deciding between two or more alternatives that become adoptable in a certain situation. There are two softgoals (less disturbance to neighbours and better adherence) associated with the alternatives, raising the questions of which softgoal is more preferred in a given situation, or when does one softgoal become preferred over the other. Likewise, by considering every relevant context, various situations can be specified when a certain alternative is preferred. The understanding of the requirements scenario is that the effective use of an alternative depends on the patient's activity, illness, and location, as well as time, distance of patient from the associated medium, and noise level. The consideration of all the factors that may influence decisions leads to the specification or modification of contextual preferences associated with the alternatives and related softgoals. Hence, reasoning about the alternatives becomes more intuitive with regard to contexts.

In the smart-home system, two alternatives to refresh air inside the home (i.e., *opening windows* or *turning-on the air ventilator*), are some of the variabilities that have been explored. The decision about which alternative to select, when both are adoptable in the current context, depends on the contextual preferences. On the one

hand, preferences to the two softgoals associated with the alternatives can be used: opening windows positively contributes to the softgoal *spend energy wisely*, while turning-on the ventilator positively contributes to the softgoal *patient privacy*. That is, which between the two softgoals is more preferred? On the other hand, there are hardgoal preferences specified for each alternative depending on the patient's activity, time of the day, weather, and outside temperature. For the smart-home goal model, most decisions regarding alternative choices depend on hardgoal preferences since there are only a few softgoals.

In the museum-guide system, one of the variabilities relates to how a visitor is informed about a piece of art in the museum. There are three alternatives for getting informed: via a terminal, via a PDA, or via museum staff. Each alternative is a contextual goal, however, two or more alternatives can become adoptable in a given situation. There are no associated softgoals, hence, specifying hardgoal preferences is necessary. Such preferences depend on contexts such as the staff availability, whether a visitor is alone or in-group, terminal and PDA availability, information type, and visitor type (either a VIP or general visitor).

Moreover, an experiment is conducted to determine if there are significant differences between the set of top-$k$ solutions that are derived applying contextual preferences and the top-$k$ solutions without preferences, or when the preferences do not depend on context. Such a difference implies the effectiveness of the contextual preferences that would establish preference specification to be a reasonable undertaking.

Utilising the personal medication assistant goal model and its contextual specification, it is interesting to observe the solutions derived in three different cases: *i*) when there are no preferences, *ii*) when preferences are fixed (i.e., the available preferences do not depend on context), and *iii*) when there are contextual preferences. A base contextual situation is assumed from which to define 20 other unique contextual situations. The contextual situations are grouped into four sets depending on the number of altered

Table 4.7: A comparison of the top-tier solutions of the personal medication assistant

| | Contextual situations | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1 context element variation | | | | | 2 context elements variation | | | | | 3 context elements variation | | | | | 5 context elements variation | | | | |
| | $s_0$ | $s_1$ | $s_2$ | $s_3$ | $s_4$ | $s_5$ | $s_6$ | $s_7$ | $s_8$ | $s_9$ | $s_{10}$ | $s_{11}$ | $s_{12}$ | $s_{13}$ | $s_{14}$ | $s_{15}$ | $s_{16}$ | $s_{17}$ | $s_{18}$ | $s_{19}$ | $s_{20}$ |
| **Dissimilarity of top-1 solutions** | | | | | | | | | | | | | | | | | | | | | |
| without preferences | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 |
| with fixed preferences | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 |
| **Dissimilarity of top-3 solutions** | | | | | | | | | | | | | | | | | | | | | |
| without preferences | - | - | 0 | 0 | - | 0 | 1 | 2 | 1 | - | 0 | - | 0 | 3 | 0 | 3 | 1 | 3 | 3 | 3 | 1 |
| with fixed preferences | - | - | 0 | 0 | - | 0 | 1 | 2 | 1 | - | 0 | - | 0 | 3 | 0 | 1 | 0 | 3 | 3 | 3 | 3 |
| **Dissimilarity of top-10 solutions** | | | | | | | | | | | | | | | | | | | | | |
| without preferences | - | - | - | - | - | 4 | - | 2 | - | - | 4 | - | - | 7 | 5 | - | - | 8 | 8 | 9 | 4 |
| with fixed preferences | - | - | - | - | - | 3 | - | 7 | - | - | 3 | - | - | 7 | 4 | - | - | 8 | 8 | 9 | 4 |

context element values of the base situation. Each set contains five variations of the base contextual situation of which any one, two, four, or five contextual values are randomly varied. The top-1, top-3, and top-10 solutions at each variation of the contextual situation are taken. Those solutions are compared against the respective top-$k$ solutions derived when the goal model has no preferences or with fixed preferences. A goal model with no preferences utilises solely the softgoal contributions to reason about solutions. In this case, all softgoals have equal importance. A goal model with fixed preferences also relies on softgoals, however, prioritisations among those softgoals are specified. The prioritisations do not change regardless of the context.

Table 4.7 shows the results of comparing the top-$k$ ranked solutions at 21 contextual situations, $s_0$ is the base situation. For each situation (having the goal model's top-$k$ solutions based on the contextual preferences), the number of solutions not found in those top-$k$ solutions that are derived using no preferences or fixed preferences, are counted. For comparing the optimal (top-1) solutions, a value 0 means the optimal solutions are similar, otherwise 1. For the top-3 and top-10 solutions, a value 0 means all top-$k$ solutions between the compared cases are the same, that is, regardless of order. A value $n > 0$ is the number of dissimilar solutions. There are situations when

the number of generated solutions is less than or equal to $k$, thus, all solutions will always belong to the top-$k$. In the table, such situations are indicated as "-" (i.e., not applicable).

The results show that a goal model specified with contextual preferences is more likely to generate a different optimal solution or set of top-tier solutions, compared to the ones generated when there are no preferences or when fixed-preferences are used. This demonstrates a significant impact of the contextual preferences in the derivation of solutions.

## 4.8 Performance evaluation

This section investigates the performance scalability of the automated analysis implemented in the DLV-ReqVar tool. The tool is run on goal models of different sizes using a machine with Intel i5-6500 CPU and 16 GB of RAM. Random portions of the personal medication assistant goal model are cloned to create multiple goal models with varying sizes. Such an approach was similarly used by Ali et al. (2013) and Liaskos et al. (2011). Cloning also applies to contextual goals or contextual preferences associated with the involved portion.

For each goal model, the performance analysis is conducted in two cases. Firstly, the time it takes to validate the consistency of every potential solution is measured, and Figure 4.14(a) shows the performance of the tool in that aspect. The first two columns report the size of the goal model in terms of the number of hardgoals and solutions. Column $N_{CG}$ shows the number of contextual goals within the AND/OR goal model structure, which excludes the contextual goals specified on contribution links. It is emphasised that a contextual goal specified on an AND-decomposition increases the variabilities in the goal model, thus, adding a number of potential solutions. Two goal models with similar AND/OR structure but different contextual goal specifications may

(a)

| Size of goal model | | $N_{CG}$ | $T_{Con}$ |
|---|---|---|---|
| $N_{HG}$ | $N_{Sol}$ | | |
| 17 | 21 | 6 | 297 |
| 17 | 42 | 10 | 625 |
| 25 | 70 | 9 | 775 |
| 30 | 448 | 16 | 4603 |
| 34 | 640 | 12 | 6185 |
| 45 | 840 | 9 | 5788 |
| 45 | 4480 | 19 | 43143 |
| 59 | 176400 | 28 | 2256144 |
| 72 | 882000 | 34 | 33076578 |
| **Legend** | | | |
| $N_{HG}$: the number of hardgoals<br>$N_{Sol}$: the number of solutions<br>$N_{CG}$: the number of contextual goals<br>$T_{Con}$: time for consistency analysis (in ms) | | | |

(b)

| Size of goal model | | | | $N_{CG}$ | $N_{CP}$ | $T_{Top}$ |
|---|---|---|---|---|---|---|
| $N_{HG}$ | $N_{SG}$ | $N_{CL}$ | $N_{Sol}$ | | | |
| 17 | 2 | 8 | 21 | 14 | 5 | 106 |
| 17 | 2 | 8 | 21 | 14 | 11 | 109 |
| 25 | 8 | 13 | 56 | 11 | 20 | 209 |
| 30 | 6 | 17 | 126 | 21 | 23 | 294 |
| 34 | 10 | 19 | 144 | 15 | 26 | 497 |
| 45 | 11 | 24 | 1008 | 12 | 22 | 1037 |
| 45 | 11 | 26 | 1008 | 24 | 34 | 2220 |
| 59 | 11 | 34 | 42336 | 37 | 43 | 77678 |
| 72 | 11 | 42 | 254016 | 44 | 53 | 635017 |
| **Legend** | | | | | | |
| $N_{HG}$: the number of hardgoals<br>$N_{SG}$: the number of softgoals<br>$N_{CL}$: number of contribution links<br>$N_{Sol}$: the number of solutions | | | | $N_{CG}$: the number of contextual goals<br>$N_{CP}$: the number of contextual preferences<br>$T_{Top}$: time to derive the top-10 solutions (in ms) | | |

(c)

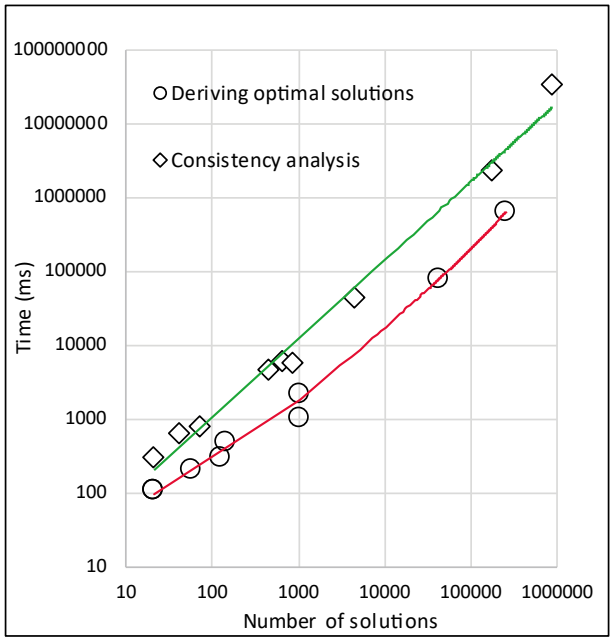



Figure 4.14: A tabular and graphical illustration of the performance of the automated tool

have a difference in their number of potential solutions. Hence, column $N_{Sol}$ refers to the number of solutions regardless of a contextual situation, i.e., the total variabilities from the *i*) OR-decompositions and *ii*) contextual goals specified on AND-decompositions.

Secondly, the time it takes to reason about contextual goals and contextual preferences is measured. The designers assume the contextual situation (*patient-activity* $\in$ {*leaving-home*} $\wedge$ *body-condition* $\in$ {*normal*} $\wedge$ *patient-location* $\in$ {*backyard-lawn*} $\wedge$ *medicine-level* $\in$ {*below-threshold*} $\wedge$ *distance-dispenser* $\in$ {*near*} $\wedge$ *noise-level* $\in$ {*low*} $\wedge$ *eye-condition* $\in$ {*normal*} $\wedge$ *accompanying-people* $\in$ {*relative*} $\wedge$ *patient-illness* $\in$ {*normal*} $\wedge$ *distance-TV* $\in$ {*near*} $\wedge$ *weather* $\in$ {*mild*} $\wedge$ *distance-phone* $\in$ {*near*} $\wedge$ *hearing-condition* $\in$ {*normal*}). Figure 4.14(b) shows the results of running the tool on the various goal models. The goal model size, in terms of number of hardgoals, softgoals, contribution links, and solutions, is shown in the first four columns. Column $N_{Sol}$ presents the number of valid solutions at the given contextual situation. Then the number of contextual goals ($N_{CG}$) and contextual preferences ($N_{CP}$) are also reported. Column $T_{Top}$ presents the time to derive the top-10 optimal solutions. The times in both columns $T_{Con}$ and $T_{Top}$ show the mean of five runs of the DLV-ReqVar tool over each goal model.

Figure 4.14(c) depicts the relationships between the number of solutions and performance, using the data results of the aforementioned cases. It shows that the computation time grows exponentially with large problem sizes. For instance, Figure 4.14(a) shows that it took 43 seconds to check the consistency of the original personal medication assistant goal model with 45 nodes having 4,480 variants, and Figure 4.14(b) shows 2 seconds to derive the optimal ones from the same goal model with 1,008 solutions. However, the bigger goal model with 72 nodes already took 9 hours for the consistency checking of its 882,000 variants, and 11 minutes to derive the optimal solutions from 254,016 variants. The automated analysis scales fairly well on the test cases with up to a few thousands of variants, of which most realistic small to medium-size goal model

problems would fall into. However, further optimisation of the tool is necessary to deal with large-size goal models. Moreover, it is important to note that for goal models with hundreds to a few thousand variants, the reasoning time to derive the optimal solutions requires only a few seconds. These results suggest the possibility of online reasoning via an interactive tool for such sizes of goal models.

It is worth mentioning that the number of nodes is not the sole factor for scalability. In the consistency analysis, the number of contextual goals is a critical attribute since the consistency checking of every requirements variant concerns those associated contextual goals. Increasing the number of contextual goals would increase computation time. In the derivation of optimal solutions, apart from the number of softgoals and contribution links, both the contextual goals and contextual preferences affect the reasoning time. However, increasing the number of contextual goals or contextual preferences does not always increase computation time. On the one hand, contextual goals constrain the space of variabilities in a given situation. Specifying contextual goals may reduce the number of valid solutions, while removing such specifications results in additional solutions. On the other hand, contextual preferences are utilised for the valid solutions. Only preferences associated with those valid solutions are significant in the reasoning process. Hence, the computation cost for a contextual preference that does not apply in any solution is negligible.

The scalability concern among large goal models affirms the usefulness of the iterative consistency checking during construction. Instead of analysing the entire goal model at once, the model is divided into clusters, and each cluster undergoes the consistency check independently. Aside from a faster combined computation time, the designer can easily locate and fix any detected inconsistency in a solution with a lesser number of associated contextual goals. Fixing an inconsistency immediately at the cluster level prevents its propagation to more solutions when considering the entire goal model.

## 4.9 Chapter Summary

Modern technologies that can sense various aspects of the environment have encouraged the growing reliance of software systems on contextual information. With the capability to recognise changes in their domains of interaction, software systems are seeking adaptability realisations, to generate better fit behaviours in response to domain changes. Requirements variability motivates adaptability, hence, understanding the influence of the domain changes (i.e., context variability) to requirements variability is necessary. This chapter has presented an approach for context-based requirements variability analysis in the goal-oriented requirements modelling. It has defined contextual goals and contextual preferences to specify the relationships of contexts with requirements and preferences, respectively. Given a requirements problem represented through a goal model, the contextual goals are used to derive applicable solutions in a given situation. Then, from the applicable solutions, the contextual preferences are used as criteria for evaluating and selecting the ones that would best satisfy stakeholder priorities. To support the variability analysis, a tool was developed to automate the derivation and evaluation of the solutions. This chapter has further demonstrated the use of the approach in detecting modelling errors and validating the impact of prioritisations, leading to improvements in the requirements specifications. The approach has broadened the scope of requirements variability by weaving context variability with both stakeholder goals and preferences, to sufficiently represent the adaptability needs of software systems where contextual changes are commonplace.

# Chapter 5

# Geographical and Functionality Aware Collaborative Filtering for Service Recommendation

Service-oriented computing (SOC) has established frameworks for creating applications through the composition of existing Web services. This facilitates reuse and development efficiency while attaining the complex functionalities that cannot be provided by standalone services (Bouguettaya et al., 2017; Sheng et al., 2014). With the remarkable developments in Internet technologies and the rise of cloud and Internet of Things (IoT) services, a rapid increase in the number of Web services (a.k.a. Web APIs) is a developing phenomenon. There is a rising number of API directories, hubs, portals, and marketplaces, allowing service providers to publish, market, and manage their APIs, as well as consumers to find, test, and connect to their APIs of interest. For example, by the end of 2019, there are already over 23,000 APIs and 7,900 mashups on ProgrammableWeb, one of the largest API directory. In the same period, RapidAPI[1], a popular API marketplace, is listing over 10,000 APIs. Additionally, the four emerging

[1] rapidapi.com

areas[2] of mobile computing, cloud computing, big data, and social computing, are actively seeking to leverage SOC in the development of their respective applications (Bouguettaya et al., 2017; Tan, Fan, Ghoneim, Hossain & Dustdar, 2016; Miorandi et al., 2012). Hence, it is expected that Web services will continue to play a core role in these emerging application domains. This robust growth of Web services conveys the emerging service economy (Tan et al., 2016), and introduces opportunities and challenges in developing Web applications through service composition.

The discovery, selection, and integration of these existing Web services (i.e., *Web service composition*), has become a practical approach for efficient applications development (Bouguettaya et al., 2017; Sheng et al., 2014). However, the growing number and diversity of Web services present challenges to service composition where it becomes difficult to discover and select the appropriate services to construct a composition. Recommendation system techniques have been explored to handle such challenges so that services can be effectively and proactively recommended to target users according to their development intentions and preferences, thus speeding up service composition. Consequently, service recommendation has become an essential process of service discovery and selection.

Most Web service recommendation approaches are built upon the analysis of a user-service matrix containing concrete QoS values. QoS values are either determined by service providers (e.g., throughput and price) or derived from user-service invocation transactions (e.g., delay, failure rate, and response time). There can also be a subjective QoS, such as user satisfaction rating (X. Wu, Cheng & Chen, 2017). Collaborative QoS prediction, which utilises the collaborative filtering (CF) recommendation approach, has become a primary concern in the literature (H. Wu et al., 2018; Xu et al., 2016; Yao et al., 2015; Zheng, Zhang & Lyu, 2014). However, the difficulty of obtaining concrete

[2]According to the IDC (www.idc.com), these four pillar technologies are the foundational elements in a digital enterprise, currently influencing the landscape of global business.

QoS values for each service invocation limits the applicability of such collaborative QoS-based approaches. Web services usually have different sets of QoS values available, and although service providers can provide some QoS values, those values may not be accurate for all users, e.g., network-based QoS. Accurate QoS values can be obtained by invoking each candidate service, but such a process is impractical as it is time and resource consuming. Such problems may lead to the unreliability and possible ineffectiveness of a purely QoS-based recommendation (Yao et al., 2018).

The most realistic and best available data is the implicit user-service invocation data, a binary data which simply tells whether a user has used a service or not. Traditional CF recommendation approaches could still become effective with this implicit data (Hu, Koren & Volinsky, 2008; Pan et al., 2008), but as the number of users and services continually grow, the presently extreme sparsity in the invocation matrix also increases. Since CF recommendation utilises the user-service invocation matrix to model latent features, such extreme sparsity negatively affects the accuracy of prediction (Kim, Park, Oh & Yu, 2017). To overcome the sparsity problem and build more effective recommendation models, it becomes necessary to explore reliable contextual factors that influence the invocation of a Web service. These contextual factors refer to the auxiliary information stretching beyond the user-service invocation matrix, i.e., the side-information of users and services, and interaction-associated information (Shi, Larson & Hanjalic, 2014). Several CF approaches have been proposed to utilise not only the invocation data but also auxiliary information such as service descriptions and tags (H. Li et al., 2017; B. Cao et al., 2017), co-invocation history (Yao et al., 2018), and temporal information (Zhong et al., 2015). However, existing service recommendation models built upon the implicit invocation data have ignored geographic information as an important contextual factor. Geographic information substantially defines the operational context of a service invocation, and the running environment of both user and service can impact the quality of invocation (Xu et al., 2016). For instance, the QoS

is affected by the connection speed on the user side and the computing power of the server on the service side.

This chapter presents a geographic-aware collaborative filtering service recommendation approach that deals with the implicit user-service invocation data. It investigates a mashup-API invocation scenario in which the user is a service composition (i.e., the mashup) invoking a Web service (i.e., the API). The matrix factorisation (MF) technique is leveraged to not just consider the invocation patterns but also recognise the varying degree of preferences behind the binary mashup-API invocation data. The degree of preference for an API varies in different invocation instances depending on the contextual factors. The impact of the contextual factors on the invocation preferences is essential for the MF in learning the latent feature vectors. The chapter particularly examines the geographic locations of mashups and APIs, which are considered to be a significant contextual factor impacting invocation preferences.

The matrix factorisation approach presented in this chapter also aims to utilise additional information (i.e., geographic location that is integrated with functionality information), to attain better recommendation performance. Although there are various approaches utilising location information as additional information in the MF-based service recommendation, these approaches are focused mainly on QoS prediction. In contrast, the approach presented in this chapter focuses on extracting the geographical and functionality information to define the context of the implicit mashup-API invocation data. Being able to define such a context will help the matrix factorisation learning process establish the degree of preference of a particular invocation.

The rest of the chapter is structured as follows. Section 5.1 presents the background concepts and motivation for the approach presented in this chapter. Aside from the discussions on fundamental concepts, Section 5.1 also provides an extensive analysis of the impact of user and Web service locations on the quality of service invocations utilising a real-world service invocation QoS dataset. Section 5.2 elaborates on the

proposed approach; it also presents a method to map geographical location information into geographical relevance scores, considering the geographical proximity between a mashup-API pair, and between a mashup-API pair and their respective neighbours. Section 5.3 presents the extensive experiments, including a comprehensive analysis, using a real-world mashup-API invocation dataset from ProgrammableWeb. Section 5.4 concludes the chapter.

## 5.1 Preliminaries and Motivation

This section contains an overview of the mashups and APIs, particularly of those from ProgrammableWeb. It also presents an in-depth analysis of the impact of the location on the quality of Web service invocation. Likewise, the basic concepts around collaborative filtering are discussed, particularly the matrix factorisation technique. Then, a discussion on implicit data is provided.

### 5.1.1 Mashups

Web services and their compositions are an emerging technology for creating automated interactions between distributed and heterogeneous applications, either for personalised consumption or for connecting business processes that span organisation boundaries. Service composition solutions bring new value not just by combining the needed component services available on the Web, but also by orchestrating them. Orchestration puts the integrated services into communication by feeding the output of one service as an input to another service, thus obtaining seamless and consistent outputs.

Among various realisations of service composition, *mashups* (Daniel & Matera, 2014; Benslimane et al., 2008) represent the data-centric, lightweight Web applications that are usually designed and developed in an agile, visual, and interactive manner. A mashup implies easy, fast integration of Web APIs (i.e., Web services) to satisfy a

broader set of requirements, collectively realising a new functionality. Although there are various definitions and interpretations of a mashup (Daniel & Matera, 2014), its intrinsic nature demonstrates the fundamental characteristics of service composition. Mashups are classified into four types (Daniel & Matera, 2014):

- Data mashup – describes a composition of disparate sources of information into a new, potentially more valuable composite data source that can be accessed and used by other services, processes, and applications. The composition can also involve some data processing activities such as cleaning, mixing, filtering, and integration.
- Logic mashup – describes a composition of logic components (e.g., Web services) at the application logic layer of the application stack. It enables the composition of functionality published as a logic or data component while mediating data compatibility issues. The main task in this composition is the orchestration of the communication between components.
- User interface (UI) mashup – describes the reuse and integration of UIs accessed from the Web at the presentation layer of the application stack, resulting in a Web application the users can interact with. This mashup synchronises the UIs of the involved components and also mediate possible data mismatches. Examples of this mashup type include HTML UI mashups, wrapped-mashups, container-based mashups, portlet-based portals, and widget-based mashups.
- Hybrid mashup – combines the data, logic, or UI mashups into a single Web application or Web service. It integrates mashup components at multiple layers of the application stack.

Although there are various definitions and interpretations of a mashup in the literature (see Daniel & Matera, 2014), its intrinsic nature demonstrates the fundamental

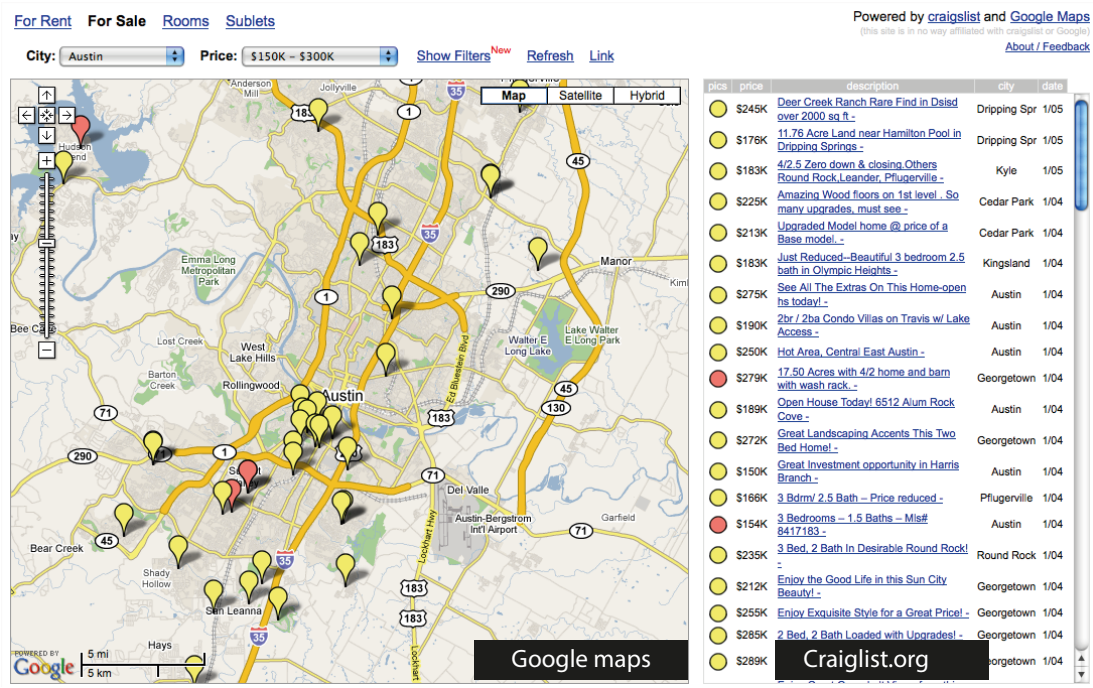



Figure 5.1: Housingmaps.com mashup integrating craiglist.org with Google maps

characteristics of a service composition. Hence, mashups are used to facilitate the discussions in this chapter. To provide an intuitive notion of mashups, Figure 5.1 illustrates the *housingmaps.com* application, one of the first mashups that emerged in 2005. It is a relatively simple but useful Web application merging the housing offers advertised on *craiglist.org* with *Google maps*. It allows a user to visualise the location of a property on a map, where as previously, real estate websites such as *craiglist.org* only provided property lists. Given a selected geographical area and price range of interest, the map on the left automatically zooms into the chosen area, while the list of offers on the right is filtered. Selecting an offer from the list will trigger a pop-up window to highlight its respective address. This integration of two previously independent resources into a new application creates *added value*. Another mashup example is *Trendsmap*[3] which integrates Twitter and Google Maps resulting in a real-time map of twitter trends. An

[3] www.trendsmap.com

example of a more complex mashup is *triprepublic*[4] – a platform for planning, collaborating, booking, and storing travel plans, which combines several Web services such as *skyscanner*, *Instagram*, *Foursquare*, *Uber*, *viator*, *musement*, and *Booking.com*.

### 5.1.2 The ProgrammableWeb

One of the largest repository of APIs and mashups – the ProgrammableWeb[5] – is examined to find out the characteristics of real-world mashup-API invocations. Crawling the website returns a list of 17,829 APIs and 6,340 mashups. Each API is described with a set of tags, short description, users (i.e., the mashups), provider information, endpoint, and Web portal. Each mashup has its description, set of tags, unique URL, and set of invoked APIs. These crawled ProgrammableWeb data will be the main source of the datasets for the approach presented in this chapter. Table 5.1 shows an example of a mashup and API from the data.

Table 5.1: An example of a mashup(a) and API(b) from the Programmable Web dataset

| **a) Mashup: SongDNA** | | **b) API: OpenStreetMap** | |
|---|---|---|---|
| **Attribute** | **Value** | **Attribute** | **Value** |
| Mashup name | SongDNA | API name | OpenStreetMap |
| Category/Tag | Music, Mobile, Lyrics, Social | Category/Tag | Mapping, Viewer |
| Description | The SongDNA iPhone app aims to be the source for song information. It collects data on any song from ... | Description | OpenStreetMap is a free wiki world map, an open volunteer-driven initiative to collaboratively create a map of the world ... |
| URL | http://songdna.me/ | Endpoint | http://api.openstreetmap.org/ |
| Component APIs | YouTube, Yahoo Image Search, MusicBrainz, Facebook, Twitter... | | |

From the crawled data, the mashup-API interaction matrix is extracted. The matrix is cleaned by dropping the duplicate interactions and rows with empty fields (i.e., a mashup associated with an API missing in the directory). The data is then reduced to 5,691 mashups and 1,170 APIs. There are only 10,737 mashup-API interactions, which show a very low matrix density of $1.6 \times 10^{-3}$. The mashup-API interaction matrix of ProgrammableWeb is extremely sparse. Most of the APIs are not used in the mashups,

[4] planner.triprepublic.com

[5] *www.programmableweb.com* is crawled in October 2018.

i.e., only seven percent of the entire API directory is linked to the mashups. Likewise, almost 95% of the mashups link to just fewer than five APIs. Figure 5.2(a) previews the API invocation count of the mashups.

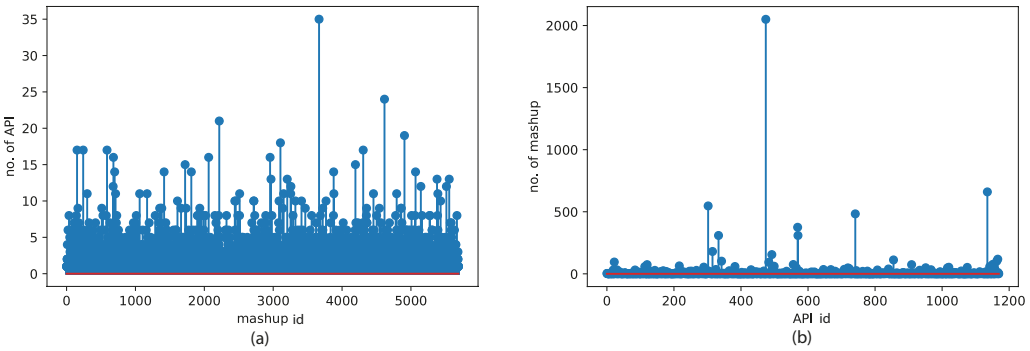


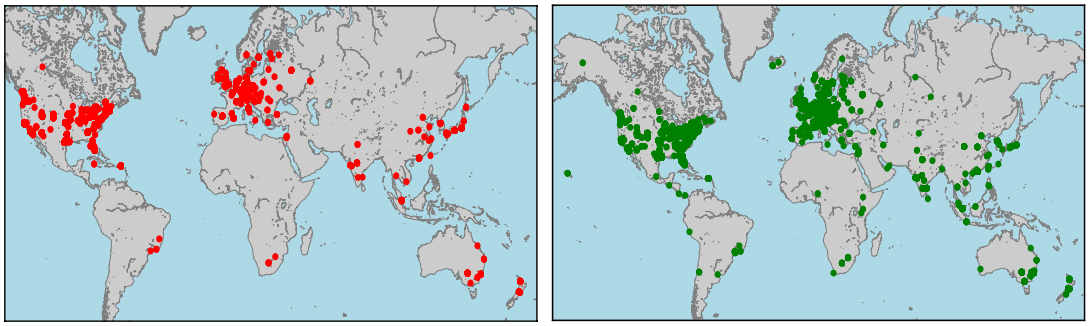

Figure 5.2: Snapshots of (a) the number of API per mashup, and (b) the number of linked mashups per API

The mashup-API interaction matrix presents a huge imbalance in the mashup-API interactions. Only a few APIs are frequently involved in the mashups, and most of the APIs are rarely used. For instance, 50% of the total mashup-API interactions involve just 11 of the most popular APIs. This is expected, since those popular (top one percent) APIs are used several hundreds of times (e.g., Google Maps is used in 2,050 mashups), while the majority (bottom 80%) are used in just less than five mashups. Figure 5.2(b) illustrates this situation.

### 5.1.3 Geographic locations of Web services

Mashups and APIs are situated in various geographical locations, e.g., a city, country, or autonomous system. These locations encompass the runtime environment and resources that support the mashups and APIs, such as the communication infrastructure, network bandwidth, CPU performance, and storage capacity (Xu et al., 2016). Given a mashup-API invocation, the respective locations of the mashup and API largely determine the operational context, which can be different for each location. It has been observed that

geographic locations significantly impact the quality ratings of service invocations and location has been considered a main ingredient in the QoS-based prediction models (see Ryu et al., 2018; Zhu et al., 2018; H. Wu et al., 2018; Tang et al., 2015).



(a) Mashup location (a) API location

Figure 5.3: Distribution of mashup and API in various geographic locations

Figure 5.3 plots the geographic locations of 5,178 mashups and 16,718 APIs listed in the ProgrammableWeb directory. There are fewer plotted mashups and APIs compared to the originally crawled ones, as several crawled items had unresolved URLs. There are a total of 1,000 different cities and 1,875 autonomous systems (AS) associated with the mashups and APIs, and these locations are spread around the world. Figure 5.4 shows an overview of the distribution count of mashups and APIs within a particular city or autonomous system.

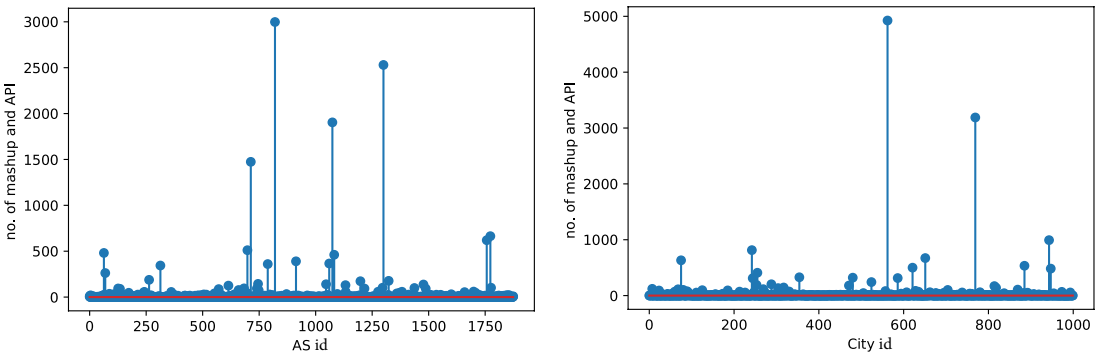
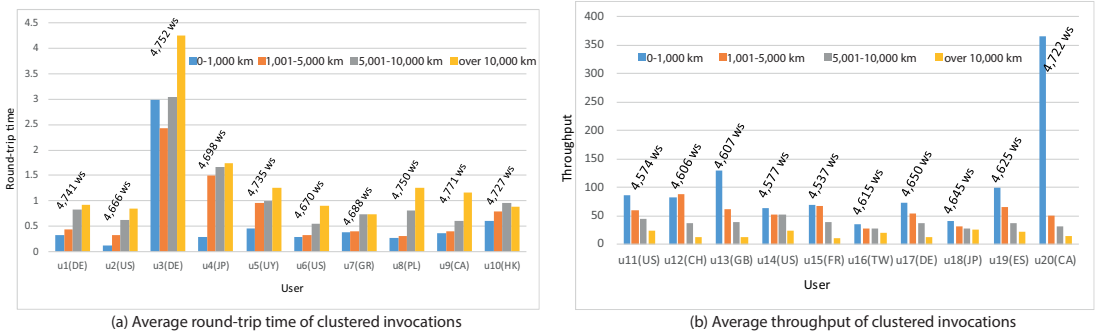



Figure 5.4: A sketch of the mashup and API count per city and autonomous system

As a rationale for the approach presented in this chapter, it is interesting to examine the behaviour of user-service invocations in the wsdream dataset (see Zheng et al., 2014).

This dataset contains real-world QoS evaluation of Web-service invocations made by 339 users on 5,825 Web services. It becomes relevant to know whether the geographic distance between users and Web services affects the quality of service invocations. Twenty (20) users are randomly selected from different geographic locations. For each selected user $u$, four clusters of its invoked Web services are created according to their geographic distance from $u$. The distance clustering are $d$={0-1,000 km, 1,001-5,000 km, 5,001-10,000 km, beyond 10,000 km}. The Equation 5.4 is used to compute the geographic distance. Figure 5.5(a) shows the average round-trip time (RTT) of the clustered service invocations for the first ten (10) selected users. The location of each user is indicated by a country code, and the total number of invoked Web services are placed above each user chart. Each user $u$ is associated with four bars that represent the clusters of Web services invoked by $u$. The users present nearly similar patterns for their average RTT per clusters. The least average RTT from the closest cluster (i.e., from those Web services located within 1,000 km of a user) is obtained. The average RTT substantially increases in the more distant clusters. Similarly, Figure 5.5(b) shows the average throughput (TP) of the clustered service invocations for the other ten selected users. A similar pattern of results is observed for the latter users. Generally, the highest average TP in the cluster of Web services located within 1,000 km of a user is obtained. The average TP decreases in the more distant clusters.



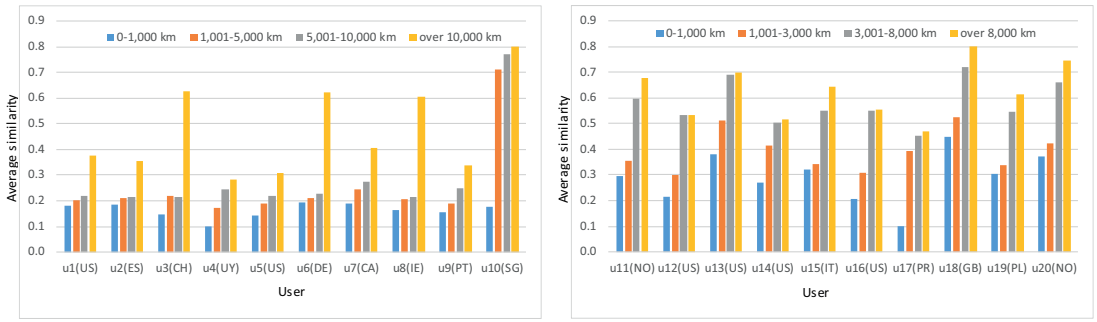

(a) Average round-trip time of clustered invocations

(b) Average throughput of clustered invocations

Figure 5.5: The user-service distance correlation with round-trip time and throughput

The quality of invocation among neighbouring users is further examined. It is interesting to see how the geographic distance of neighbouring users affects the similarity of their invocation quality when invoking the same Web service. Twenty (20) users are again randomly selected from various geographic locations. Each selected user $u$ is paired to every other user that has a commonly invoked Web service with $u$ (i.e., the neighbouring users). Since every selected user $u$ has a commonly invoked Web service with all the other users, this creates a total of 338 user-user ($u$-neighbour) pairings for each $u$. The geographic distance and similarity value for each user-user pair are computed. The similarity value is the Euclidean distance between the pair's respective QoS on commonly invoked Web services. Then, the neighbours of $u$ are clustered according to their geographic distance from $u$ using the previously mentioned distance clustering. Figure 5.6(a) shows the average similarity from the clustered neighbours of the first ten (10) selected users. A bar for each user $u$ represents the average RTT-similarity among the user-user pairings within a particular cluster. Likewise, Figure 5.6(b) shows the average TP-similarity from the clustered neighbours of the other ten selected users. Since most of the latter selected users have very few neighbours located beyond 10,000 km, the previous distance clustering is adjusted to get fairly balanced clusters: $d$={0-1,000 km, 1,001-3,000 km, 3,001-8,000 km, beyond 8,000 km}. The average similarity values are re-scaled to be within the range [0,1]. A common pattern is recognised among the 20 users regarding their RTT and TP similarity with their respective neighbours. As shown in Figure 5.6, the nearest cluster of neighbours provides the narrowest similarity both in the RTT-based and TP-based user similarity. When invoking the same Web service, neighbours located closer to a user $u$ have more similarity with $u$ in terms of their RTT and TP. The similarity relatively widens in the distant clusters.

The quality of invocation among neighbouring Web services is also examined. It is interesting to see whether the geographic distance of neighbouring services affects the similarity of their invocation quality when invoked by the same user. Twenty (20) Web

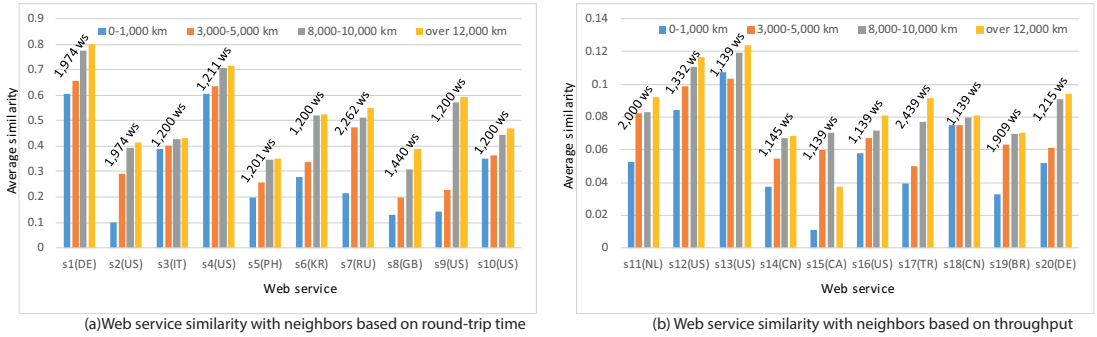



(a) user similarity with neighbors based on round-trip time

(b) user similarity with neighbors based on throughput

Figure 5.6: The user-user distance correlation with round-trip time and throughput

services are randomly selected from various geographic locations. Each selected Web service $s$ is paired with every neighbouring service. However, the number of pairings of $s$ is reduced by considering only a Web service that is invoked by all the users that invoked $s$. All users of $s$ must be in the set of users of a neighbouring service. In Figure 5.7, the numbers above the bars indicate the total service-service ($s$-neighbour) pairings for each $s$. The geographic distance and a similarity value for every service-service pair are computed. The similarity value is the Euclidean distance between the pair's respective QoS on their common users. Four (4) clusters are created from the neighbours of $s$ using their geographic distance from $s$: $d$={0-1,000 km, 3,000-5,000 km, 8,000-10,000 km, beyond 12,000 km}. Figure 5.7(a) shows the average similarity from the clustered neighbours of the first ten (10) selected services (the values are re-scaled to be within the range [0,1]). A bar for each Web service $s$ represents the average RTT-similarity among the service-service pairings within a particular cluster. Likewise, Figure 5.7(b) shows the average TP-similarity from the clustered neighbours of the other ten selected services. A relatively similar pattern is observed among the 20 Web services regarding their RTT and TP similarity with their respective neighbours. A nearer cluster of neighbours is somewhat more similar to $s$ than the farther clusters.




(a)Web service similarity with neighbors based on round-trip time

(b) Web service similarity with neighbors based on throughput

Figure 5.7: Service-service distance correlation with round-trip time and throughput

### 5.1.4 Collaborative filtering for service recommendation

Collaborative filtering (CF) predicts the preferences for the available Web services that can be invoked in a target service composition. In the mashup-API setting, a set of $m$ mashups and a set of $n$ APIs are assumed. In this chapter, special indexing letters are reserved for distinguishing mashups from APIs (for mashups $u$, $v$, and for APIs $i$, $j$). Let $\mathbf{R} \in \mathbb{R}^{m \times n}$ be a preference matrix where $r_{ui}$ denotes the preference to an API $i$ that is invoked by mashup $u$. The preference can be expressed either explicitly, such as by the QoS ratings, or implicitly, using binary values, i.e., $r_{ui} = 1$ or $r_{ui} = 0$ denotes whether or not a mashup $u$ has invoked API $i$. Traditionally, a $r_{ui} =?$ denotes that a preference to an API $i$ in mashup $u$ is either unknown or missing. Given the existing $\mathbf{R}$ matrix that represents the known mashup-API preferences, CF addresses the problem of recommending a list of APIs ranked according to the order of relevance to the target mashup. Traditional collaborative filtering (CF) methods are classified into two (Bobadilla, Ortega, Hernando & Gutiérrez, 2013; Shi et al., 2014): memory-based CF and model-based CF.

#### Memory-based collaborative filtering

Memory-based CF approaches (e.g., *neighbourhood* methods) are centred on computing relationships between users, or alternatively, between items. These approaches are

categorised as *user-based* (see Herlocker et al., 1999) and *item-based* (see Linden et al., 2003; Sarwar et al., 2001). The user-based approach estimates an unknown rating, by first finding users with similar ratings to the active user, then computing a weighted combination of those users' ratings. The $k$-nearest neighbours, the $k$ users with the highest similarity to the active user, are selected. Let $u$ be the intended mashup and $i$ an API, the estimated rating for $i$ in $u$ is:

$$\hat{r}_{ui} = \frac{\sum_{v \in U}(sim_{u,v})r_{vi}}{\sum_{v \in U} sim_{u,v}} \tag{5.1}$$

where $U$ denotes the set of $k$-nearest neighbours to mashup $u$ who rated $i$, and $sim_{u,v}$ denotes the similarity between the mashup $u$ and a neighbour $v$. The Pearson correlation coefficient and Cosine similarity are usually used to compute similarity. An analogous and more favourable approach because of better scalability and improved accuracy in many cases, is the item-based CF (R. M. Bell & Koren, 2007; Sarwar et al., 2001). A rating is estimated by finding items similarly rated by the active user, then aggregating those items' ratings. The predicted rating for an API $i$ in mashup $u$ is the weighted rating average of $i$'s $k$-nearest neighbours:

$$\hat{r}_{ui} = \frac{\sum_{j \in S}(sim_{i,j})r_{uj}}{\sum_{j \in S} sim_{i,j}} \tag{5.2}$$

where $S$ denotes the set of $k$-nearest neighbours to API $i$ (those that are invoked in $u$), and $sim_{i,j}$ denotes the similarity between the API $i$ and a neighbour $j$. Item-based approaches have a computational advantage because the item space is more stable and bounded than is user space in a typical CF setting. Likewise, item-based approaches can provide convenient explanations for the reasoning behind predictions, since users can be familiar with their previously preferred items, but it would be unusual to know like-minded users.

However, in the mashup development setting, an item-based CF may not be helpful,

since API recommendation is mostly needed at the start of development. In this case, a mashup has no prior invoked APIs to infer recommendations.

### Model-based collaborative filtering

Model-based CF approaches (e.g., latent factor models) use prediction models that are trained using the user-item rating matrix **R**, in part or whole, as input. Most approaches derive latent factor models by characterising items and users on features inferred from the rating patterns. The model-based CF has been providing more accurate recommendation results (Koren, Bell & Volinsky, 2009). The most popular technique used in model-based CF is matrix factorisation, which is known to provide better accuracy, flexibility, and scalability (Koren et al., 2009; Salakhutdinov & Mnih, 2007; Kim et al., 2017). The following discussion elaborates on matrix factorisation using the mashup-API scenario.

Matrix factorisation (MF) derives latent factor models by characterising users and services on features inferred from the rating[6] patterns. Given the observed ratings, the objective is to discover those latent features that would comprise an automatically generated alternative to the explicitly defined user and service features. Assuming there are $m$ number of users and $n$ number of services, matrix factorisation maps both users and services to a joint latent-feature space of lower dimensionality $d$ (i.e., $d \ll \min\{m,n\}$), such that user-service interactions are modelled as inner products in that space. Let $R \in \mathbb{R}^{m \times n}$ be the interaction matrix, $r_{ui}$ denotes the rating when a user $u$ has invoked service $i$. Alternatively, with the implicit invocation data, $r_{ui} = 1$ or $r_{ui} = 0$ are used to denote whether or not user $u$ has invoked service $i$. Each user $u$ is associated with a user-feature vector $x_u \in \mathbb{R}^d$, and each service $i$ is associated with a service-feature vector $y_i \in \mathbb{R}^d$. The prediction of user $u$'s rating for service $i$ is denoted by $\hat{r}_{ui} = x_u^T y_i$, i.e., the inner product of the corresponding latent feature vectors of $u$

[6]The term *rating* may refer to either the explicit QoS values or binary invocation data.

and $i$. The challenging task is learning the feature vectors, which is commonly done by minimising the sum of squared errors between the actual and predicted ratings:

$$\min_{x^*,y^*} \sum_{(u,i)\in K} (r_{ui} - \hat{r}_{ui})^2 + \lambda(\|x_u\|^2 + \|y_i\|^2) \tag{5.3}$$

where $K$ is the training set composed of $(u, i)$ pairs for which $r_{ui}$ is observed. The goal is to generalise the observed ratings to predict the unknown ones while avoiding overfitting through $L_2$ regularisation $\lambda(\|x_u\|^2 + \|y_i\|^2)$, where $\|.\|$ denotes a Frobenius norm and the coefficient $\lambda$ controls the magnitude of regularisation.

In the QoS-based MF prediction, the predicted rating $\hat{r}_{ui}$ is an estimate of the actual QoS, i.e., $\hat{r}_{ui} \approx r_{ui}$. However, with implicit invocation data, $\hat{r}_{ui}$ is interpreted as the degree of *preference* of invoking service $i$ in mashup $u$.

### 5.1.5 Implicit feedback datasets

Collaborative filtering relies on different types of input data contained in a user-item matrix. The domain-independent *explicit feedback* is the more convenient to use, for which the majority of CF recommendations are focused on. The users explicitly provide such explicit feedback in the form of ratings. The ratings can then be directly interpreted as levels of user interest or satisfaction with the items. Alternatively, ratings can translate to either positive or negative appreciation of the items. For example, the wsdream dataset contains Web services that are explicitly rated based on QoS values measured during their invocation in service compositions (Zheng, Ma, Lyu & King, 2013). A recommendation, therefore, involves predicting the unknown QoS for a given invocation instance. There are various other forms of explicit feedback such as the 5-star ratings formerly used by Netflix to rate movies or Youtube's thumbs-up/thumbs-down (like/dislike) buttons. Since any single user is likely to rate only a small percentage of items, explicit feedback datasets are usually sparse.

As discussed in Chapter 2 (Section 2.5) obtaining explicit feedback can be difficult, hence, the better availability of *implicit feedback* has attracted attention in the CF recommendation research. Implicit feedback comes from merely observing user behaviour, such as media listening or watching history, browsing history, purchase history, article reading history, or even mouse movements. More detailed feedback can also be gathered from these user-item interactions, such as time spent reading an article, number of times a media item is played, or whether a user finished reading, listening or watching an item. Moreover, implicit feedback can be complemented with contextual information that includes the available information about the users and items (Y. Li et al., 2010), and circumstances describing the user-item interaction (Adomavicius & Tuzhilin, 2011). The implicit feedback indirectly reflects user interests, hence, the direct interpretation used in explicit feedback may not apply.

In the mashup-API invocation setting, two characteristics of implicit feedback constrain the direct use of recommendation algorithms based on explicit feedback. *First*, there are no negative preferences. When an API is invoked, it can be inferred that API is probably preferred and thus has been chosen for the mashup. But reliably inferring a "disliked" API is not as straightforward. A non-invoked API does not solely mean "non-preference"; the API may be unknown to the developer or was unavailable during the mashup development. With the explicit QoS ratings, values directly translate to either positive or negative preference. For instance, a short response time up to a certain threshold value is "preferred", otherwise, beyond that threshold value is "not preferred". Accordingly, QoS-based recommendations can focus only on analysing the mashup-API pairs with known ratings, since those pairs can already provide a balanced notion of the preferences. The unrated pairs, which comprise the majority of the data, are considered *missing data*, and can be removed from the analysis. However, with implicit feedback, focusing only on the known ratings will give just positive preferences, which can lead to major misinterpretations of the interaction profile. Hence, it is important to consider

the missing data, where most of the negative preferences are likely to be found (Hu et al., 2008). *Second*, a positive preference for a mashup-API invocation is denoted by a single value, i.e., $r_{ui} = 1$ when a mashup $u$ invokes API $i$. This value does not tell anything about the degree of preference for an interaction. Knowing the degree of preference is essential to generating better recommendations. Hence, it is necessary to explore the contextual information of mashup-API invocations to infer the preference degree that will complement the implicit invocation data. It is worth mentioning that the effectiveness of a typical QoS-based recommendation relies on the explicitly expressed degree of preference, i.e., the gathered explicit QoS ratings are naturally mapped to a preference scale that runs from a positive to negative extremity.

## 5.2 Proposed Approach

The service recommendation task involves suggesting a list of suitable Web services for a potential composition, i.e., a Web application or a mashup to be developed. The proposed approach leverages the matrix factorisation (MF) model to solve the recommendation problem. Figure 5.8 illustrates the framework of the approach. The approach takes the invocation history of APIs in previous mashups, and learns the degree of preferences for those invocations. A preference degree, which is inferred from the invocation patterns while considering the influence of two contextual factors (geographic information and textual descriptions), indicates how strong the API is preferred in a mashup-API invocation. To provide more precise predictions, the contextual factors are utilised in the MF modelling in learning the latent feature vectors. Given a target mashup, the approach returns a list of potential APIs ranked according to the preference degrees. The mashup-API interaction is implicit feedback. Hence, there is no explicit interaction feedback (i.e., ratings) that is essential for matrix factorisation to generate the latent factor vectors. Instead, the inferred preferences are embedded in the mashup-API

invocation patterns, from which the joint latent factor space that characterises both mashups and APIs is derived.

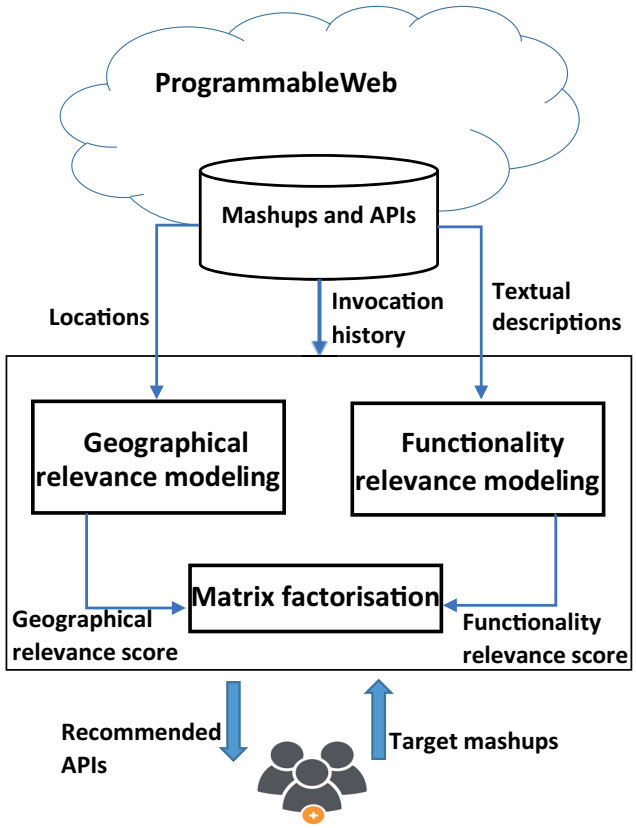



Figure 5.8: Architectural framework of the proposed approach

This section presents the details of the proposed approach. The approach is divided into three major components: *geographic relevance modelling*, *functionality relevance modelling*, and the *integration into the geographic-aware recommendation model*.

### 5.2.1 Geographic relevance modelling

From the preliminary observations presented in Section 5.1.3, three assumptions in the mashup-API service invocation scenario are formulated:

- *First*, the geographical proximity between the client (mashup) and server (API) hosts can affect the performance and quality of a service invocation (Nikravesh,

Choffnes, Katz-Bassett, Mao & Welsh, 2014). Service invocation involves interactions between the server and client hosts, and such interactions span communication paths on the Internet. The distance between communicating hosts generally indicates their network proximity. In farther distances, there are possibly more gateway nodes and hops involved in the communication network (Banerjee et al., 2014). Every node has its own routing procedure, and a routing procedure has inherent processing delays, which depend on the respective network infrastructure efficiency.

- *Second*, mashups located near each other tend to share similar contexts, e.g., the operational environment and IT infrastructure. Hence, the quality of service invocation for the same API is likely to be similar among the neighbouring mashups.
- *Third*, APIs near each other are also likely to share similar contexts, e.g., APIs provided by the same company, and hosted at the same location, usually run using the same computing platform and resources. The quality of invocation by the same mashup is likely to be similar among the neighbouring APIs.

The aforementioned assumptions are utilised to derive the geographical relevance score of a mashup-API invocation. The first assumption implies better quality of service invocation in shorter geographic distances, hence, a higher preference should be given to a potential API that is located closer to the mashup. Considering the second assumption, the preference of mashup $m$ for a potential API $a$, should also reflect the preferences of $m$'s neighbouring mashups. When $a$ is invoked by $m$'s neighbouring mashup $n$, it is expected that $a$ provides *better* quality of service invocation to $n$. When $m$ is geographically close to $n$, it is likely to get that same better quality of invocation when $m$ invokes $a$. The geographic distance $d$ between two locations is computed using the inverse haversine formula:

$$d(x,y) = 2r \cdot arcsin\left(\sqrt{sin^2\left(\frac{\varphi_x - \varphi_y}{2}\right) + cos(\varphi_y)cos(\varphi_x)sin^2\left(\frac{\gamma_x - \gamma_y}{2}\right)}\right) \quad (5.4)$$

where $r$ = 6,371 km pertaining to the Earth radius, $\varphi_x, \varphi_y \in (-180, 180]$ represent the latitudes in the respective geolocations of x and y, and $\gamma_u, \gamma_i \in (-180, 180]$ represent the respective longitudes. The geolocation information for the mashup-API dataset are derived from the GeoIP database[7] through a lookup of the mashup and API URLs.

A geographical similarity function that transforms the geographic distance $d_{ui}$ between a mashup $u$ and a potential API $i$ into a *geographical similarity* value is defined as:

$$geosim_{ui} = 2 - \frac{2}{1 + e^{(-\frac{d_{ui}}{\delta})}} \quad (5.5)$$

where $\delta$ is the dispersion factor that approximates the decrease rate of the similarity value as the distance increases. The sensitivity of the proposed model to the setting of $\delta$ is reported in the experiments. The derived similarity value indicates the similarity of the contexts of $u$ and $i$. The similarity function should satisfy two constraints. First, the function should be equal to 1 when $d_{ui} = 0$. A mashup-API pair, both on the same location, has a geographical similarity of 1, and most likely shares the same context. Second, the function tends to be 0 as the distance increases. A geographically distant mashup-API pair is least likely to have similar contexts. Figure 5.9 shows the similarity function applied to the dataset's mashup-API invocation distances.

Given a mashup $u$ and a potential API $i$, the list of nearest neighbours of mashup $u$ is obtained: $M(u) = \{m \mid m$ is within $k$ distance from $u\}$. The set of *first class neighbours* of $u$ is further defined as: $M_f(u) = \{m \mid m \in M(u)$ such that $m$ invokes $i\}$. The sensitivity of the proposed model to the setting of $k$ is reported in the experiments.

[7]The database was downloaded from https://dev.maxmind.com/ in November 2018.

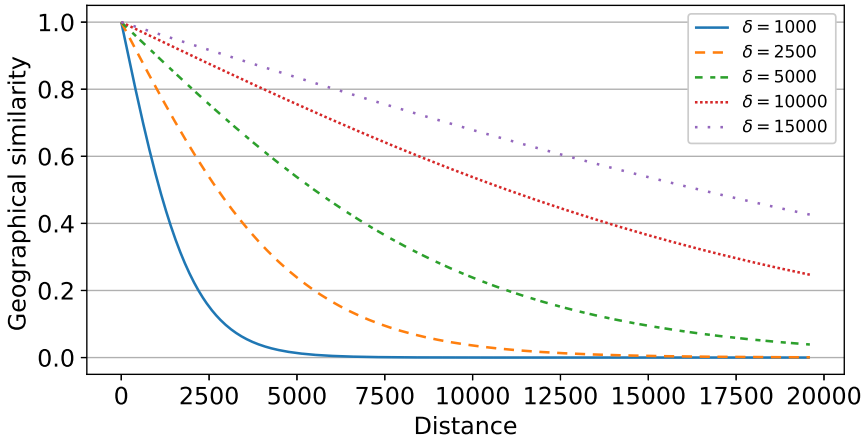



Figure 5.9: The similarity function at different values of $\delta$

There is a computation of the geographical similarity between each $m \in M_f(u)$ and $i$: $geosim_{mi}$, and between each $m \in M_f(u)$ and $u$: $geosim_{um}$. Since an $m$ invokes $i$ (i.e., $r_{mi} = 1$), $geosim_{mi} \cdot geosim_{um}$ denotes the extent of which $r_{mi}$ is preferred relative to $u$. The geographical similarity between every $m \in M_f(u)$ and $i$ into a *mashup context similarity* is aggregated as:

$$consim_f(u) = \left( \frac{1}{\mid M_f(u) \mid} \sum_{m \in M_f(u)} (geosim_{um} \cdot geosim_{mi}) \right) \cdot \left( 1 - \frac{1}{1 + \sqrt{\mid M_f(u) \mid}} \right) \tag{5.6}$$

However, when there are no first class neighbours for mashup $u$ (i.e., $M_f(u) = \varnothing$), the approach resorts to the aforementioned third assumption. The preference of mashup $u$ for a potential API $i$ alternatively considers the preferences of $u$'s *second class neighbours*, to $i$'s neighbouring APIs. The set of neighbours of API $i$ is defined as: $A(i) = \{j \mid j$ is within $k$ distance from $i$, and $m \in M(u)$ invokes $j\}$. The list of $u$'s second class neighbours is: $M_s(u) = \{m \in M(u)$ such that $m$ invokes $j \in A(i)\}$. The approach computes the geographical similarity between each invocation pair $m \in M_s(u)$ and $j \in A(i)$: $geosim_{mj}$, between each $m \in M_s(u)$ and $u$: $geosim_{um}$, and between each $j \in A(i)$ and $i$: $geosim_{ij}$. Then, the geographical similarity of each invocation

pair $m \in M_s(u)$ and $j \in A(i)$ is aggregated as:

$$consim_s(u) = \left( \frac{1}{\mid M_s(u) \mid} \sum_{\substack{m \in M_s(u) \\ j \in A(i)}} (geosim_{mj} \cdot geosim_{um} \cdot geosim_{ij}) \right) \cdot \left( 1 - \frac{1}{1 + \sqrt{\mid M_s(u) \mid}} \right) \tag{5.7}$$

where the approach captures the notion of, for instance, when $m \in M_s(u)$ invokes a $j \in A(i)$ (i.e., $r_{mj} = 1$), $geosim_{mj} \cdot geosim_{um} \cdot geosim_{ij}$ denotes the extent to which $r_{mj}$ is preferred relative to both $u$ and $i$.

The geographical relevance score between a mashup $u$ and API $i$ combines the two similarity values, the geographical similarity $geosim_{ui}$ and mashup context similarity $consim_u$:

$$g(u, i) = geosim_{ui} \cdot consim_u \tag{5.8}$$

where $consim_u = consim_f(u)$ when $M_f(u)$ is not empty, otherwise $consim_u = consim_s(u)$.

### 5.2.2 Functionality relevance modelling

#### The Latent Dirichlet Allocation

Latent Dirichlet Allocation (LDA) (Blei, 2012) defines a generative process for inferring the topic distributions by computing the latent structure that likely generated the observed document content. Figure 5.10 shows the LDA graphical model where the shaded and unshaded variables indicate observed and latent variables respectively. It is assumed there are $N$ number of words in a document, $D$ number of documents in the corpus, and $K$ number of topics. Likewise, LDA assumes that the topics are specified

before any document has been generated. In Figure 5.10, the topics are $\beta_{1:K}$ and each topic $\beta_k$ contains a distribution of the vocabulary. All documents depend on the topic distributions $\beta$, hence, it is considered a global parameter. For each document in the corpus, words are generated in a two-step process. Step 1, is to randomly select a distribution of the topics. In Step 2, for each word in the document, *i*) randomly select a topic according to the topic distribution in Step 1, then *ii*) randomly select a word from the corresponding topic's vocabulary distribution. The per-document topic distributions are $\theta$, where $\theta_d$ are the topic proportions for the $d$th document, and $\theta_{d,k}$ is the topic proportion of topic $k$ in document $d$. Document specific word-topic assignments are $Z$, where $z_d$ are the topic assignments for the $d$th document, and $z_{d,n}$ is the topic assignment of the $n$th word in document $d$. The observed words are $W$, where $w_d$ are the observed words of document $d$, and $w_{d,n}$ is the $n$th word in $d$.

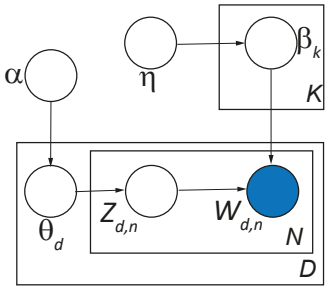



Figure 5.10: The LDA topic model

The hyperparameters $\alpha$ and $\eta$ are the Dirichlet priors to $\theta$ and $\beta$ respectively. Both priors are considered as constants in the model. A Dirichlet prior $\alpha$ smooths the topic distribution per document while $\beta$ smooths the word distribution per topic (Steyvers & Griffiths, 2006). The directed links indicate conditional dependencies between variables, and the rectangular plates indicate sampling repetitions. For example, the inner plate surrounding $Z$ and $W$ illustrates a repeated sampling of a word-topic assignment until $N$ words are generated for document $d$. The outer plate indicates a repeated sampling of a topic distribution $\theta_d$ for each document $d$ for a total of $D$ documents. Similarly,

the plate surrounding $\beta_k$ illustrates the repeated sampling of word distribution for each topic $k$ until $K$ topics are generated.

The directed links between the variables imply dependencies. For instance, the observed word $w_{d,n}$ depends on the topic assignment $z_{d,n}$ and all of the topics $\beta_{1:K}$. Implementing such dependency involves a looking up which topic $z_{d,n}$ refers to and looking up the probability of the word $w_{d,n}$ within that topic. Likewise, the topic assignment $z_{d,n}$ depends on per-document topic distributions $\theta_d$.

The generative process is represented as a joint probability distribution over both the latent (i.e., $\beta$, $\theta$, and $Z$) and observed (i.e., $W$) variables as follows:

$$p(\beta_{1:K}, \theta_{1:D}, z_{1:D}, w_{1:D}) = \prod_{k=1}^{K} p(\beta_k) \prod_{d=1}^{D} p(\theta_d) \left( \prod_{n=1}^{N} p(z_{d,n}|\theta_d) p(w_{d,n}|\beta_{1:K}, z_{d,n}) \right). \quad (5.9)$$

The joint probability distribution is used to compute the conditional distribution (i.e., posterior distribution) of the latent variables given the observed variables. Here, the latent variables are the topic structure, and the observed variables are the words in the documents. The conditional distribution is as follows:

$$p(\beta_{1:K}, \theta_{1:D}, z_{1:D}|w_{1:D}) = \frac{p(\beta_{1:K}, \theta_{1:D}, z_{1:D}, w_{1:D})}{p(w_{1:D})}. \quad (5.10)$$

An important LDA process is computing the topic distribution $\theta$ for each document as well as the word distribution $\beta$ for each topic. Since it is intractable to directly compute such variables using the conditional distribution in Equation (5.10) (Blei, 2012), the Gibbs sampling algorithm is used as an alternative method to approximate the latent distributions (see Steyvers & Griffiths, 2006). Through Gibbs sampling, both $\theta$ and $\beta$ can be inferred using the posterior distribution of parameter $Z$.

The Gibbs sampling algorithm starts by going to each document and randomly

assigning each word in the document to one of the $K$ topics. This creates a topic-assignment list. Two sparse matrices are also created: *i*) a document-topic matrix $\mathbf{\Theta}^{DxK}$ which holds the count of words assigned to each topic for each document, i.e., the topic-assignment distribution, and *ii*) a word-topic matrix $\mathbf{\Phi}^{WxK}$ which holds the count of words assigned to each topic for the entire collection. $D$ is the number of documents, $K$ is the number of topics, and $W$ is the number of unique words in the corpus.

Then, for each document $d$, the process goes through each $n$th word. A new topic $k$ is reassigned to the $n$th word where a topic $k$ is chosen considering the probability of the $n$th word given the current word distribution of $k$ and the probability of $k$ given the current topic distribution of $d$. Such an estimation of a topic assignment is mathematically expressed as follows:

$$p(z_n = k|z_{-n}, w_n, d_n) \propto \frac{\mathbf{\Phi}_{w_n k} + \eta}{\sum_{w=1}^{W} \mathbf{\Phi}_{wk} + W\eta} \cdot \frac{\mathbf{\Theta}_{d_n k} + \alpha}{\sum_{j=1}^{K} \mathbf{\Theta}_{d_n j} + K\alpha} \tag{5.11}$$

where $z_n = k$ is the assignment of the $n$th word to topic $k$, $z_{-n}$ represents topic assignments of all other words excluding the $n$th word, $w_n$ represents the index of the $n$th word in the vocabulary of $W$ unique words, and $d_n$ is the document containing the $n$th word. Furthermore, $\mathbf{\Phi}_{wk}$ is the number of times word $w$ is assigned to topic $k$, excluding the current instance $n$, and $\mathbf{\Theta}_{dk}$ is the number of times topic $k$ is assigned to some words in document $d$, excluding the current instance $n$.

Assigning words to topics depends on how likely a word is to occur in a topic, and how dominant a topic is in a document (Steyvers & Griffiths, 2006). By examining Equation (5.11), it can be understood that as more instances of a word $w$ are assigned to a topic $k$ (i.e., across all documents), the probability of assigning any instance of $w$ to $k$ is increased. Similarly, as the assignment count of a topic $k$ increases in a document $d$, the probability that a word from $d$ is assigned to $k$ also increases.

The algorithm keeps track of the count matrices $\mathbf{\Phi}$ and $\mathbf{\Theta}$ during the process of

parameter estimation. For each word instance, the corresponding entries for the current topic assignment are first decremented by one in both count matrices. After a new topic is sampled from the distribution in Equation (5.11), the corresponding entries to the new topic assignment in both matrices are incremented by one. A single pass through all documents gives one Gibbs sample, which is a set of topic assignments to all instances of the $W$ words in the corpus.

From the count matrices, the estimates of $\beta$ and $\theta$ can be obtained as follows:

$$\beta_{wk} = \frac{\mathbf{\Phi}_{wk} + \eta}{\sum_{j=1}^{W} \mathbf{\Phi}_{jk} + W\eta} \tag{5.12}$$

$$\theta_{dk} = \frac{\mathbf{\Theta}_{dk} + \alpha}{\sum_{j=1}^{K} \mathbf{\Theta}_{dj} + K\alpha} \tag{5.13}$$

where $\beta_{wk}$ is the probability of word $w$ in topic $k$, and $\theta_{dk}$ is the proportion of topic $k$ in document $d$.

**Generating topic models for service documents**

The derivation of functionality similarity between the mashups and APIs of ProgrammableWeb is based on their textual descriptions. The textual description is the most important feature that should be considered in service recommendation (Y. Cao et al., 2019). For each mashup, the description, category, tags, and related APIs are aggregated into one mashup document. Similarly for an API, the description, category, and tags are aggregated into an API document. Hence, the approach creates a corpus of service documents where each document defines a functionality vector associated with a mashup or API. It is assumed that service documents having similar topic distributions are functionally similar. The Latent Dirichlet Allocation (LDA) topic modelling (Blei, 2012) is used to analyse each document and obtain the associated topic distribution. LDA is a simple but powerful topic model that has been proved to perform well even

with very short and sparse texts such as tweets and microblogs (Naveed, Gottron, Kunegis & Alhadi, 2011; Weng, Lim, Jiang & He, 2010). In several evaluations of existing topic modelling representations, LDA often performs as the best among the competitors (T.-H. Chen, Thomas & Hassan, 2016; J. Chang, Gerrish, Wang, Boyd-Graber & Blei, 2009; Blei, Ng & Jordan, 2003). Hence, LDA topic modeling has been heavily adopted in various information retrieval and text analysis fields such as social networks, software engineering, crime science, biomedical science, and linguistic science (Jelodar et al., 2019).

Prior to applying the LDA process, some data pre-processing steps for the whole document collection are performed, such as tokenisation (splitting texts into individual words), cleaning (removal of punctuation marks, non-alphanumeric characters, and stopwords), and stemming (reducing words into their root form). Performing these pre-processing tasks utilise the methods provided by the nltk toolkit[8]. Consequently, the resulting pre-processed words in the collection serve as the LDA model's vocabulary. Figure 5.11 illustrates the pre-processing tasks applied to the corpus.

The topic distribution of each document in the corpus is derived through the LDA generative process (i.e., LDA topic modeling) implemented by gensim[9]. Finding the best performing values for the LDA parameters: $alpha$, $eta$, and the number of topics $K$, is crucial to generate reliable topic distributions (Steyvers & Griffiths, 2006). To this end, both quantitative (coherence score) and qualitative (visual) evaluation of the topic model are performed. In the coherence score evaluation[10], the coherence score of the resulting models is calculated as a varying number of topics ($K$=[5,10,15,...,100]) for every pair ($\alpha_i$, $\eta_j$) of hyperparameter values ($\alpha$=[0.1, 0.5, 1.0, 2.5, 5.0], $\eta$=[0.1, 0.5, 1.0, 2.5, 5.0]) are set. At each $K$, a coherence score is computed five times for each pair, then the mean score obtained. Figure 5.12 shows the mean coherence scores from some

[8] www.nltk.org

[9] https://radimrehurek.com/gensim/models/ldamodel.html

[10] https://radimrehurek.com/gensim/models/coherencemodel.html

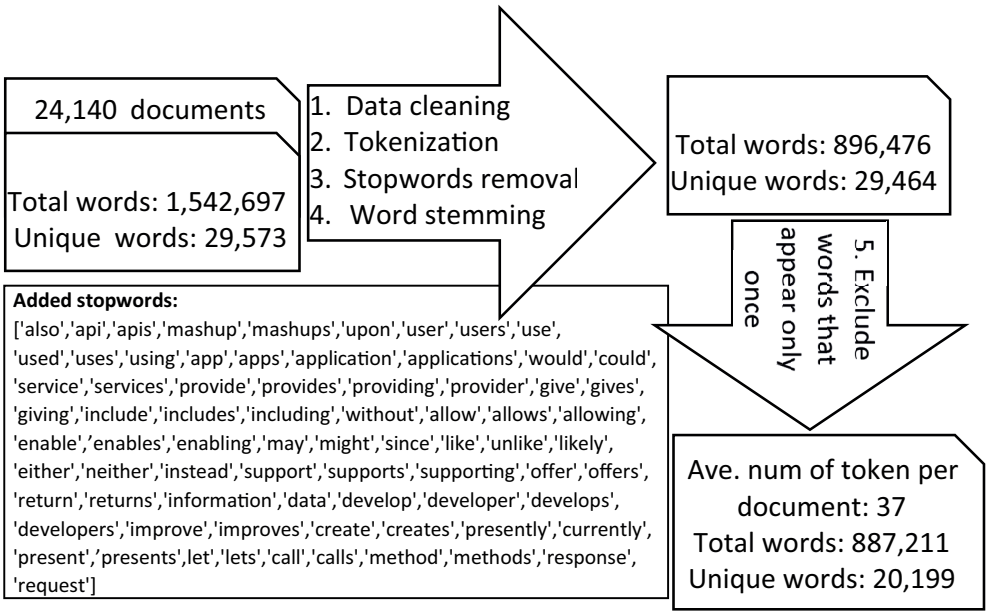



Figure 5.11: The pre-processing of the ProgrammableWeb data

hyperparameter pairs for the different numbers of topics.

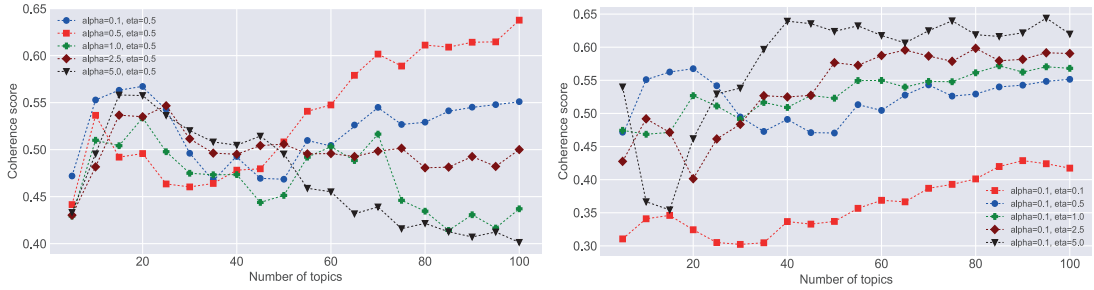



Figure 5.12: The coherence scores of $(alpha, eta)$ pairs at $K$ number of topics

To complement the coherence score, the topic clusters produced in each model are visually examined. The visual inspection utilises the pyLDAvis visualisation tool (Sievert & Shirley, 2014). Although topic coherence provides a convenient measure to assess a topic model, relying solely on the coherence score does not guarantee an optimal model. For example, many instances of the hyperparameter pairs give higher coherence scores at some $K > 60$, but their resulting topics seem hard to interpret. Likewise, their visualisations become too granular, resulting in topic overlapping where similar words are repeated in multiple topics. Figure 5.13(a) shows a snapshot of the intertopic

distance map of the 20 topics generated by a model (i.e., $alpha$ = 0.1, $eta$ = 0.5, and $K$ = 20). The topics are represented by the bubbles numbered according to size (i.e., the percentage of assigned words). The size indicates a topic's relative prevalence in the corpus. The plot is rendered through a multidimensional scaling algorithm where similar topics should appear close to each other. It seems the generating model gives a desired visualisation with those fairly sized non-overlapping bubbles scattered throughout the chart quadrants. Figure 5.13(b) shows a horizontal barchart with the topmost relevant words assigned to the selected topic indicated by the red bubble (i.e., Topic 2). By examining these words, Topic 2 can be interpreted as about places and travels. The words can be arranged according to both their probability within the topic and their topic exclusivity (i.e., the likelihood that a word occurs only in the particular topic). For each pair of overlaid bars, the red bar represents the topic-specific frequency of a word and the bluish one represents the corpus-wide frequency. This visualisation provided by pyLDAvis supports the tuning process towards obtaining a better topic model.

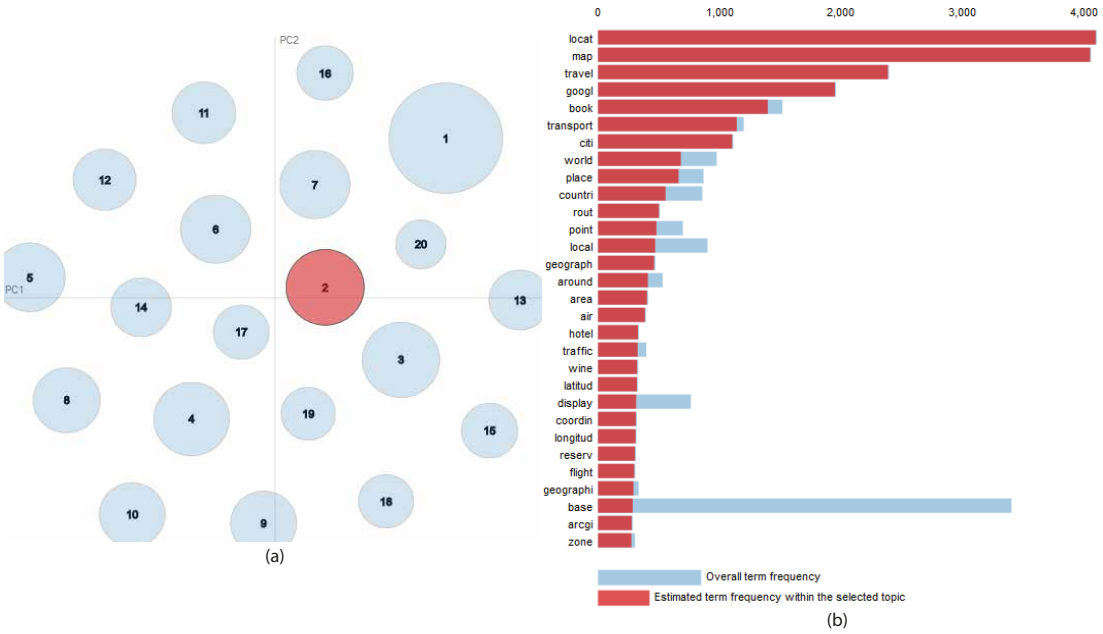



Figure 5.13: Intertopic distance map of 20 topics (a), and the topmost relevant terms for a topic (b)

Furthermore, the variance among the resulting document topic distributions can be considered in tuning the model hyperparameters. In the dataset, the higher values of $\alpha$ return better coherence scores. However, the variance of the documents' topic distributions becomes too small and it becomes hard to distinguish differences between documents. The higher values for $\alpha$ map each document into more topics, making the documents appear very similar. Since the purpose is to compare the Web services through their documents, the differences between the corresponding topic distributions should be emphasised. Besides, the average length of the documents is around 37 words, so it would be intuitive to assume that a document defines only one to a couple of topics. Hence, each document is mapped to a few but highly relevant topics rather than to a mixture of most of the topics, that is, we set $\alpha$ = $0.1$. In both coherence score and visual evaluation, setting $alpha$ = $0.1$, $eta$ = $0.5$, and the number of topics at 20 provides better topic clustering for the dataset.

The functionality relevance score between a mashup $u$ and API $i$ is defined as the similarity of the topic distribution of $u$: $\theta_u$, and the topic distribution of $i$: $\theta_i$. The comparison of the topic distributions uses the Jensen-Shannon divergence method:

$$JSD(\theta_u \| \theta_i) = 0.5 * D(\theta_u \| M) + 0.5 * D(\theta_i \| M) \tag{5.14}$$

where $M = 0.5 * (\theta_u + \theta_i)$ and $D(.\|.)$ is the Kullback-Leibler divergence. The Jensen-Shannon divergence is a positive definite measure of the difference between the two probability distributions where, $0 \leq JSD(\theta_u \| \theta_i) \leq 1$, $JSD(\theta_u \| \theta_i) = 0$ if and only if $\theta_u = \theta_i$. Hence, the functionality relevance score of $u$ and $i$ is defined as:

$$s(u, i) = 1 - JSD(\theta_u \| \theta_i) \tag{5.15}$$

### 5.2.3 Geographical and functionality aware recommendation model

Functional relevance has been a significant contextual factor in API recommendation, but considering geographical relevance is also important. It is because geographical locations can influence the QoS that can largely determine the effectiveness of API use. Augmenting functional relevance with geographic information expands the comprehensiveness of the contextual factor. Hence, the geographical relevance score is integrated with the functionality relevance score, to derive a contextual preference score $\rho_{ui}$ for a mashup $u$ that invokes API $i$:

$$\rho_{ui} = \omega \cdot g(u,i) + (1-\omega) \cdot s(u,i) \tag{5.16}$$

where $\omega$ is a weighting parameter to control the proportion of the two components. The sensitivity of the proposed model to the setting of $\omega$ is reported in the experiments.

The model integrates the geographical and functionality relevance of mashup-API invocations into the probabilistic matrix factorisation (Salakhutdinov & Mnih, 2007). Figure 5.14 shows an overview of the integration. Suppose there are $m$ number of mashups and $n$ number of APIs, and the invocation data are represented by a matrix $R \in \mathbb{R}^{m \times n}$. Each matrix element $r_{ui}$ represents the invocation of API $i$ in mashup $u$ (i.e., $r_{ui} = 1$ if mashup $u$ invokes API $i$). The objective is to find the mashup and API latent features (i.e., $X \in \mathbb{R}^{d \times m}$ and $Y \in \mathbb{R}^{d \times n}$, with $x_u$ and $y_i$ representing mashup-specific and API-specific latent feature vectors respectively), to estimate the preference degrees of the invocation data in $R$ (i.e., $\hat{r}_{ui} = x_u^T y_i$).

The conditional distribution of the observed invocation data is defined as:

$$p(R|X,Y,\rho,\sigma^2) = \prod_{u=1}^{m} \prod_{i=1}^{n} \left[ \mathcal{N}(r_{ui} \mid f(x_u, y_i, \rho_{ui}), \sigma^2) \right]^{I_{ui}} \tag{5.17}$$

where $\mathcal{N}(. \mid \mu, \sigma^2)$ is the probability density function of the Gaussian distribution

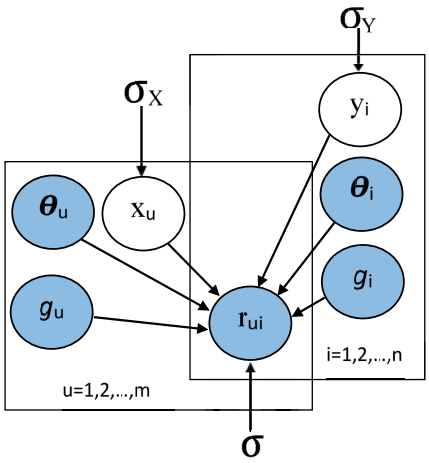



Figure 5.14: Graphical model of geographic-aware PMF

with mean $\mu$ and variance $\sigma^2$; and $I_{ui}$ is the indicator function that is equal to 1 when mashup $u$ has invoked API $i$, 0 otherwise. The function $f(x_u, y_i, \rho_{ui})$ is the approximate preference degree given to $r_{ui}$, and is used as the mean of the Gaussian distribution.

The function $f(x_u, y_i, \rho_{ui})$ is defined as:

$$f(x_u, y_i, \rho_{ui}) = x_u^T y_i \cdot \rho_{ui} \tag{5.18}$$

where computing the inner product of the latent feature vectors $x_u$ and $y_i$ of mashup $u$ and API $i$ is weighted by the contextual preference score $\rho_{ui}$. The weighting parameter denotes the impact of both geographical and functional relevance to the mashup-API invocation.

For the generative models of both mashup and API latent feature vectors, the conventional zero-mean spherical Gaussian priors are placed:

$$\begin{aligned} p(X|\sigma_X^2) &= \prod_{u=1}^{m} \mathcal{N}(X_u \mid 0, \sigma_X^2 I) \\ p(Y|\sigma_Y^2) &= \prod_{i=1}^{n} \mathcal{N}(Y_i \mid 0, \sigma_Y^2 I) \end{aligned} \tag{5.19}$$

Using Bayesian inference, the posterior distribution of Equation 5.17 is defined as:

$$
\begin{aligned}
&p(X, Y \mid R, \sigma^2, \rho, \sigma_X^2, \sigma_Y^2) \approx p(R \mid X, Y, \sigma^2, \rho, \sigma_X^2, \sigma_Y^2) p(X \mid \sigma_X^2) p(Y \mid \sigma_Y^2) \\
&= \prod_{u=1}^{m} \prod_{i=1}^{n} \left[ \mathcal{N}(r_{ui} \mid f(x_u, y_i, \rho_{ui}), \sigma^2) \right]^{I_{ui}} \\
&\times \prod_{u=1}^{m} \mathcal{N}(X_u \mid 0, \sigma_X^2 I) \times \prod_{i=1}^{n} \mathcal{N}(Y_i \mid 0, \sigma_Y^2 I)
\end{aligned}
\tag{5.20}
$$

The log of the posterior distribution over the mashup and API features is:

$$
\begin{aligned}
&\ln p(X, Y \mid R, \sigma^2, \rho, \sigma_X^2, \sigma_Y^2) \\
&= -\frac{1}{2\sigma^2} \sum_{u=1}^{m} \sum_{i=1}^{n} I_{ui} (r_{ui} - f(x_u, y_i, \rho_{ui}))^2 - \frac{1}{2\sigma_X^2} \sum_{u=1}^{m} X_u^T X_u - \frac{1}{2\sigma_Y^2} \sum_{i=1}^{n} Y_i^T Y_i \\
&- \frac{1}{2} \left[ \left( \sum_{u=1}^{m} \sum_{i=1}^{n} I_{ui} \right) \ln \sigma^2 + (m \cdot d \ln \sigma_X^2) + (n \cdot d \ln \sigma_Y^2) \right] + C
\end{aligned}
\tag{5.21}
$$

where $d$ is the latent factor dimension, and $C$ is a constant independent of the parameters. Maximising the log posterior with fixed hyperparameters is equivalent to minimising the sum-of-squared-errors objective function with quadratic regularisation terms (Salakhutdinov & Mnih, 2007). Hence, Equation 5.21 is reformulated into an objective loss function:

$$
\mathcal{L} = \sum_{u=1}^{m} \sum_{i=1}^{n} \frac{I_{ui}}{2} (r_{ui} - \rho_{ui} \cdot x_u^T y_i)^2 + \frac{\lambda_X}{2} \sum_{u=1}^{m} \|X_u\|_{Fro}^2 + \frac{\lambda_Y}{2} \sum_{i=1}^{n} \|Y_i\|_{Fro}^2 \tag{5.22}
$$

where $\lambda_X$ is $\sigma^2/\sigma_X^2$, $\lambda_Y$ is $\sigma^2/\sigma_Y^2$, and $\|.\|_{Fro}^2$ is the Frobenius norm. Learning algorithms such as the stochastic gradient descent (Koren et al., 2009) or alternating least squares (Hu et al., 2008) are utilised to arrive at the local minimum with respect to $X$ and $Y$:

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial x_u} &= -\sum_{i=1}^{n} I_{ui}(r_{ui} - \rho_{ui} \cdot x_u^T y_i) \cdot \rho_{ui} y_i + \lambda_X x_u \\ \frac{\partial \mathcal{L}}{\partial y_i} &= -\sum_{u=1}^{m} I_{ui}(r_{ui} - \rho_{ui} \cdot x_u^T y_i) \cdot \rho_{ui} x_u + \lambda_Y y_i\end{aligned} \tag{5.23}$$

With the optimised $X$ and $Y$, the model can finally predict the preference degrees for invoking an API $i$ in mashup $u$:

$$\hat{r}_{ui} \approx \mathbb{E}[r_{ui} \mid x_u, y_i] = x_u^T y_i = \rho_{ui} \cdot x_u^T y_i \tag{5.24}$$

Alternatively, the loss function can be written as:

$$\mathcal{L} = \sum_{u=1}^{m} \sum_{i=1}^{n} \frac{1}{2} \rho_{ui}^2 (\frac{r_{ui}}{\rho_{ui}} - x_u^T y_i)^2 + \frac{\lambda_X}{2} \sum_{u=1}^{m} \|X_u\|_{Fro}^2 + \frac{\lambda_Y}{2} \sum_{i=1}^{n} \|Y_i\|_{Fro}^2 \tag{5.25}$$

where $r_{ui}$ is replaced by $\frac{r_{ui}}{\rho_{ui}}$ and each squared error $(\frac{r_{ui}}{\rho_{ui}} - x_u^T y_i)^2$ is weighted by $\rho_{ui}^2$. It is worth noting that the invocation data (i.e., the original matrix $R$) is modified by the contextual preference score $\rho_{ui}$. Thus, there will be different preference levels among the APIs invoked by the mashups. A larger $\rho_{ui}$ indicates a stronger preference for an API $i$ in the mashup $u$. The use of the preference score $\rho_{ui}$ as a weighting parameter is likened to the idea presented by Belkin and Niyogi (2001).

## 5.3 Experiments

This section reports the empirical performance of the model based on the proposed method, on the ProgrammableWeb dataset. The experiments aim to answer the following questions:

Q1. How does the proposed model perform against the selected baseline models using the experimental dataset?

Q2. How do the contextual factors (i.e., geographical and functional relevance) improve recommendation performance? Does considering geographical relevance improve the performance (i.e., precision) of the model?

Q3. How do the model parameters impact the recommendation performance?

### 5.3.1 Evaluation metrics

The evaluation uses the two popular metrics for measuring the performance of classification and ranking recommenders (Schröder, Thiele & Lehner, 2011): *discounted cumulative gain* and *average precision*. Unlike the prediction accuracy metrics that measure how close the estimated ratings are to the actual ones (e.g., RMSE and MAE), the chosen metrics regard *i*) the degree of correctness of recommended APIs based on relevance, and *ii*) the ability to estimate the correct order of recommended APIs based on the degree of preferences.

1. Normalised Discounted Cumulative Gain (nDCG@K): the Discounted Cumulative Gain (DCG) measures the quality of recommendation based on the ranking of relevant APIs in the recommendation list. For each relevant API, it assumes a usefulness, or *gain*, that is accumulated from the top of the recommendation list to the bottom. The gain is discounted at every lower rank, penalising relevant APIs at the lower ranks. The DCG@K considers the subset of a recommendation list from rank 1 to K and is defined as:

$$DCG@K = \sum_{i=1}^{K} \frac{rel_i}{log_2(i+1)} \tag{5.26}$$

where $rel_i$ is the predicted score of the relevant API at rank $i$. To maintain consistency in the DCG of various recommendation lists, particularly the lists that vary in length, a

normalised DCG (nDCG) is formulated:

$$nDCG@K = fracDCG@KIDCG@K \tag{5.27}$$

where IDCG@K is the ideal DCG, i.e., the maximum possible DCG@K of all relevant APIs within the rank 1 to K sorted by their predicted score.

$$IDCG@K = \sum_{i=1}^{|REL|} \frac{rel_i}{log_2(i+1)} \tag{5.28}$$

where $\mid REL \mid$ is the list of sorted relevant APIs. The nDCG@K score ranges from 0 to 1.

This metric is appropriate for the case study because of its ability to measure how well predictions are ranked rather than the actual value of the prediction. It is worth mentioning that the approach is concerned with providing a recommendation for where the highly relevant APIs should appear at the top of recommendation lists.

The evaluation strategy for nDCG is based on the approach by Rendle, Freudenthaler, Gantner and Schmidt-Thieme (2009). A certain percentage of the mashup-API interactions is randomly hidden to create the training set. Figure 5.15 illustrates this approach. The masked interactions in the training set (those in *) are considered zero entries, i.e., an $r_{ui}$ value is altered from 1 to 0, meaning it is as if the mashup $u$ did not invoke API $i$. The testing set is the same as the original data.

After training a model and generating predictions, the nDCG of each mashup that has an altered interaction data is computed by examining the corresponding rows in both the testing and prediction sets. Those entries that are 1 in the training set are removed from both rows. The purpose is to know from the testing set if the most recommended APIs up to rank $K$ are the ones actually invoked by the mashup. The more frequent a mashup ends up invoking those most recommended APIs, the higher the nDCG score.

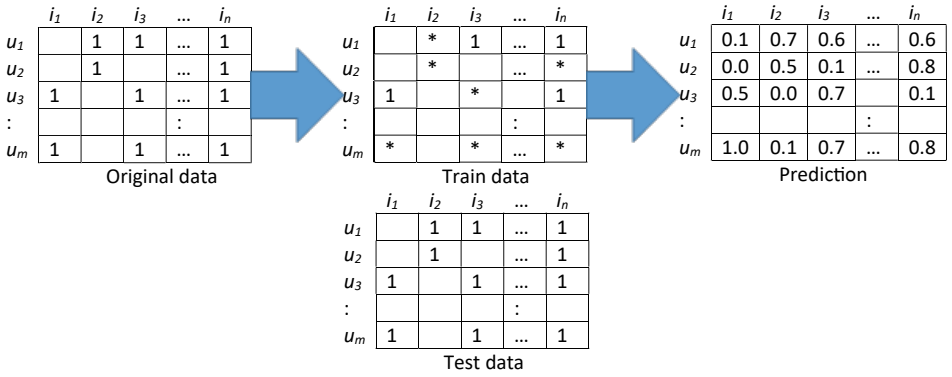



Figure 5.15: Creating the training and testing data for nDCG

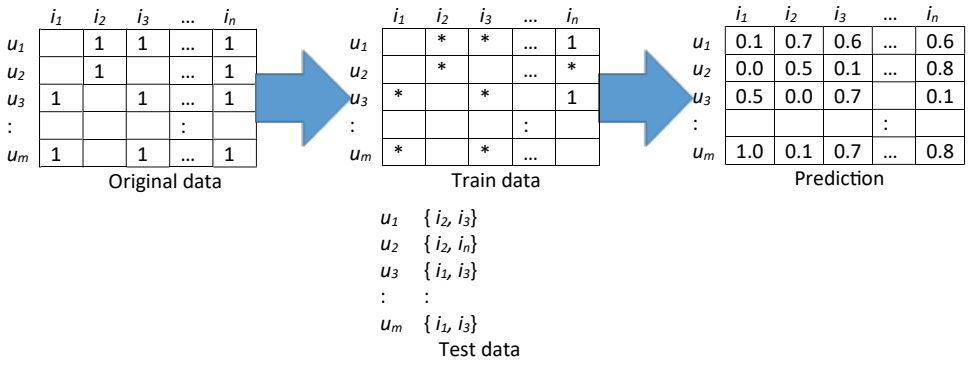



Figure 5.16: Creating the training and testing data for MAP

The mean nDCG@K of all the evaluated mashups is computed by:

$$nDCG@K_{mean} = \frac{1}{\mid U \mid} \sum_{u \in U} nDCG@K_u \tag{5.29}$$

where $U$ is the set of mashups with altered interaction data.

2. Mean Average Precision (MAP@K): *precision P* (a.k.a. *true positive accuracy* [Schröder et al., 2011]) is the ratio between the recommended APIs that are relevant (those actually invoked by the mashup) and the total number of recommended APIs, i.e., $P = \frac{TP}{TP+FP}$. Precision measures the probability that a recommended API is the one preferred for the mashup. $P@K$ is a precision calculated by considering only a subset of the recommendations from rank 1 through $K$. Given a mashup $u$, the *Average Precision* $AP@K_u$ calculates the precision of each relevant API, considering its position in the ranked list of $K$ recommendations:

$$AP@K_u = \frac{1}{m} \sum_{n=1}^{K} (P(n) \times rel(n)) \tag{5.30}$$

where $P(n)$ is the precision at cut-off $n$ in the list, and $rel(n)$ is a function equalling to 1 when the API at rank $n$ is relevant, zero otherwise. The $AP$ metric recognises not just every relevant recommendation, but also rewards front-loading the recommendation with the most-likely correct APIs, i.e., the relevant APIs at the top of the list. Then, the arithmetic mean of the $AP@K$ of all mashups is called the *Mean Average Precision*:

$$MAP@K = \frac{1}{\mid U \mid} \sum_{u=1}^{|U|} AP@K_u \tag{5.31}$$

For the MAP metric, Figure 5.16 illustrates the approach of creating the training and testing set which also follows the masking of data values. However, rather than hiding a random percentage of mashup-API interactions from the overall data, the approach

hides an $n$ equal number of interactions from each mashup. Hence, the testing set contains all the mashups, each with $n$ invoked APIs (i.e., the APIs associated with the hidden interactions). Then, the $MAP$ of the top $K$ recommended APIs is evaluated on the hidden ones.

### 5.3.2 Comparison with baseline approaches

The recommendation performance of the proposed approach is compared to several baselines.

- Neighbour-based Collaborative Filtering (NCF [Zheng, Ma, Lyu & King, 2009]). This non-MF method computes the similarity of APIs based on their invocation in the mashups, and is typically called the item-based CF. The predictions are based on discovering similarly invoked APIs and then calculating a weighted combination of the preferences.
- Regularised Matrix Factorisation (RMF [Koren et al., 2009]). This method is the standard matrix factorisation that adds regularisation terms to avoid overfitting. It assumes all missing data as negatives (i.e., an unknown value in the mashup-API matrix is assigned to zero).
- Weighted Regularised Matrix Factorisation (wRMF [Pan et al., 2008; Srebro & Jaakkola, 2003]). This method extends RMF by incorporating a global weighting factor to the invocation data (i.e., 1 is given to observed examples and 0 to the missing ones), to specify the relative preference to the optimisation.
- Implicit-tailored Matrix Factorisation (IMF [Hu et al., 2008]). This implicit-based matrix factorisation is adopted, in which functionality relevance is integrated to the invocation data.
- Probabilistic Matrix Factorisation (PMF [Salakhutdinov & Mnih, 2007]). This is the standard probabilistic matrix factorisation approach, in which only the

invocation data is used to perform collaborative filtering.

- Func-PMF. This is a variant of the proposed approach, which extends the implicit invocation data with functionality relevance.

Table 5.2: Comparison of input data and regularisation parameter

| | The usage of | | | |
|---|---|---|---|---|
| | Invocation data | Textual description | Geographic info | Regularisation param $(\lambda_X, \lambda_Y)$ |
| NCF | ✓ | – | – | – |
| RMF | ✓ | – | – | 300 |
| wRMF | ✓ | – | – | 1.0 |
| IMF | ✓ | ✓ | – | 300 |
| PMF | ✓ | – | – | 0.1 |
| Func-PMF | ✓ | ✓ | – | 0.001 |
| The proposed model | ✓ | ✓ | ✓ | 0.001 |

The latent dimension is set to 20, and both $X$ and $Y$ are initialised to some normally distributed random values. Each method is evaluated on its best performing regularisation parameter from among the following values $\lambda(\lambda_X, \lambda_Y) = \{0.001, 0.01, 0.1, 0.5, 1.0, 100, 300, 500, 1{,}000\}$. Table 5.2 shows the respective regularisation parameters of each method, as well as the input data being considered.

Table 5.3: The nDCG scores of the different methods

| | nDCG@1 | | | | | nDCG@3 | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 10% | 20% | 30% | 40% | 50% | 10% | 20% | 30% | 40% | 50% |
| NCF | 0.0001 | 0.0006 | 0.0004 | 0.0002 | 0.0001 | 0.0003 | 0.0003 | 0.0006 | 0.0005 | 0.0001 |
| RMF | 0.0884 | 0.1073 | 0.0938 | 0.0947 | 0.0861 | 0.1323 | 0.1421 | 0.1292 | 0.1220 | 0.1090 |
| wRMF | 0.1089 | 0.1078 | 0.1116 | 0.1034 | 0.1001 | 0.1493 | 0.1434 | 0.1341 | 0.1243 | 0.1103 |
| IMF | 0.1703 | 0.1568 | 0.1475 | 0.1356 | 0.1268 | 0.2188 | 0.1921 | 0.1743 | 0.1521 | 0.1345 |
| PMF | 0.2251 | 0.2486 | 0.2693 | 0.2855 | 0.2964 | 0.2577 | 0.3044 | 0.3144 | 0.3265 | 0.3299 |
| Func-PMF | 0.2364 | 0.2503 | 0.2689 | 0.2877 | 0.2994 | 0.2884 | 0.3053 | 0.3164 | 0.3279 | 0.3325 |
| The proposed model | **0.2468** | **0.2784** | **0.2799** | **0.2879** | **0.2999** | **0.3003** | **0.3261** | **0.3254** | **0.3292** | **0.3357** |

| | nDCG@5 | | | | | nDCG@10 | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 10% | 20% | 30% | 40% | 50% | 10% | 20% | 30% | 40% | 50% |
| NCF | 0.0001 | 0.001 | 0.0011 | 0.0009 | 0.0005 | 0.0005 | 0.0013 | 0.0023 | 0.0011 | 0.0007 |
| RMF | 0.1551 | 0.1631 | 0.1435 | 0.1352 | 0.1199 | 0.1654 | 0.1715 | 0.1545 | 0.1450 | 0.1267 |
| wRMF | 0.1728 | 0.1639 | 0.1516 | 0.1512 | 0.1502 | 0.1892 | 0.1811 | 0.1665 | 0.1553 | 0.1524 |
| IMF | 0.2332 | 0.2104 | 0.189 | 0.1663 | 0.1563 | 0.2509 | 0.2249 | 0.2025 | 0.1830 | 0.1565 |
| PMF | 0.2983 | 0.3242 | 0.3414 | 0.3526 | 0.3598 | 0.3328 | 0.3615 | 0.3768 | 0.3886 | 0.3962 |
| Func-PMF | 0.3263 | 0.3346 | 0.3458 | 0.3564 | **0.3615** | 0.3564 | 0.3665 | 0.3809 | 0.3906 | 0.3975 |
| The proposed model | **0.3364** | **0.3862** | **0.3618** | **0.3603** | 0.3577 | **0.3613** | **0.3819** | **0.3977** | **0.3920** | **0.3989** |

To evaluate a method's performance using nDCG, there are five (5) settings of

train-test split for the dataset; each setting randomly masks a certain percentage of the mashup-API interactions (i.e., 10%, 20%, 30%, 40%, 50% data masking). In each setting, five (5) instances of the data are created (i.e., through seeding), so that a different set of data is masked per instance. Table 5.3 summarises the obtained nDCG scores at different values of $K = \{1, 3, 5, 10\}$, where a tabulated score is the average nDCG score from the five data instances of each setting. Figure 5.17 plots the nDCG@K scores of the top-performing methods.

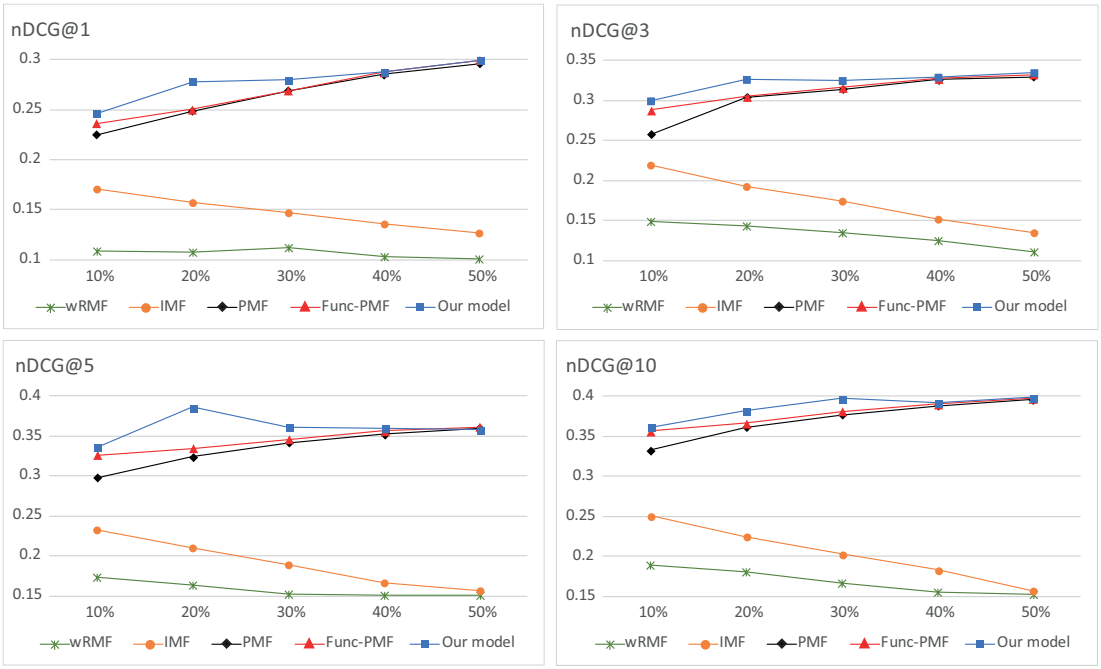



Figure 5.17: NDCG@K scores comparison of the top performing methods

The proposed model outperforms all the four top performing baselines based on the nDCG metric, particularly at 10%, 20%, and 30% dataset cut-offs. There are two contrasting patterns among the obtained nDCG scores. For wRMF and IMF, nDCG@K gradually decreases as the size of the masked interaction data increases. Such behaviour is normally expected since increasing the percentage of masked data results in less available mashup-API interaction data to train a model. Hence, among the five data settings, it is assumed that the worst performance of a method happens when

50% of the data are masked, and the best performance happens when only 10% are masked. Interestingly, the PMF-based models, including the proposed model, portray an increasing score as the size of masked data increases. The PMF-based models perform less with lower percentage data masking (e.g., they produce significantly better nDCG scores at 20% masking than at 10%). This behaviour is also possible, since a masking percentage denotes the train-test dataset cut-off, as discussed in Section 5.3.1. A lower percentage of masked data means less actual data in the testing set. Since nDCG mainly looks for predictions that exist in the actual data, the chances of true positives can relatively increase as the number of actual data increases. The spike at 20% masking indicates that the model performs better at that dataset setting.

Table 5.4: The dataset structure and statistics for MAP

| Setting | Mashup-API invocation (dataset) | No. of API in the testing set ($t$) | Mashup-API invocation (training set) |
|---|---|---|---|
| (a) $s > 4$ | 2,460 | 4 | 1,080 |
| | | 3 | 1,425 |
| | | 2 | 1,770 |
| | | 1 | 2,115 |
| (b) $s > 3$ | 3,436 | 3 | 1,669 |
| | | 2 | 2,258 |
| | | 1 | 2,847 |
| (c) $s > 2$ | 4,867 | 2 | 2,735 |
| | | 1 | 3,801 |
| (d) $s > 1$ | 7,459 | 1 | 5,097 |

The evaluation using the MAP metric creates four (4) settings of training-test data: $s > 4$, $s > 3$, $s > 2$, and $s > 1$, where $s$ is the number of APIs in a mashup. Table 5.4 summarises these dataset arrangements. For example, the setting $s > 3$ considers only the mashup rows invoking more than three APIs, and in this case, the dataset size is reduced to 3,436. Each setting has its test cases with varying sizes of the training-test split. For example, the setting $s > 3$ has three test cases: $t = \{3, 2, 1\}$, where each $t$ corresponds to the number of API in the testing set of each mashup. The training-test split varies with $t$. For example, a test case $t = 2$ for the setting $s > 3$ randomly picks

two API invocations per mashup as test data, leaving a training-set sized 2,258. The MAP@$K$ ($1 \leq K \leq 10$) of a method in each test case is derived.

Table 5.5: MAP scores of the different methods

| | MAP@$K$: $s > 1$, $t = 1$ | | | | | MAP@$K$: $s > 2$, $t = 2$ | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | $K$=1 | $K$=3 | $K$=5 | $K$=7 | $K$=10 | $K$=1 | $K$=3 | $K$=5 | $K$=7 | $K$=10 |
| NCF | 0.0004 | 0.0004 | 0.0004 | 0.0004 | 0.0005 | 0.0001 | 0.0002 | 0.0002 | 0.0003 | 0.0004 |
| RMF | 0.0313 | 0.0486 | 0.0556 | 0.0583 | 0.0606 | 0.0759 | 0.0958 | 0.1020 | 0.1056 | 0.1086 |
| wRMF | 0.1257 | 0.1543 | 0.1621 | 0.1667 | 0.1699 | 0.1519 | 0.1278 | 0.1429 | 0.1521 | 0.1582 |
| IMF | 0.1531 | 0.1601 | 0.1678 | 0.1812 | 0.2157 | 0.1589 | 0.1395 | 0.1598 | 0.1653 | 0.1682 |
| PMF | 0.1042 | 0.1321 | 0.1570 | 0.1662 | 0.1740 | 0.1228 | 0.1145 | 0.1405 | 0.1526 | 0.1615 |
| Func-PMF | 0.1574 | 0.1999 | 0.2108 | 0.2139 | 0.2197 | 0.2082 | 0.1552 | 0.1617 | 0.1695 | 0.1713 |
| The proposed model | **0.1651** | **0.2085** | **0.2184** | **0.2221** | **0.2326** | **0.2148** | **0.1729** | **0.1971** | **0.1990** | **0.2196** |

| | MAP@$K$: $s > 3$, $t = 2$ | | | | | MAP@$K$: $s > 4$, $t = 2$ | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | $K$=1 | $K$=3 | $K$=5 | $K$=7 | $K$=10 | $K$=1 | $K$=3 | $K$=5 | $K$=7 | $K$=10 |
| NCF | 0.0237 | 0.0096 | 0.0096 | 0.0096 | 0.0096 | 0.0086 | 0.0045 | 0.0039 | 0.0039 | 0.004 |
| RMF | 0.0865 | 0.1004 | 0.1102 | 0.1158 | 0.119 | 0.0463 | 0.0724 | 0.0853 | 0.0892 | 0.0921 |
| wRMF | 0.1409 | 0.1129 | 0.1322 | 0.1417 | 0.1496 | 0.1507 | 0.1096 | 0.1143 | 0.1163 | 0.1172 |
| IMF | 0.1816 | 0.1279 | 0.1523 | 0.1634 | 0.1662 | 0.1799 | 0.1299 | 0.1406 | 0.1481 | 0.1601 |
| PMF | 0.1518 | 0.1087 | 0.1351 | 0.1473 | 0.156 | 0.1507 | 0.1084 | 0.1197 | 0.1256 | 0.1362 |
| Func-PMF | 0.1865 | 0.144 | 0.1655 | 0.1659 | 0.174 | 0.2071 | 0.1388 | 0.1541 | 0.1646 | 0.1712 |
| The proposed model | **0.2156** | **0.1832** | **0.1984** | **0.2043** | **0.2102** | **0.2463** | **0.1437** | **0.1583** | **0.1724** | **0.1902** |

Table 5.5 presents the obtained MAP scores at different values of $K = \{1, 3, 5, 7, 10\}$, in the four dataset settings. Each setting shows the MAP scores from a selected test case (e.g., the left upper quadrant of Table 5.5 corresponds to the setting $s > 1$ on test case $t = 1$). Figure 5.18 illustrates the comparison of MAP@$K$ scores of the proposed method with the top-performing baseline methods. In the individual test cases, there is a decrease in the MAP score from $K = 1$ to $K = t$, then a gradual increase in the MAP score starts after $K = t$. Such an increasing score after $K = t$ is expected, since the probability of encountering relevant APIs increases as the list of recommendations grows (i.e., MAP does not penalise wrong recommendations). The results show that the proposed method outperforms the baseline methods in all settings in terms of precision in the top-10 recommendations. Since MAP recognises positioning relevant APIs at the top of recommendations, the results show that the method is more precise for identifying and ranking relevant APIs. Hence, the method performs more effectively in recommending the few most-likely correct APIs.

In both evaluation metrics, NCF obtains the lowest performance that is far below

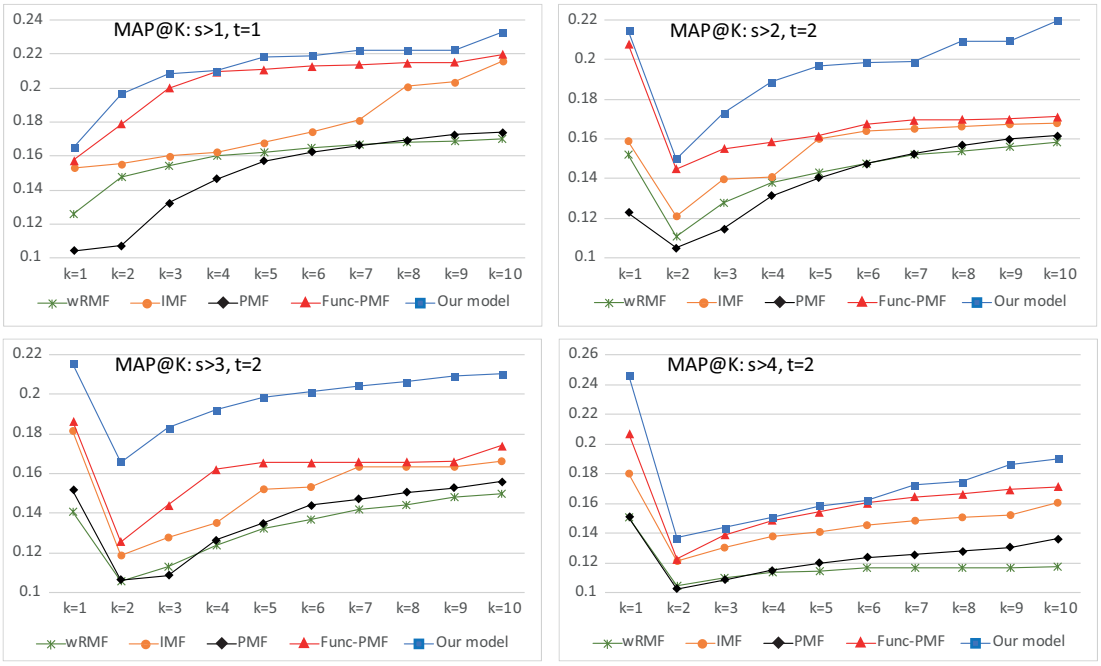



Figure 5.18: MAP@K scores comparison

the other baselines. This is expected, since it uses the API-based collaborative filtering that depends on the API-API similarity, i.e., similarity based on API invocation in the mashups. The majority of the APIs are rarely invoked, so the existing highly-sparse data are not sufficient to generate some decent similarity values. With the matrix factorisation techniques, there is a significant improvement in the scores. The main reason for this, is the learned latent factors underlying the existing mashup-API interactions. Discovering these latent factors can partially relieve such a sparsity-constrained recommendation problem (Bobadilla et al., 2013).

### 5.3.3 Impact of the dispersion factor $\delta$

In Section 5.2.1, the geographical similarity function that maps a distance into a similarity value is presented. It is interesting to examine the impact of the dispersion factor $\delta$ to the geographical relevance scores. Table 5.6 shows the variance among the derived geographical relevance scores at different values of $d$ and $k$. At every $k$ value,

the variance among the relevance scores similarly increases as the value set for $\delta$ is increased. This observation is expected, since a low value for $\delta$ would quickly map relatively high distance values to 0, and those high distance values would yield similarity values close to 0 with dense intervals, such that their differences would become hard to distinguish. A high value for $\delta$, in contrast, would hardly map high distance values to 0.

Table 5.6: Variance of the geographical relevance scores at different values of $\delta$ and $k$

| Neighbor distance k | Dispersion factor $\delta$ | | | | |
|---|---|---|---|---|---|
| | $\delta$=1,000 | $\delta$=2,500 | $\delta$=5,000 | $\delta$=10,000 | $\delta$=15,000 |
| $k$=1 | 0.0162 | 0.0320 | 0.0524 | 0.0741 | 0.0859 |
| $k$=10 | 0.0163 | 0.0321 | 0.0522 | 0.0733 | 0.0846 |
| $k$=50 | 0.0166 | 0.0328 | 0.0529 | 0.0729 | 0.0834 |
| $k$=100 | 0.0166 | 0.0328 | 0.0528 | 0.0724 | 0.0826 |
| $k$=500 | 0.0167 | 0.0331 | 0.0527 | 0.0709 | 0.0803 |
| $k$=1,000 | 0.0156 | 0.0324 | 0.0531 | 0.0722 | 0.0818 |

The impact of $\delta$ on the performance of the model is also examined. At each value of $\delta$, the NDCG@5 and MAP@5 of the model at different values of $k = \{1, 10, 50\}$ are obtained. Table 5.7 shows the average NDCG@5 and MAP@5 derived from these three settings for $k$. Each row of Table 5.7 represents the model performance at a particular weighting ($\omega$) for the geographical relevance. Deriving NDCG uses the train-test dataset setting that masks 20% of the interactions. Deriving MAP uses the $t = 1$ instance of dataset setting $s > 2$. The scores in bold are the highest ones at that particular weighting parameter. Most of the highest scores are obtained at $\delta = 5,000$ and $\delta = 10,000$. It is observed that when the weight of the geographical relevance is above 50% (i.e., geographical relevance has a significant contribution to the prediction of preference degrees), most of the best performance scores are on $\delta = 5,000$. Based on these results, $\delta = 5,000$ is used as a default setting in the experiments.

Table 5.7: Impact of different $\delta$ values to the model's performance

| Weighting parameter | Dispersion factor $\delta$ | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | $\delta$=1,000 | | $\delta$=2,500 | | $\delta$=5,000 | | $\delta$=10,000 | | $\delta$=15,000 | |
| $\omega$ | ndcg@5 | map@5 | ndcg@5 | map@5 | ndcg@5 | map@5 | ndcg@5 | map@5 | ndcg@5 | map@5 |
| $\omega$=0.1 | 0.3481 | **0.1989** | 0.3478 | 0.1494 | **0.3494** | 0.1534 | 0.3478 | 0.1670 | 0.3472 | 0.1431 |
| $\omega$=0.2 | 0.3479 | 0.1578 | 0.3478 | 0.1778 | 0.3485 | **0.2017** | **0.3493** | 0.1957 | 0.3485 | 0.1608 |
| $\omega$=0.3 | 0.3470 | 0.1990 | 0.3478 | 0.1842 | 0.3475 | 0.1908 | **0.3486** | 0.2034 | 0.3484 | **0.2074** |
| $\omega$=0.4 | 0.3466 | 0.1989 | 0.3473 | 0.1995 | 0.3477 | 0.1879 | 0.3478 | **0.1996** | **0.3481** | 0.1780 |
| $\omega$=0.5 | 0.3466 | 0.1497 | 0.3473 | 0.1783 | **0.3478** | 0.1901 | 0.3473 | 0.1564 | 0.3477 | **0.1981** |
| $\omega$=0.6 | 0.3482 | 0.1718 | 0.3474 | 0.1637 | **0.3491** | **0.2112** | 0.3474 | 0.2015 | 0.3473 | 0.2055 |
| $\omega$=0.7 | 0.3470 | 0.1818 | 0.3476 | 0.1633 | **0.3478** | 0.2020 | 0.3476 | **0.2100** | 0.3473 | 0.1929 |
| $\omega$=0.8 | 0.3453 | 0.1656 | 0.3432 | 0.1987 | 0.3483 | **0.2057** | **0.3484** | 0.1897 | 0.3479 | 0.1916 |
| $\omega$=0.9 | 0.3245 | 0.1986 | 0.3372 | 0.1932 | **0.3477** | **0.1998** | 0.3474 | 0.1718 | 0.3472 | 0.1936 |

### 5.3.4 Impact of the neighbours' distance $k$

The parameter $k$ of the geographic relevance model sets the distance from which neighbours are considered. Setting a too large value for $k$ results in too many neighbours and may introduce noise to the prediction data, since distant neighbours are less likely to have a similar operational context. A too-small value for $k$ constrains the neighbour size, which may lead to the model's disregarding of some relevant neighbours. This section examines the impact of $k$ to the performance of the model. At each value of $k = \{1, 10, 50, 100, 500, 1,000\}$, the NDCG@5 and MAP@5 of the model are obtained at different values of $\omega = \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$. Deriving NDCG uses the train-test dataset setting that masks 20% of the interactions. Deriving MAP uses the $t = 1$ instance of dataset setting $s > 2$.

The respective means of the obtained NDCG@5 and MAP@5 among the different values of $\omega$, are shown in Figure 5.19 (mean scores are shown at each $k$). It is observed that setting $k = 50$ yields better performance scores. Hence, $k = 50$ is used as the default setting in the experiments.

### 5.3.5 Impact of the weighting parameter $\omega$

The parameter $\omega$ determines the proportion of geographic relevance considered in the integrated model. When $\omega > 0.5$, the role of geographic relevance is given more

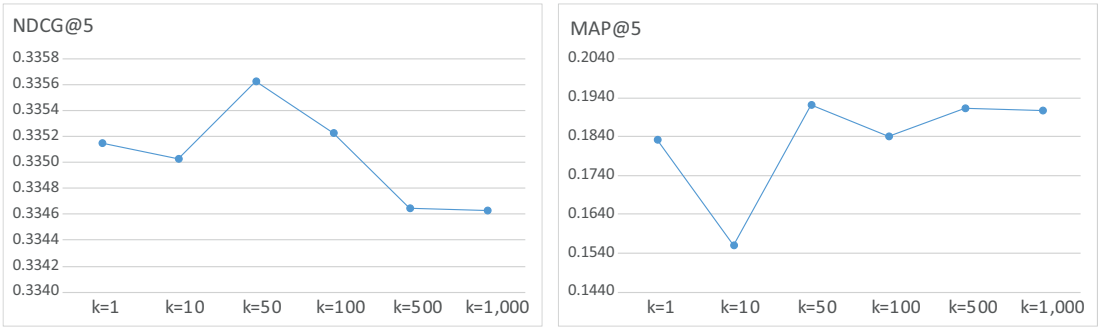



Figure 5.19: Impact of $k$ to the model's performance

significance than functionality relevance, in the prediction. An $\omega = 1.0$ disregards the role of functionality relevance. When $\omega < 0.5$, the role of geographic relevance is given less significance than functionality relevance. An $\omega = 0$ disregards the role of geographical relevance. This section examines the impact of $\omega$ to the model's performance. The NDCG@5 and MAP@5 of the model are derived at various settings for $\omega = \{0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$. Figure 5.20 shows the derived NDCG@5 scores from two dataset settings. In both dataset settings, the highest scores are obtained at $\omega = \{0.3 - 0.5\}$. Figure 5.20 similarly shows the MAP@5 scores from two dataset settings, where the highest scores are obtained at $\omega = \{0.4 - 0.6\}$.

Figure 5.20 shows that when $\omega < 0.5$ the scores are slightly higher than the scores when $\omega > 0.5$. This suggests that the functionality relevance has a greater impact on the prediction performance. Comparing the scores at $\omega = 0.0$ and $\omega = 1.0$ (i.e., the lowest scores are obtained at these settings of $\omega$), the scores at $\omega = 0.0$ are slightly better than those at $\omega = 1.0$. The implication is that a prediction that considers solely functionality relevance, provides better precision than one that considers only geographical relevance. However, the role of geographic relevance should not be ignored since better performance is obtained when $1 > \omega > 0$.

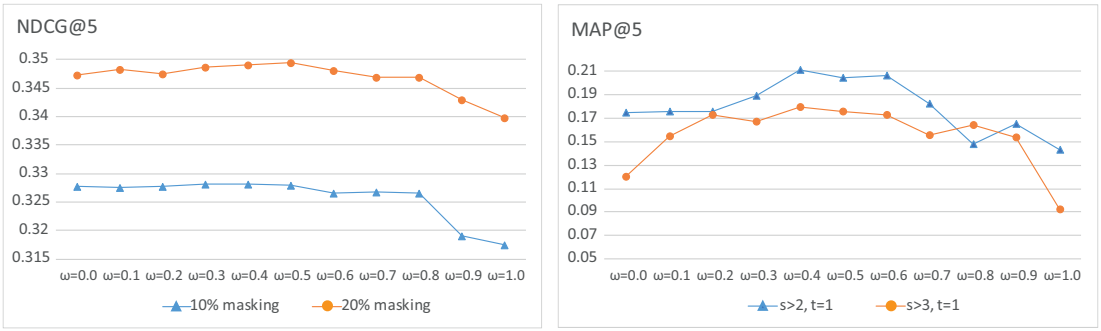



Figure 5.20: The impact of $\omega$ to the model's performance

### 5.3.6 Impact of the latent feature dimensionality $d$

This section examines the impact of the number of latent features to the performance of the model by varying the value of $d$ = {10,20,..., 200, 500, 1,000}. Figure 5.21(a) shows the derived nDCG@5 at different values of $d$, on a dataset with 10% of the data masked. The nDCG score looks stable from $d$ = 10 to $d$ = 60, then shows a noticeable increase at $d$ = 70 up to $d$ = 80. After reaching the highest nDCG score at $d$ = 80, the score gradually decreases as $d$ increases. Similarly, Figure 5.21(b) shows the derived MAP@5 at different values of $d$, using the dataset setting $t > 2$, $s = 1$. There is a relative increase in the MAP score, as $d$ increases from $d$ = 10 to $d$ = 80. However, beyond $d$ = 80, the scores dramatically fall as $d$ increases. Both metrics only plot the derived score from $d$ = 10 to $d$ = 140. Since the score on values $d > 140$ becomes significantly lower (e.g., at $d$ = 500), the values 0.099 and 0.098 for nDCG@5 and MAP@5 are obtained respectively.

The results on the varying dimensionality indicate two points. *First*, given the range of $d$ values from which stable or increasing scores are obtained, there are several choices for a $d$ that can provide superior performance (e.g., the values $d$ = {10 - 80} for both metrics). *Second*, the observed relative increase in the metric scores as $d$ increases, conforms to the intuitive assumption that more latent features will extract more relevant shared information. It is also intuitive to assume that having too few latent features

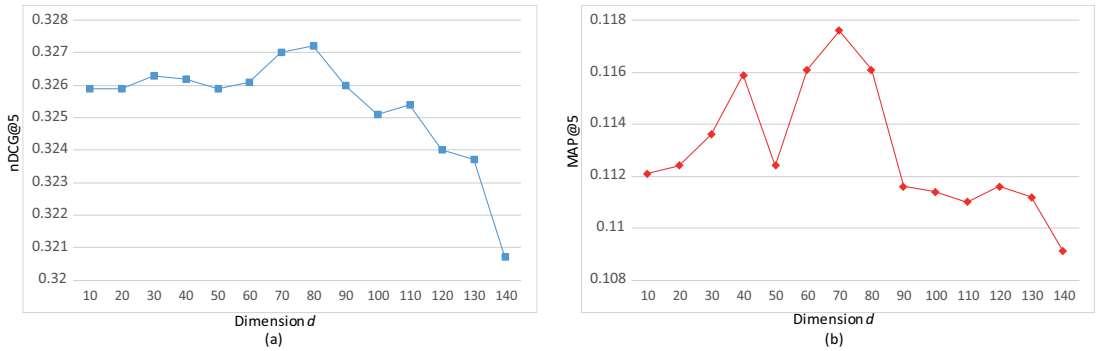



Figure 5.21: Impact of $d$ to the model's performance

(e.g., $d < 10$) will limit the ability of the model to extract relevant information, leading to poor performance. However, having an excessive number of latent features leads to overfitting (H. Wu et al., 2018; Yao et al., 2018), which degrades the performance of the model. As observed in Figure 5.21, the score decreases as $d$ goes beyond 80.

The experiments use $d = 20$, unless specified otherwise. Although utilising more latent features is theoretically beneficial in obtaining better performance (e.g., precision and accuracy) (H. Wu et al., 2018), a large dimensionality value incurs a higher computation cost. That is, the computational complexity of matrix factorisation is directly proportional to the latent feature dimensionality. Hence, the only desired factors are those that define the underlying relationships between mashups and APIs – the relationships that impact the invocation preference.

## 5.4 Chapter Summary

The explosion of reusable Web services (e.g., open APIs, open data sources, and cloud/IoT services), has provided a new opportunity for modern service-composition based applications development. However, this enormous growth of Web services increases the difficulty of selecting the best suitable Web services for a particular application. Hence, the design of an effective and efficient Web service recommendation,

primarily based on user feedback, has become a challenge. In the mashup-API recommendation scenario, the most available feedback is the implicit invocation data (i.e., the binary data indicating whether or not a mashup has invoked an API). Various efforts are being made to exploit potential impact factors, such as the invocation context, to augment the implicit invocation data, with the aim of improving service recommendation performance. One significant factor affecting the context of Web service invocations is geographical location, but it has received less attention in the implicit-based service recommendation. This chapter has presented a probabilistic matrix factorisation based recommendation approach, which considers geographic location information in the derivation of the preference degree underlying a mashup-API interaction. The geographic information, which is integrated with functional descriptions, complements the mashup-API invocation data input for the matrix factorisation model. The chapter has demonstrated the effectiveness of the approach by presenting the results of extensive experiments on a real dataset retrieved from ProgrammableWeb. The evaluation results have shown that augmenting the implicit data with geographical location information, in addition to functional descriptions, increases the precision of API recommendation for mashup services.

# Chapter 6

# Conclusion and Future Work

It is no longer sufficient for today's software systems to just correctly perform a fixed function for which they are designed. Modern contexts of software systems are now characterised by complex environments built out of diverse infrastructures, components, services, and other systems that are not directly controlled by the software developer. Such contexts are unstable. Besides, with the prevalence of mobility and ubiquity, contexts have become highly dynamic. Requirements inevitably change and the ever changing context has furthered requirements variability. Hence, adaptability has become necessary for software systems to respond appropriately to the changing contexts and requirements.

In order for service-based software systems to rise to the challenges of modern contexts and requirements variability, adaptability must be explicitly engineered into the development process. An existing adaptability conceptual framework is thoroughly discussed in this thesis (see Chapter 1 Section 1.2 and Chapter 2 Section 2.2). That framework describes the integration of adaptability processes with the service composition (SC) life cycle and provides an abstract of a holistic approach for adaptability that spans throughout all SC life cycle phases. Moreover, since adaptability is a complex task with extremely diverse concerns, the framework can significantly establish the basis

of organising and conceptualising adaptability solutions. Each specific solution that is intended for an adaptability concern can be mapped into the framework. Accordingly, this thesis is intended to contribute solutions to address specifically three adaptability concerns. Chapters 3, 4, and 5 of the thesis have presented in detail the proposed solutions to address those three concerns respectively that are within the two SC phases: requirements engineering & design, and construction.

The rest of this concluding chapter summarises the thesis contributions, key results, and topics for future work. The summary and discussions are presented by chapter.

## 6.1 Adaptability Metrics

Service composition has become a promising paradigm for fulfilling the demands of modern service-based distributed systems. The volatile contexts of these systems make adaptability a significant property to consider. Chapter 3 has proposed metrics to quantify the adaptability of Business Process Execution Language (BPEL-based) processes. The metrics consider two adaptability dimensions: the first is the structure variability that changes the execution behaviour of a process, a common adaptation mechanism among extended-BPEL frameworks; the second is the binding variability of a process to select a partner service. In the former and latter cases, the metrics respectively reward the availability of more variations in the process execution workflow and the presence of multiple alternatives to existing component services. Chapter 3 has demonstrated the applicability of the metrics to processes specified from the different adaptive BPEL-based frameworks. It has established the use of the metrics in the comparison not just of processes but also of specification frameworks. The metrics would facilitate an informed choice from among adaptive service compositions and frameworks. It is expected that the metrics and their potential utilisation are significant contributions toward considering adaptability as a first-class concern in the design and

implementation of service compositions.

Despite the demonstrated effectiveness of the adaptability metrics, the existing limitations have indicated several directions for future work.

- *Expansion of the metrics on non-workflow composition frameworks*. The case study in Chapter 3 has established the general applicability of the metrics to the selected representative frameworks (i.e., workflow-based SC frameworks). However, there is no guarantee of similar effectiveness in adaptability implementations outside workflow-based composition frameworks (e.g., self-organising adaptability [N. Chen, Cardozo & Clarke, 2018]) since the metrics are mainly based on BPEL's workflow-oriented specification constructs. It would be interesting to extend the metrics to accommodate the adaptability of non-workflow service composition frameworks.
- *Weighted prioritisation among variation points*. Presently, the metrics regard all variation points to have equal importance (i.e., equal prioritisation). However, there are variation points in the process that are vital to the overall operations. Hence, they should need more adaptability than less critical ones. The inclusion of variation point priorities in the metrics might enhance comprehensiveness and accuracy (e.g., a weighted adaptability measurement wherein the weights determine, for instance, the "importance" of adapting a process component).
- *Automated translation of process specification*. Non-BPEL process specifications, such as the Business Process Modelling Notation (BPMN), are manually translated into the input code specification of the support tool (see Section 3.5). Since BPMN is the most common process specification language, there is a need for an automated translator of BPMN to BPEL specifications. Having such a translator alleviates the burden of encoding, especially when involving large processes, and minimises translation errors.

- *Utilising probability in the variabilities of branching and looping constructs*. Determining the actual runtime variabilities can be tricky when involving branching and looping constructs. In such situations, the designers' knowledge of the application domain and their familiarity with the actual business structure and processes are essential. The branching constructs are dealt with through assuming some fixed, application-specific probabilities determined by designers. Likewise with the looping constructs, a predetermined number of concrete services (e.g., the average number of available services from the iterations) is assumed. Exploiting probability distributions and Monte Carlo-like simulations (Brogi et al., 2018; Bartoloni, Brogi & Ibrahim, 2014) for those branching and looping constructs might be beneficial towards a more accurate variability estimation.
- *Integration of adaptability in multiple quality trade-offs*. The importance of adaptability and the capability for its measurement support its inclusion as a property to consider in design trade-offs. Along this path, there is a need to explore the practical integration of adaptability in multiple quality trade-offs, and its evaluation against other service composition concerns. It is also interesting to look for ways to assess the mutual influence between process adaptability and the variability of other quality attributes.

## 6.2 Context-aware Requirements Variability Framework

Chapter 4 has proposed a goal-oriented requirements variability framework that integrates contextual preferences with contextual goals to reason out requirements variability in software systems. It is worth mentioning that the framework does not apply only to the engineering of adaptability in service-based systems but also in general software systems. To represent contextual information, the framework utilises a hierarchical

context specification enabling the abstraction and refinement of contexts. The framework has defined a contextual goal specification that annotates a goal model element with contextual information. A contextual goal expresses the situation when that goal is valid. Similarly, the framework has defined a contextual preference specification that annotates the traditional goal model preferences with contextual information. A contextual preference indicates the situation when a particular level of importance, which is denoted by a numerical score, is applied to an alternative hardgoal or a softgoal. A DLV-based reasoner is then utilised to automate the derivation of the optimal solutions to the requirements variability problem.

Moreover, Chapter 4 has presented an evaluation of the framework on three aspects. Firstly, the effectiveness of the approach was observed, both in identifying inconsistencies among contextual goals and validating contextual preferences. In the case study presented, utilising the approach helped recognise a significant number of inconsistencies that led to further revisions in the goal models and contextual goal specifications. With the use of iterative validation of contextual preferences (i.e., the iterative validation is a component approach of the framework), the designers have also made a number of contextual preference modifications furthering enhancements of the involved requirements models. Secondly, the relevance of contextual preferences in the goal model analysis was examined. The experiences in exploring the three goal model problems has established contextual preferences to be important in the reasoning process. In another experiment, a significant number of differences between the generated solutions applying contextual preferences and those without preferences or with fixed preferences was found, thus suggesting the effectiveness of contextual preferences. Thirdly, the scalability of the automated analysis tool was evaluated, the tool is shown to scale well with the small to medium size goal models.

Chapter 4 has presented a framework that combines the influence of context and preferences to requirements variability, supporting the exploration of potential design

alternatives at the early requirements stages. The framework provides a perspective of software systems adaptability via the lens of requirements variability. Moreover, the involved analysis activities allow designers to better understand the relationships among contexts, stakeholder goals, and preferences. Such activities *i*) help improve domain understanding and model accuracy, *ii*) complement priority elicitation by directly showing the impact of certain prioritisations, and *iii*) potentially support personalisation of software systems by associating the obtained high-level solutions into design or runtime configurations. In general, recognising contexts at the goal-level requirements analysis, i.e., through contextual goals and contextual preferences, initiates discussions and elaborations of important design measures. For instance, additional system modules to monitor relevant contexts associated with the contextual goals and preferences can be considered early in the development phase.

The present limitations of the proposed requirements variability framework have paved the way for several future works.

- *Comprehensive usability evaluation*. The case study in Chapter 4 significantly relied on the author's judgement in exploring the usability of the proposed approach. It would be interesting to extend the case study, to involve the practising requirements engineers and designers in the evaluation. The involvement of third parties would further establish a practical demonstration of the approach's usefulness. It would also be interesting to look for case studies that would integrate the approach with industrial requirements analysis of which more complex requirements models are expected.
- *Optimisation of the automated analysis algorithm and enhancement of the support tool*. Likewise, there is a need to further optimise the automated analysis to scale well with requirements problems having a large solution space. Activities of transforming a requirements model into a Disjunctive Datalog (DLV) input

and defining the logical relations among context values are still done manually. Hence, it is desirable to have a comprehensive automated tool with a graphical user interface to support every modelling activity involved in the framework, as well as to facilitate the practical evaluation.

- *Elicitation of priority scores*. The elicitation of priority scores is also a valuable undertaking to extend the framework in the future. The demonstration of the approach in Chapter 4 only utilises a range of score values indicating five levels of preference strength. The designers perform the specification of such scores as part of the iterative process of refining contextual preferences. It would be interesting to explore the integration of existing preference score elicitation approaches to the requirements variability analysis, i.e., from popular approaches such as the Analytic Hierarchy Process (Saaty, 2008) to newly proposed ones such as the CrowdRE-based preference elicitation (Groen et al., 2017).
- *Runtime or evolving contextualisation*. This is another important problem to address in the future. Requirements tend to change over time as the operational environment, users, and users' needs may change. The reasoning specifications (i.e., contextual goals and preferences) created during design-time may not be sufficient to handle the changes. *Runtime uncertainty* is a related adaptability concern that needs attention. This happens when a system is unable to determine the alternative configuration to hold at a certain point due to several factors, such as *i*) incomplete or inconsistent knowledge about the current contextual situation, *ii*) new (unpredicted) requirements having emerged, or *iii*) requirements having changed or disappeared. It is therefore desired that systems be able to evolve their designed adaptability behaviour to cope with these unexpected changes. With the ability of context-aware systems to monitor the environment, runtime data can be collected to learn the requirements that would better suit certain contexts. For instance, the history of users' interaction with the system in different contexts

can be used to formulate new contextual requirements.

## 6.3 Geographic-aware Service Recommender

Chapter 5 has proposed a geographic-aware collaborative filtering service recommendation approach, which is capable of utilising both the geographical and functional relevance of a service invocation to improve the precision of mashup-API recommendations. To derive the geographical relevance, the approach considers the location proximity *i*) between a target mashup and a potential API, *ii*) between the target mashup and its neighbouring mashups, and *iii*) between the potential API and its neighbouring APIs. Meanwhile, to derive the functional relevance, the approach compares the topic distribution of the textual descriptions of a mashup and API in the invocation pair. The derived geographical and functional relevance are integrated into the matrix factorisation model, so that the preference degrees behind the implicit mashup-API invocations can be inferred. The approach can facilitate full use of geographic information of mashups and APIs in the recommendation process, in addition to their functional descriptions. The intensive experimental analysis has indicated the effectiveness of the approach.

There are various key research directions to extend the proposed service recommendation approach.

- *Social-network-based recommendation*. The information that can be obtained from the social network (see Kalaï, Zayani, Amous, Abdelghani & Sèdes, 2018) and complex network (see Adeleye, Yu, Yongchareon & Han, 2018) analysis of mashups and APIs would augment the geographical and functional information being used by the recommendation model. The social relationships among neighbouring mashups and APIs can be significant additional contextual information. It would be interesting to examine the incorporation of social network data that

record the interaction of service users with the service data to infer hidden relationships between services (i.e., APIs). For instance, user activities manifested through social media services, such as postings of experiences, feedback, and questions on public forums, can be leveraged in the recommendation of services to other users for building mashups. Likewise, user behavioural patterns and interests can be extracted from event logs that are captured by social media platforms.

- *Dynamic recommendation in the big data setting*. The scalability of the recommendation approach for large-scale, heterogeneous service repositories is another interesting challenge. In addition to existing traditional Web services, cloud computing and the Internet of Things (IoT) have been influencing the emergence of modern Web services. There is a fast growth of these cloud-based and IoT-based services; the social networks among users, as well as services, are getting complex (i.e., big data). Moreover, the IoT environment is federated, where IoT services are often provided by independent providers with diverse interfaces, as well as business, cost, and QoS models (Bouguettaya et al., 2017). Therefore, it becomes desirable to have a recommender approach that can scale through the resultant big data, thereby significantly supporting the service composition that regards millions of diverse and heterogeneous services.
- *Online service recommendation*. Online service composition is a promising direction towards scalable and adaptable composition solutions to address large-scale, highly dynamic, and diverse big data services. Integrating the proposed recommendation approach to algorithms and models for online data processing could be an interesting future work.

# References


Abeywickrama, D. B. & Ovaska, E. (2017). A survey of autonomic computing methods in digital service ecosystems. *Service Oriented Computing and Applications*, *11*(1), 1–31.

Abowd, G. D., Dey, A. K., Brown, P. J., Davies, N., Smith, M. & Steggles, P. (1999). Towards a better understanding of context and context-awareness. In *Proceedings of the 1st international symposium on handheld and ubiquitous computing* (pp. 304–307). Springer.

Adacal, M. & Bener, A. B. (2006). Mobile web services: A new agent-based framework. *IEEE Internet Computing*, *10*(3), 58–65.

Adeleye, O., Yu, J., Yongchareon, S. & Han, Y. (2018). Constructing and evaluating an evolving web-api network for service discovery. In *Proceedings of the 16th international conference on service-oriented computing* (pp. 603–617). Springer.

Adomavicius, G. & Tuzhilin, A. (2011). Context-aware recommender systems. In *Recommender systems handbook* (pp. 217–253). Boston, MA: Springer.

Afify, Y. M., Moawad, I. F., Badr, N. L. & Tolba, M. F. (2014). Cloud services discovery and selection: survey and new semantic-based system. In *Bio-inspiring cyber security and cloud services: Trends and innovations* (pp. 449–477). Berlin: Springer.

Agrawal, R. & Wimmers, E. L. (2000). A framework for expressing and combining preferences. In *Acm sigmod record* (Vol. 29, pp. 297–306). New York, NY: ACM.

Akkiraju, R., Farrell, J., Miller, J. A., Nagarajan, M., Sheth, A. P. & Verma, K. (2005). *Web service semantics – wsdl-s* (Tech. Rep.). Athens, GA: University of Georgia Research Foundation, Inc.

Alabool, H., Kamil, A., Arshad, N. & Alarabiat, D. (2018). Cloud service evaluation method-based multi-criteria decision-making: A systematic literature review. *Journal of Systems and Software*, *139*, 161–188.

Al-alshuhai, A. & Siewe, F. (2015). An extension of the use case diagram to model context-aware applications. In *Proceedings of the sai intelligent systems conference* (pp. 884–888). IEEE.

Aldris, A., Nugroho, A., Lago, P. & Visser, J. (2013). Measuring the degree of service orientation in proprietary soa systems. In *Proceedings of the 7th ieee international symposium on service-oriented system engineering* (p. 233-244). IEEE Computer Society.

Alebrahim, A., Faßbender, S., Filipczyk, M., Goedicke, M., Heisel, M. & Zdun, U. (2016). Variability for qualities in software architecture. *ACM SIGSOFT Software Engineering Notes*, *41*(1), 32–35.

Alegre, U., Augusto, J. C. & Clark, T. (2016). Engineering context-aware systems and applications: A survey. *Journal of Systems and Software*, *117*, 55–83.

Alférez, G. H. & Pelechano, V. (2017). Achieving autonomic web service compositions with models at runtime. *Computers & Electrical Engineering*, *63*, 332–352.

Alférez, G. H., Pelechano, V., Mazo, R., Salinesi, C. & Diaz, D. (2014). Dynamic adaptation of service compositions with variability models. *Journal of Systems and Software*, *91*, 24–47.

Ali, R., Dalpiaz, F. & Giorgini, P. (2010). A goal-based framework for contextual requirements modeling and analysis. *Requirements Engineering*, *15*(4), 439–458.

Ali, R., Dalpiaz, F. & Giorgini, P. (2013). Reasoning with contextual requirements: detecting inconsistency and conflicts. *Information and Software Technology*, *55*(1), 35–57.

Ali, R., Dalpiaz, F. & Giorgini, P. (2014). Requirements-driven deployment. *Software & Systems Modeling*, *13*(1), 433–456.

Al-Masri, E. & Mahmoud, Q. H. (2008). Investigating web services on the world wide web. In *Proceedings of the 17th international conference on world wide web* (pp. 795–804). ACM.

Alviano, M., Faber, W., Leone, N., Perri, S., Pfeifer, G. & Terracina, G. (2011). The disjunctive datalog system dlv. In *Proceedings of the international datalog 2.0 workshop* (pp. 282–301). Springer.

Andreozzi, S., De Bortoli, N., Fantinel, S., Ghiselli, A., Rubini, G. L., Tortone, G. & Vistoli, M. C. (2005). Gridice: a monitoring service for grid systems. *Future Generation Computer Systems*, *21*(4), 559–571.

Andresen, K. & Gronau, N. (2005). An approach to increase adaptability in erp systems. In *Proceedings of the 2005 information resources management association international conference* (pp. 883–885). Idea Group Inc.

Andrikopoulos, V., Bertoli, P., Bindelli, S., Nitto, E. D., Gehlert, A., Germanovich, L., ... Weyer, T. (2008). *State of the art report on software engineering design knowledge and survey of hci and contextual knowledge (po-jra-1.1.1)* (Tech. Rep.). European Community's Seventh Framework Programme: S-Cube Consortium.

Angelopoulos, K., Papadopoulos, A., Souza, V. E. S. & Mylopoulos, J. (2018). Engineering self-adaptive software systems: from requirements to model predictive control. *ACM Transactions on Autonomous and Adaptive Systems*, *13*(1), 1–27.

Ankolekar, A., Burstein, M., Hobbs, J. R., Lassila, O., Martin, D., McDermott, D., ... Sycara, K. (2002). Daml-s: Web service description for the semantic web. In *Proceedings of the 1st international semantic web conference* (pp. 348–363). Springer.

Anselmi, J., Ardagna, D. & Cremonesi, P. (2007). A qos-based selection approach of autonomic grid services. In *Proceedings of the workshop on service-oriented computing performance: Aspects, issues, and approaches* (pp. 1–8). ACM.

Anton, A. I. (1996). Goal-based requirements analysis. In *Proceedings of the 2nd*

*international conference on requirements engineering* (pp. 136–144). IEEE Computer Society.

Ardagna, D., Comuzzi, M., Mussi, E., Pernici, B. & Plebani, P. (2007). Paws: A framework for executing adaptive web-service processes. *IEEE Software*, *24*(6), 39–46.

Ardagna, D. & Pernici, B. (2007). Adaptive service composition in flexible processes. *IEEE Transactions on Software Engineering*, *33*(6), 369–384.

Arsanjani, A., Ghosh, S., Allam, A., Abdollah, T., Ganapathy, S. & Holley, K. (2008). Soma: A method for developing service-oriented solutions. *IBM Systems Journal*, *47*(3), 377–396.

Aschoff, R. R., Zisman, A. & Alexandre, P. (2019). Parallel adaptation of multiple service composition instances. In *Engineering adaptive software systems* (pp. 115–134). Singapore: Springer.

Autili, M., Di Salle, A., Gallo, F., Pompilio, C. & Tivoli, M. (2018). On the model-driven synthesis of adaptable choreographies. In *Proceedings of models workshops* (pp. 12–17). CEUR-WS.

Bagga, P., Joshi, A. & Hans, R. (2019). Qos based web service selection and multi-criteria decision making methods. *International Journal of Interactive Multimedia & Artificial Intelligence*, *5*(4), 113–121.

Banerjee, S., Kapur, S. S., Merrill, J., Ganesan, A., Dutta, D., Dai, H. & Ghosh, A. K. (2014). *Placement of a cloud service using network topology and infrastructure performance. https://patents.google.com/patent/us20120239792a1/en.* Google Patents.

Barakat, L., Miles, S. & Luck, M. (2018). Adaptive composition in dynamic service environments. *Future Generation Computer Systems*, *80*, 215–228.

Baresi, L., Guinea, S. & Pasquale, L. (2007). Self-healing bpel processes with dynamo and the jboss rule engine. In *Proceedings of the international workshop on engineering of software services for pervasive environments* (pp. 11–20). ACM.

Baresi, L., Guinea, S. & Pasquale, L. (2012). Service-oriented dynamic software product lines. *Computer*, *45*(10), 42–48.

Barreto, C., Bullard, V., Erl, T., Evdemon, J., Jordan, D., Kand, K., ... Yiu, A. (2007). *Web services business process execution language version 2.0* (Tech. Rep.). WS-BPEL 2.0 Primer: OASIS. (Retrieved from https://www.oasis-open.org/committees/download.php/23974/wsbpel-v2.0-primer.pdf)

Bartoloni, L., Brogi, A. & Ibrahim, A. (2014). Probabilistic prediction of the qos of service orchestrations: a truly compositional approach. In *Proceedings of the 12th international conference on service-oriented computing* (pp. 378–385). Springer.

Bauer, V., Broy, M., Irlbeck, M., Leuxner, C., Spichkova, M., Dahlweid, M. & Santen, T. (2013). *Survey of modeling and engineering aspects of self-adapting & self-optimizing systems.* (Retrieved from https://mediatum.ub.tum.de/attfile/1130307/hd2/incoming/2013-Feb/857478.pdf)

Belkin, M. & Niyogi, P. (2001). Laplacian eigenmaps and spectral techniques for embedding and clustering. In *Proceedings of the advances in neural information*

*processing systems* (pp. 585–591). MIT Press.

Bell, M. (2008). *Service-oriented modeling: Service analysis, design, and architecture*. Hoboken, NJ: Wiley & Sons.

Bell, R. M. & Koren, Y. (2007). Scalable collaborative filtering with jointly derived neighborhood interpolation weights. In *Proceedings of the 7th ieee international conference on data mining* (pp. 43–52). IEEE Computer Society.

Benatallah, B., Casati, F., Grigori, D., Nezhad, H. R. M. & Toumani, F. (2005). Developing adapters for web services integration. In *Proceedings of the international conference on advanced information systems engineering* (pp. 415–429). Springer.

Bencomo, N., Götz, S. & Song, H. (2019). Models@run.time: a guided tour of the state of the art and research challenges. *Software & Systems Modeling*, *18*, 3049—3082.

Bencomo, N., Hallsteinsen, S. & De Almeida, E. S. (2012). A view of the dynamic software product line landscape. *Computer*, *45*(10), 36–41.

Bencomo, N., Whittle, J., Sawyer, P., Finkelstein, A. & Letier, E. (2010). Requirements reflection: requirements as runtime entities. In *Proceedings of the 32nd acm/ieee international conference on software engineering* (Vol. 2, pp. 199–202). ACM.

Bennaceur, A., McCormick, C., García-Galán, J., Perera, C., Smith, A., Zisman, A. & Nuseibeh, B. (2016). Feed me, feed me: an exemplar for engineering adaptive software. In *Proceedings of the 11th international symposium on software engineering for adaptive and self-managing systems* (pp. 89–95). ACM.

Benslimane, D., Dustdar, S. & Sheth, A. (2008). Services mashups: the new generation of web applications. *IEEE Internet Computing*, *12*(5), 13–15.

Bianculli, D., Jurca, R., Binder, W., Ghezzi, C. & Faltings, B. (2007). Automated dynamic maintenance of composite services based on service reputation. In *Proceedings of the 5th international conference on service-oriented computing* (pp. 449–455). Springer.

Bistarelli, S., Pini, M. S., Rossi, F. & Venable, K. B. (2005). Positive and negative preferences. In *Proceedings of cp 2005 workshop on preferences and soft constraints.*

Blake, M. B. & Nowlan, M. F. (2007). A web service recommender system using enhanced syntactical matching. In *Proceedings of the ieee international conference on web services* (pp. 575–582). IEEE Computer Society.

Blei, D. M. (2012). Probabilistic topic models. *Commun. ACM*, *55*(4), 77–84.

Blei, D. M., Ng, A. Y. & Jordan, M. I. (2003). Latent dirichlet allocation. *Journal of Machine Learning Research*, *3*, 993–1022.

Blom, J. (2000). Personalization: a taxonomy. In *Proceedings of the chi'00 extended abstracts on human factors in computing systems* (pp. 313–314). ACM.

Bobadilla, J., Ortega, F., Hernando, A. & Gutiérrez, A. (2013). Recommender systems survey. *Knowledge-based Systems*, *46*, 109–132.

Boehm, B., Bose, P., Horowitz, E. & Lee, M. J. (1995). Software requirements negotiation and renegotiation aids: a theory-w based spiral approach. In *Proceedings of the 17th international conference on software engineering* (pp. 243–243). ACM.

Bosch, J., Capilla, R. & Hilliard, R. (2015). Trends in systems and software variability [guest editors' introduction]. *IEEE Software*, *32*(3), 44–51.

Botangen, K. A., Yu, J., Han, Y., Sheng, Q. Z. & Han, J. (2020). Quantifying the adaptability of workflow-based service compositions. *Future Generation Computer Systems*, *102*, 95-111.

Botangen, K. A., Yu, J. & Sheng, M. (2017). Towards measuring the adaptability of an ao4bpel process. In *Proceedings of the australasian computer science week multiconference* (pp. 6:1–6:7). ACM.

Botangen, K. A., Yu, J., Sheng, Q. Z., Han, Y. & Yongchareon, S. (2020). Geographic-aware collaborative filtering for web service recommendation. *Expert Systems with Applications*, *151*, 113347.

Botangen, K. A., Yu, J., Yeap, W. K. & Sheng, Q. Z. (2020). Integrating context to preferences and goals for goal-oriented adaptability of software systems. *The Computer Journal*, *DOI: 10.1093/comjnl/bxz167*.

Botangen, K. A., Yu, J., Yongchareon, S., Yang, L. & Sheng, Q. Z. (2019). Integrating geographical and functional relevance to implicit data for web service recommendation. In *Proceedings of the 17th international conference on service-oriented computing* (pp. 53–57). Springer.

Botangen, K. A., Yu, J., Yongchareon, S., Yang, L. H. & Bai, Q. (2018). Specifying and reasoning about contextual preferences in the goal-oriented requirements modelling. In *Proceedings of the australasian computer science week multiconference* (pp. 47:1–47:10). ACM.

Bouguettaya, A., Sheng, Q. Z. & Daniel, F. (2014a). *Advanced web services*. Springer.

Bouguettaya, A., Sheng, Q. Z. & Daniel, F. (2014b). *Web services foundations*. Springer.

Bouguettaya, A., Singh, M., Huhns, M., Sheng, Q. Z., Dong, H., Yu, Q., ... Papazoglou, M. (2017). A service computing manifesto: the next 10 years. *Communications of the ACM*, *60*(4), 64–72.

Boukhebouze, M., Amghar, Y., Benharkat, A.-N. & Maamar, Z. (2009). Towards self-healing execution of business processes based on rules. In *Proceedings of the 11th international conference on enterprise information systems* (pp. 501–512). Springer.

Boyle, J. M. (1979). *Program adaptation and program transformation* (Tech. Rep.). Lemont, Illinois: Argonne National Lab.

Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F. & Mylopoulos, J. (2004). Tropos: An agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems*, *8*(3), 203–236.

Broens, T., Pokraev, S., Van Sinderen, M., Koolwaaij, J. & Costa, P. D. (2004). Context-aware, ontology-based service discovery. In *Proceedings of the 2nd european symposium on ambient intelligence* (pp. 72–83). Springer.

Brogi, A., Danelutto, M., De Sensi, D., Ibrahim, A., Soldani, J. & Torquati, M. (2018). Analysing multiple qos attributes in parallel design patterns-based applications. *International Journal of Parallel Programming*, *46*(1), 81–100.

Brogi, A. & Popescu, R. (2006). Automated generation of bpel adapters. In *Proceedings*

*of the 4th international conference on service-oriented computing* (pp. 27–39). Springer.

Brown, A., Johnston, S. K., Larsen, G. & Palistrant, J. (2005). Soa development using the ibm rational software development platform: a practical guide [Computer software manual]. Rational Software.

Brun, Y., Serugendo, G. D. M., Gacek, C., Giese, H., Kienle, H., Litoiu, M., ... Shaw, M. (2009). Engineering self-adaptive systems through feedback loops. In *Software engineering for self-adaptive systems* (pp. 48–70). Berlin: Springer.

Bucchiarone, A., Cappiello, C., Di Nitto, E., Kazhamiakin, R., Mazza, V. & Pistore, M. (2009). Design for adaptation of service-based applications: main issues and requirements. In *Proceedings of the service-oriented computing icsoc/servicewave 2009 workshops* (pp. 467–476). Springer.

Bucchiarone, A., Marconi, A., Mezzina, C. A., Pistore, M. & Raik, H. (2013). On-the-fly adaptation of dynamic service-based systems: incrementality, reduction and reuse. In *Proceedings of the 11th international conference on service-oriented computing* (pp. 146–161). Springer.

Calinescu, R., Grunske, L., Kwiatkowska, M., Mirandola, R. & Tamburrelli, G. (2010). Dynamic qos management and optimization in service-based systems. *IEEE Transactions on Software Engineering*, *37*(3), 387–409.

Canfora, G., Di Penta, M., Esposito, R. & Villani, M. L. (2005). An approach for qos-aware service composition based on genetic algorithms. In *Proceedings of the 7th annual conference on genetic and evolutionary computation* (pp. 1069–1075). ACM.

Cao, B., Liu, X., Rahman, M. M., Li, B., Liu, J. & Tang, M. (2017). Integrated content and network-based service clustering and web apis recommendation for mashup development. *IEEE Transactions on Services Computing*, *DOI: 10.1109/TSC.2017.2686390*.

Cao, Y., Liu, J., Shi, M., Cao, B., Chen, T. & Wen, Y. (2019). Service recommendation based on attentional factorization machine. In *Proceedings of the ieee international conference on services computing* (pp. 189–196). IEEE.

Capilla, R., Bosch, J., Trinidad, P., Ruiz-Cortés, A. & Hinchey, M. (2014). An overview of dynamic software product line architectures and techniques: observations from research and industry. *Journal of Systems and Software*, *91*, 3–23.

Capilla, R., Ortiz, Ó. & Hinchey, M. (2014). Context variability for context-aware systems. *IEEE Computer*, *47*(2), 85–87.

Caporuscio, M., Grassi, V., Marzolla, M. & Mirandola, R. (2016). Goprime: A fully decentralized middleware for utility-aware service assembly. *IEEE Transactions on Software Engineering*, *42*(2), 136–152.

Cardellini, V., Casalicchio, E., Grassi, V., Iannucci, S., Presti, F. L. & Mirandola, R. (2011). Moses: A framework for qos driven runtime adaptation of service-oriented systems. *IEEE Transactions on Software Engineering*, *38*(5), 1138–1159.

Casati, F., Ceri, S., Pernici, B. & Pozzi, G. (1998). Workflow evolution. *Data & Knowledge Engineering*, *24*(3), 211–238.

Casati, F. & Shan, M.-C. (2001). Dynamic and adaptive composition of e-services. *Information Systems*, *26*(3), 143–163.

Castro, J., Kolp, M. & Mylopoulos, J. (2002). Towards requirements-driven information systems engineering: the tropos project. *Information Systems*, *27*(6), 365–389.

Cavallaro, L., Sawyer, P., Sykes, D., Bencomo, N. & Issarny, V. (2012). Satisfying requirements for pervasive service compositions. In *Proceedings of the 7th workshop on models@run.time* (pp. 17–22). ACM.

Cetina, C., Giner, P., Fons, J. & Pelechano, V. (2009). Using feature models for developing self-configuring smart homes. In *Proceedings of the 5th international conference on autonomic and autonomous systems* (pp. 179–188). IEEE Computer Society.

Chafle, G., Dasgupta, K., Kumar, A., Mittal, S. & Srivastava, B. (2006). Adaptation in web service composition and execution. In *Proceedings of the ieee international conference on web services* (pp. 549–557). IEEE Computer Society.

Chang, C. K., Jiang, H.-y., Ming, H. & Oyama, K. (2009). Situ: a situation-theoretic approach to context-aware service evolution. *IEEE Transactions on Services Computing*, *2*(3), 261–275.

Chang, J., Gerrish, S., Wang, C., Boyd-Graber, J. L. & Blei, D. M. (2009). Reading tea leaves: How humans interpret topic models. In *Proceedings of the 22nd international conference on neural information processing systems* (pp. 288–296). ACM.

Chang, S. H. (2007). A systematic analysis and design approach to develop adaptable services in service oriented computing. In *Proceedings of the ieee international conference on services computing - workshops* (pp. 375–378). IEEE Computer Society.

Chang, S. H. & Kim, S. D. (2007). A variability modeling method for adaptable services in service-oriented computing. In *Proceedings of the 11th international software product lines conference* (pp. 261–268). IEEE Computer Society.

Channabasavaiah, K., Holley, K. & Tuggle, E. (2004, 4). *Migrating to a service-oriented architecture* (Tech. Rep.). New York: IBM Corporation Software Group.

Charfi, A. & Mezini, M. (2004). Hybrid web service composition: business processes meet business rules. In *Proceedings of the 2nd international conference on service oriented computing* (pp. 30–38). ACM.

Charfi, A. & Mezini, M. (2007). Ao4bpel: An aspect-oriented extension to bpel. *World Wide Web*, *10*(3), 309–344.

Chen, F., Dou, R., Li, M. & Wu, H. (2016). A flexible qos-aware web service composition method by multi-objective optimization in cloud manufacturing. *Computers & Industrial Engineering*, *99*, 423–431.

Chen, H.-p. & Li, S.-c. (2010). Src: a service registry on cloud providing behavior-aware and qos-aware service discovery. In *Proceedings of the ieee international conference on service-oriented computing and applications* (pp. 1–4). IEEE Computer Society.

Chen, N., Cardozo, N. & Clarke, S. (2018). Goal-driven service composition in mobile and pervasive computing. *IEEE Transactions on Services Computing*, *11*(1),

49–62.

Chen, T.-H., Thomas, S. W. & Hassan, A. E. (2016). A survey on the use of topic models when mining software repositories. *Empirical Software Engineering*, *21*(5), 1843–1919.

Chen, X., Liu, X., Huang, Z. & Sun, H. (2010). Regionknn: A scalable hybrid collaborative filtering algorithm for personalized web service recommendation. In *Proceedings of the ieee international conference on web services* (pp. 9–16). IEEE Computer Society.

Chen, X., Zheng, Z., Liu, X., Huang, Z. & Sun, H. (2013). Personalized qos-aware web service recommendation and visualization. *IEEE Transactions on Services Computing*, *6*(1), 35–47.

Cheng, B. H. C., de Lemos, R., Giese, H., Inverardi, P., Magee, J., Andersson, J., ... Whittle, J. (2009). Software engineering for self-adaptive systems: a research roadmap. In *Software engineering for self-adaptive systems. lecture notes in computer science* (pp. 1–26). Berlin, Heidelberg: Springer.

Choi, J. (2007). Context-driven requirements analysis. In *Proceedings of the international conference on computational science and its applications* (pp. 739–748). Springer.

Christophe, B., Boussard, M., Lu, M., Pastor, A. & Toubiana, V. (2011). The web of things vision: things as a service and interaction patterns. *Bell Labs Technical Journal*, *16*(1), 55–61.

Chung, L. & do Prado Leite, J. C. S. (2009). On non-functional requirements in software engineering. In *Conceptual modeling: foundations and applications* (pp. 363–379). Berlin, Heidelberg: Springer.

Coello, C. A., Lamont, G. B. & Van Veldhuizen, D. A. (2007). *Evolutionary algorithms for solving multi-objective problems* (Vol. 5). Boston, MA: Springer.

Cognini, R., Corradini, F., Gnesi, S., Polini, A. & Re, B. (2018). Business process flexibility-a systematic literature review with a software systems perspective. *Information Systems Frontiers*, *20*(2), 343–371.

Colombo, M., Di Nitto, E. & Mauri, M. (2006). Scene: A service composition execution environment supporting dynamic changes disciplined through rules. In *Proceedings of the 4th international conference on service-oriented computing* (pp. 191–202). Springer.

Conrad, M. (1983). *Adaptability: The significance of variability from molecule to ecosystem*. New York: Plenum Press.

Console, L. & Fugini, M. (2007). Ws-diamond: An approach to web services—diagnosability, monitoring and diagnosis. In *Expanding the knowledge economy: Issues, applications, case studies.* Amsterdam: IOS Press.

Dai, Y., Yang, L. & Zhang, B. (2009). Qos-driven self-healing web service composition based on performance prediction. *Journal of Computer Science and Technology*, *24*(2), 250–261.

Dalpiaz, F., Franch, X. & Horkoff, J. (2016). *istar 2.0 language guide.* (arXiv preprint arXiv:1605.07767)

Dalpiaz, F., Giorgini, P. & Mylopoulos, J. (2013). Adaptive socio-technical systems: a requirements-based approach. *Requirements Engineering*, *18*(1), 1–24.

Daniel, F. & Matera, M. (2014). Mashups. In *Mashups: Concepts, models and architectures* (pp. 137–181). Berlin, Heidelberg: Springer.

Darimont, R., Delor, E., Massonet, P. & van Lamsweerde, A. (1997). Grail/kaos: an environment for goal-driven requirements engineering. In *Proceedings of the 19th international conference on software engineering* (pp. 612–613). ACM.

Dastjerdi, A. V., Tabatabaei, S. G. H. & Buyya, R. (2010). An effective architecture for automated appliance management system applying ontology-based cloud discovery. In *Proceedings of the 10th ieee/acm international conference on cluster, cloud and grid computing* (pp. 104–112). IEEE Computer Society.

De Bruijn, J., Lausen, H., Polleres, A. & Fensel, D. (2006). The web service modeling language wsml: an overview. In *Proceedings of the 3rd european semantic web conference* (pp. 590–604). Springer.

De Lemos, R., Giese, H., Müller, H. A., Shaw, M., Andersson, J., Litoiu, M., ... Vogel, T. (2013). Software engineering for self-adaptive systems: A second research roadmap. In *Software engineering for self-adaptive systems ii* (pp. 1–32). Berlin, Heidelberg: Springer.

De Sanctis, M. & Marconi, A. (2018). A design for adaptation framework for self-adaptive systems. In *Proceedings of the 3rd ieee international workshops on foundations and applications of self* systems* (pp. 3–4). IEEE.

Dey, S. & Lee, S.-W. (2017). Reassure: Requirements elicitation for adaptive socio-technical systems using repertory grid. *Information and Software Technology*, *87*, 160–179.

Dinh, H. T., Lee, C., Niyato, D. & Wang, P. (2013). A survey of mobile cloud computing: architecture, applications, and approaches. *Wireless Communications and Mobile Computing*, *13*(18), 1587–1611.

Di Nitto, E., Di Penta, M., Gambi, A., Ripa, G. & Villani, M. L. (2007). Negotiation of service level agreements: An architecture and a search-based approach. In *Proceedings of the 5th international conference on service-oriented computing* (pp. 295–306). Springer.

Di Penta, M., Bastida, L., Sillitti, A., Baresi, L., Maiden, N., Melideo, M., ... Ripa, G. (2009). Secse – service-centric systems engineering: an overview. In *At your service: Service-oriented computing from an eu perspective.* Cambridge, MA: MIT Press.

Domingue, J., Fensel, D., Davies, J., González-Cabero, R. & Pedrinaci, C. (2009). The service web: a web of billions of services. In *Towards a future internet: a european research perspective* (pp. 203–216). Amsterdam: IOS Press.

Doncaster, L. (1917). Adaptation and adaptability. *The Eugenics review*, *9*(3), 213–217.

Dong, H., Hussain, F. K. & Chang, E. (2011). A service concept recommendation system for enhancing the dependability of semantic service matchmakers in the service ecosystem environment. *Journal of Network and Computer Applications*, *34*(2), 619–631.

Dong, X., Halevy, A., Madhavan, J., Nemes, E. & Zhang, J. (2004). Similarity search

for web services. In *Proceedings of the 30th international conference on very large databases-volume 30* (pp. 372–383). VLDB Endowment.

Dourish, P. (2004). What we talk about when we talk about context. *Personal and Ubiquitous Computing*, *8*(1), 19–30.

Duan, Y., Fu, G., Zhou, N., Sun, X., Narendra, N. C. & Hu, B. (2015). Everything as a service (xaas) on the cloud: origins, current and future trends. In *Proceedings of the 8th ieee international conference on cloud computing* (pp. 621–628). IEEE Computer Society.

Dueñas, J. C., de Oliveira, W. L. & de la Puente, J. A. (1998). A software architecture evaluation model. In F. van der Linden (Ed.), *2nd international esprit ares workshop* (pp. 148–157). Heidelberg: Springer.

Duran, M. B. & Mussbacher, G. (2019). Reusability in goal modeling: a systematic literature review. *Information and Software Technology*, *110*, 156–173.

Durvasula, S. (2006). Soa practitioners' guide part 3: introduction to services lifecycle [Computer software manual]. BEA Systems, Inc. (Retrieved from https://www.soablueprint.com/whitepapers/SOAPGPart3.pdf)

Dustdar, S. & Schreiner, W. (2005). A survey on web services composition. *International Journal of Web and Grid Services*, *1*(1), 1–30.

Eiben, A. E. (2004). Evolutionary computing and autonomic computing: Shared problems, shared solutions? In *Self-star properties in complex information systems, conceptual and practical foundations* (pp. 36–48). Berlin: Springer.

Ernst, N. A., Mylopoulos, J., Borgida, A. & Jureta, I. J. (2010). Reasoning with optional and preferred requirements. In *Proceedings of the 29th international conference on conceptual modeling* (pp. 118–131). Springer.

Erradi, A., Maheshwari, P. & Tosic, V. (2006). Policy-driven middleware for self-adaptation of web services compositions. In *Proceedings of the 7th international middleware conference* (pp. 62–80). Springer.

Evans, M. & Marciniak, J. (1987). *Software quality assurance management*. New York: John Wiley & Sons, Inc.

Fenton, N. E. (1991). *Software metrics: a rigorous approach*. London, UK: Chapman & Hall, Ltd.

Fielding, R. T. (2000). *Architectural styles and the design of network-based software architectures* (Unpublished doctoral dissertation). University of California, Irvine.

Finkelstein, A. & Savigni, A. (2001). A framework for requirements engineering for context-aware services. In *Proceedings of the 23rd international conference on software engineering*. IEEE Computer Society.

Fisher, R. & Ford, E. (1926). Variability of species. *Nature*, *118*(2971), 515–516.

Fleuren, T. & Müller, P. (2008). Bpel workflows combining standard ogc web services and grid-enabled ogc web services. In *Proceedings of the 34th euromicro conference software engineering and advanced applications* (pp. 337–344). IEEE Computer Society.

Foster, I., Kesselman, C., Nick, J. M. & Tuecke, S. (2002). Grid services for distributed system integration. *Computer*(6), 37–46.

Friese, T., Müller, J. P. & Freisleben, B. (2005). Self-healing execution of business processes based on a peer-to-peer service architecture. In *Proceedings of the 18th international conference on architecture of computing systems* (pp. 108–123). Springer.

Furno, A. & Zimeo, E. (2014). Context-aware composition of semantic web services. *Mobile Networks and Applications*, *19*(2), 235–248.

Galster, M., Weyns, D., Tofan, D., Michalik, B. & Avgeriou, P. (2014). Variability in software systems—a systematic literature review. *IEEE Transactions on Software Engineering*, *40*(3), 282–306.

Galster, M., Zdun, U., Weyns, D., Rabiser, R., Zhang, B., Goedicke, M. & Perrouin, G. (2017). Variability and complexity in software design: towards a research agenda. *ACM SIGSOFT Software Engineering Notes*, *41*(6), 27–30.

Gamez, N., Fuentes, L. & Aragüez, M. A. (2011). Autonomic computing driven by feature models and architecture in famiware. In *Proceedings of the 5th european conference on software architecture* (pp. 164–179). Springer.

Garlan, D. (2017). Foreword by david garlan. In I. Mistrik, N. Ali, R. Kazman, J. Grundy & B. Schmerl (Eds.), *Managing trade-offs in adaptable software architectures* (p. xix - xx). Boston: Morgan Kaufmann.

Gaß, O., Ortbach, K., Kretzer, M., Maedche, A. & Niehaves, B. (2015). Conceptualizing individualization in information systems – a literature review. *Communications of the Association for Information Systems*, *37*.

Gilb, T. (1988). *Principles of software engineering management*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Gillain, J., Faulkner, S., Jureta, I. J. & Snoeck, M. (2013). Using goals and customizable services to improve adaptability of process-based service compositions. In *Proceedings of the ieee 7th international conference on research challenges in information science* (pp. 1–9). IEEE.

Giorgini, P., Mylopoulos, J., Nicchiarelli, E. & Sebastiani, R. (2002). Reasoning with goal models. In *Proceedings of the 21st international conference on conceptual modeling* (pp. 167–181). Springer.

Godse, M., Sonar, R. & Mulik, S. (2008). The analytical hierarchy process approach for prioritizing features in the selection of web service. In *Proceedings of the 6th european conference on web services* (pp. 41–50). IEEE Computer Society.

Groen, E. C., Seyff, N., Ali, R., Dalpiaz, F., Doerr, J., Guzman, E., ... Stade, M. (2017). The crowd in requirements engineering: The landscape and challenges. *IEEE software*, *34*(2), 44–52.

Gronmo, R., Skogan, D., Solheim, I. & Oldevik, J. (2004). Model-driven web service development. *International Journal of Web Services Research (IJWSR)*, *1*(4), 1–13.

Gu, Q. & Lago, P. (2011). Guiding the selection of service-oriented software engineering methodologies. *Service Oriented Computing and Applications*, *5*(4), 203–223.

Guinard, D., Trifa, V. & Wilde, E. (2010). A resource oriented architecture for the web of things. In *Proceedings of the 2nd international internet of things conference* (pp. 1–8). IEEE.

Guinea, A. S., Nain, G. & Le Traon, Y. (2016). A systematic review on the engineering of software for ubiquitous systems. *Journal of Systems and Software*, *118*, 251–276.

Guo, L., Wang, S., Kang, L. & Cao, Y. (2015). Agent-based manufacturing service discovery method for cloud manufacturing. *International Journal of Advanced Manufacturing Technology*, *81*(9-12), 2167–2181.

Hagen, C. & Alonso, G. (2000). Exception handling in workflow management systems. *IEEE Transactions on Software Engineering*, *26*(10), 943–958.

Hallerbach, A., Bauer, T. & Reichert, M. (2010). Capturing variability in business process models: the provop approach. *Journal of Software Maintenance and Evolution: Research and Practice*, *22*(6-7), 519–546.

Hallsteinsen, S., Geihs, K., Paspallis, N., Eliassen, F., Horn, G., Lorenzo, J., ... Papadopoulos, G. A. (2012). A development framework and methodology for self-adapting applications in ubiquitous computing environments. *Journal of Systems and Software*, *85*(12), 2840–2859.

Han, S. N. & Crespi, N. (2017). Semantic service provisioning for smart objects: Integrating iot applications into the web. *Future Generation Computer Systems*, *76*, 180–197.

Hartmann, H. & Trew, T. (2008). Using feature diagrams with context variability to model multiple product lines for software supply chains. In *Proceedings of the 12th international software product lines conference* (pp. 12–21). IEEE Computer Society.

Henricksen, K. (2003). *A framework for context-aware pervasive computing applications* (Unpublished doctoral dissertation). University of Queensland.

Henricksen, K. & Indulska, J. (2004). A software engineering framework for context-aware pervasive computing. In *Proceedings of the 2nd ieee annual conference on pervasive computing and communications* (pp. 77–86). IEEE Computer Society.

Henricksen, K. & Indulska, J. (2006). Developing context-aware pervasive computing applications: models and approach. *Pervasive and mobile computing*, *2*(1), 37–64.

Henricksen, K., Indulska, J. & Rakotonirainy, A. (2006). Using context and preferences to implement self-adapting pervasive computing applications. *Software: Practice and Experience*, *36*(11-12), 1307–1330.

Herlocker, J. L., Konstan, J. A., Borchers, A. & Riedl, J. (1999). An algorithmic framework for performing collaborative filtering. In *Proceedings of the 22nd international acm sigir conference on research and development in information retrieval* (pp. 230–237). ACM.

Hielscher, J., Metzger, A. & Kazhamiakin, R. (2009). *Taxonomy of adaptation principles and mechanisms (cd-jra-1.2.2)* (Tech. Rep.). European Community's Seventh Framework Programme: S-Cube Consortium.

Hinchey, M., Park, S. & Schmid, K. (2012). Building dynamic software product lines. *IEEE Computer*, *45*(10), 22–26.

Horkoff, J., Borgida, A., Mylopoulos, J., Barone, D., Jiang, L., Yu, E. & Amyot, D. (2012). Making data meaningful: the business intelligence model and its formal

semantics in description logics. In *Proceedings of the on the move to meaningful internet systems: Otm 2012, confederated international conferences: Coopis, doa-svi, and odbase* (pp. 700–717). Springer.

Hu, Y., Koren, Y. & Volinsky, C. (2008). Collaborative filtering for implicit feedback datasets. In *Proceedings of the 8th ieee international conference on data mining* (pp. 263–272). IEEE Computer Society.

Huang, A. F., Lan, C.-W. & Yang, S. J. (2009). An optimal qos-based web service selection scheme. *Information Sciences*, *179*(19), 3309–3322.

Huang, G., He, J. & Zhang, Y. (2014). Web services for things. In *Advanced web services* (p. 605-629). New York, NY: Springer.

Hui, B., Liaskos, S. & Mylopoulos, J. (2003). Requirements analysis for customizable software: a goals-skills-preferences framework. In *Proceedings of the 11th ieee international requirements engineering conference* (pp. 117–126). IEEE Computer Society.

In, H., Boehm, B., Rodger, T. & Deutsch, M. (2001). Applying winwin to quality requirements: a case study. In *Proceedings of the 23rd international conference on software engineering* (pp. 555–564). IEEE Computer Society.

ISO. (2001). Iso/iec 9126-1:2001 software engineering – product quality – part 1: Quality model [Computer software manual]. International Organization for Standardization. (Retrieved from https://www.iso.org/standard/22749.html)

ISO. (2006). Iso/iec 14764:2006 software engineering – software life cycle processes – maintenance [Computer software manual]. International Organization for Standardization. (Retrieved from https://www.iso.org/standard/39064.html)

ISO. (2011). Iso/iec 25010:2011 systems and software engineering – systems and software quality requirements and evaluation (square) – system and software quality models [Computer software manual]. International Organization for Standardization. (Retrieved from https://www.iso.org/standard/35733.html)

Jawaheer, G., Szomszor, M. & Kostkova, P. (2010). Comparison of implicit and explicit feedback from an online music recommendation service. In *Proceedings of the 1st international workshop on information heterogeneity and fusion in recommender systems* (pp. 47–51). ACM.

Jelodar, H., Wang, Y., Yuan, C., Feng, X., Jiang, X., Li, Y. & Zhao, L. (2019). Latent dirichlet allocation (lda) and topic modeling: models, applications, a survey. *Multimedia Tools and Applications*, *78*(11), 15169–15211.

Jiao, H., Zhang, J., Li, J. H. & Shi, J. (2017). Research on cloud manufacturing service discovery based on latent semantic preference about owl-s. *International Journal of Computer Integrated Manufacturing*, *30*(4-5), 433–441.

Jurca, R., Faltings, B. & Binder, W. (2007). Reliable qos monitoring based on client feedback. In *Proceedings of the 16th international conference on world wide web* (pp. 1003–1012). ACM.

Jureta, I. J., Borgida, A., Ernst, N. A. & Mylopoulos, J. (2010). Techne: Towards a new generation of requirements modeling languages with goals, preferences, and inconsistency handling. In *Proceedings of the 18th ieee international requirements engineering conference* (pp. 115–124). IEEE Computer Society.

Kaddoum, E., Raibulet, C., Georgé, J.-P., Picard, G. & Gleizes, M.-P. (2010). Criteria for the evaluation of self-* systems. In *Proceedings of the icse workshop on software engineering for adaptive and self-managing systems* (pp. 29–38). ACM.

Kalaï, A., Zayani, C. A., Amous, I., Abdelghani, W. & Sèdes, F. (2018). Social collaborative service recommendation approach based on user's trust and domain-specific expertise. *Future Generation Computer Systems*, *80*, 355–367.

Kang, K. C. & Lee, H. (2013). Variability modeling. In *Systems and software variability management* (pp. 25–42). Heidelberg: Springer.

Kapitsaki, G. M., Prezerakos, G. N., Tselikas, N. D. & Venieris, I. S. (2009). Context-aware service engineering: A survey. *Journal of Systems and Software*, *82*(8), 1285–1297.

Karastoyanova, D. & Leymann, F. (2009). Bpel'n'aspects: Adapting service orchestration logic. In *Proceedings of the ieee international conference on web services* (pp. 222–229). IEEE Computer Society.

Karlsson, J. & Ryan, K. (1997). A cost-value approach for prioritizing requirements. *IEEE Software*, *14*(5), 67–74.

Kazhamiakin, R., Benbernou, S., Baresi, L., Plebani, P., Uhlig, M. & Barais, O. (2010). Adaptation of service-based systems. In *Service research challenges and solutions for the future internet* (pp. 117–156). Berlin: Springer.

Kazhamiakin, R., Pistore, M. & Zengin, A. (2009). Cross-layer adaptation and monitoring of service-based applications. In *Service-oriented computing. icsoc/servicewave 2009 workshops* (pp. 325–334). Berlin: Springer.

Kell, S. (2008). A survey of practical software adaptation techniques. *Journal of Universal Computer Science*, *14*(13), 2110–2157.

Kephart, J. O. & Chess, D. M. (2003). The vision of autonomic computing. *IEEE Computer*(1), 41–50.

Khoshkbarforoushha, A., Jamshidi, P., Gholami, M. F., Wang, L. & Ranjan, R. (2016). Metrics for bpel process reusability analysis in a workflow system. *IEEE Systems Journal*, *10*(1), 36–45.

Khoshkbarforoushha, A., Jamshidi, P., Nikravesh, A., Khoshnevis, S. & Shams, F. (2009). A metric for measuring bpel process context-independency. In *Proceedings of the ieee international conference on service oriented computing and applications* (pp. 1–8). IEEE Computer Society.

Kim, D., Park, C., Oh, J. & Yu, H. (2017). Deep hybrid recommender systems via exploiting document context and statistics of items. *Information Sciences*, *417*, 72–87.

Kindberg, T., Barton, J., Morgan, J., Becker, G., Caswell, D., Debaty, P., ... Spasojevic, M. (2002). People, places, things: Web presence for the real world. *Mobile Networks and Applications*, *7*(5), 365–376.

Kitchenham, B., Pfleeger, S. L. & Fenton, N. (1995). Towards a framework for software measurement validation. *IEEE Transactions on Software Engineering*, *21*(12), 929–944.

Ko, I.-Y., Ko, H.-G., Molina, A. J. & Kwon, J.-H. (2016). Soiot: Toward a user-centric iot-based service framework. *ACM Transactions on Internet Technology*, *16*(2),

8.

Kolos, L., van Eck, P. & Wieringa, R. J. (2006). *A survey of requirements engineering methods for pervasive services* (Tech. Rep.). Enschede, Netherlands: University of Twente.

Kolos-Mazuryk, L., Poulisse, G.-J. & van Eck, P. (2005). Requirements engineering for pervasive services. In *Proceedings of the 2nd workshop on building software for pervasive computing*. University of Twente.

Kongdenfha, W., Saint-Paul, R., Benatallah, B. & Casati, F. (2006). An aspect-oriented framework for service adaptation. In *Proceedings of the 4th international conference on service-oriented computing* (pp. 15–26). Springer.

Koning, M., Sun, C.-a., Sinnema, M. & Avgeriou, P. (2009). Vxbpel: Supporting variability for web services in bpel. *Information and Software Technology*, *51*(2), 258–269.

Koren, Y., Bell, R. & Volinsky, C. (2009). Matrix factorization techniques for recommender systems. *IEEE Computer*, *42*(8), 30–37.

Kostavelis, I., Giakoumis, D., Malasiotis, S. & Tzovaras, D. (2015). Ramcip: towards a robotic assistant to support elderly with mild cognitive impairments at home. In *Proceedings of the 5th international symposium on pervasive computing paradigms for mental health* (pp. 186–195). Springer.

Kritikos, K. & Plexousakis, D. (2009). Requirements for qos-based web service description and discovery. *IEEE Transactions on Services Computing*, *2*(4), 320–337.

Krogstie, J. (2001). Requirement engineering for mobile information systems. In *Proceedings of the 7th international workshop on requirements engineering: Foundations for software quality* (p. 221-226).

Kruchten, P. (2004). *The rational unified process: an introduction*. Addison-Wesley Professional.

Krupitzer, C., Roth, F. M., VanSyckel, S., Schiele, G. & Becker, C. (2015). A survey on engineering approaches for self-adaptive systems. *Pervasive and Mobile Computing*, *17*, 184–206.

Kurdija, A. S., Silic, M., Delac, G. & Vladimir, K. (2019). Fast multi-criteria service selection for multi-user composite applications. *IEEE Transactions on Services Computing*, *DOI: 10.1109/TSC.2019.2925614*.

Lane, S., Gu, Q., Lago, P. & Richardson, I. (2010). *Adaptation of service based applications: a maintenance process* (Tech. Rep.). University of Limerick, Ireland: Lero - The Irish Software Engineering Research Centre.

Lane, S., Gu, Q., Lago, P. & Richardson, I. (2014). Towards a framework for the development of adaptable service-based applications. *Service Oriented Computing and Applications*, *8*(3), 239–257.

Lapouchnian, A. (2005). *Goal-oriented requirements engineering: An overview of the current research* (Tech. Rep.). Ontario, Canada: University of Toronto.

Lapouchnian, A. & Mylopoulos, J. (2009). Modeling domain variability in requirements engineering with contexts. In *Proceedings of the 28th international conference on conceptual modeling* (pp. 115–130). Springer.

Lapouchnian, A. & Mylopoulos, J. (2011). Capturing contextual variability in i* models. In *Proceedings of the 5th international i* workshop (ceur)* (pp. 96–101). CEUR-WS.

Lazovik, A., Aiello, M. & Papazoglou, M. (2004). Associating assertions with business processes and monitoring their execution. In *Proceedings of the 2nd international conference on service oriented computing* (pp. 94–104). ACM.

Lécue, F. & Delteil, A. (2007). Making the difference in semantic web service composition. In *Proceedings of the 22nd national conference on artificial intelligence – volume 2* (pp. 1383–1388). AAAI Press.

Lee, J.-N., Huynh, M. Q., Kwok, R. C.-W. & Pi, S.-M. (2003). It outsourcing evolution–: past, present, and future. *Communications of the ACM*, *46*(5), 84–89.

Lehman, M. M. (1980). Programs, life cycles, and laws of software evolution. *Proceedings of the IEEE*, *68*(9), 1060–1076.

Lemmens, R., Wytzisk, A., By, R., Granell, C., Gould, M. & Van Oosterom, P. (2006). Integrating semantic and syntactic descriptions to chain geographic services. *IEEE Internet Computing*, *10*(5), 42–52.

Lemos, A. L., Daniel, F. & Benatallah, B. (2016). Web service composition: a survey of techniques and tools. *ACM Computing Surveys*, *48*(3), 33.

Lenhard, J. (2014). Towards quantifying the adaptability of executable bpmn processes. In *Proceedings of the 6th central-european workshop on services and their composition (zeus)* (pp. 34–41). CEUR-WS.

Lenhard, J., Geiger, M. & Wirtz, G. (2015). On the measurement of design-time adaptability for process-based systems. In *Proceedings of the ieee symposium on service-oriented system engineering* (pp. 1–10). IEEE Computer Society.

Leone, N. & Ricca, F. (2015). Answer set programming: a tour from the basics to advanced development tools and industrial applications. In *Reasoning web international summer school* (pp. 308–326). Berlin: Springer.

Letier, E. & Van Lamsweerde, A. (2004). Reasoning about partial goal satisfaction for requirements and design engineering. In *Proceedings of the 12th international symposium on foundations of software engineering* (Vol. 29, pp. 53–62). ACM.

Li, H., Liu, J., Cao, B., Tang, M., Liu, X. & Li, B. (2017). Integrating tag, topic, co-occurrence, and popularity to recommend web apis for mashup creation. In *Proceedings of the ieee international conference on services computing* (pp. 84–91). IEEE Computer Society.

Li, L., Wang, Y. & Lim, E.-P. (2009). Trust-oriented composite service selection and discovery. In *Proceedings of the 7th international joint conference, icsoc-servicewave 2009* (pp. 50–67). Springer.

Li, Y., Hu, J., Zhai, C. & Chen, Y. (2010). Improving one-class collaborative filtering by incorporating rich user information. In *Proceedings of the 19th international conference on information and knowledge management* (pp. 959–968). ACM.

Liaskos, S., Khan, S. M., Litoiu, M., Jungblut, M. D., Rogozhkin, V. & Mylopoulos, J. (2012). Behavioral adaptation of information systems through goal models. *Information Systems*, *37*(8), 767–783.

Liaskos, S., Lapouchnian, A., Wang, Y., Yu, Y. & Easterbrook, S. (2005). Configuring

common personal software: a requirements-driven approach. In *Proceedings of the 13th ieee international conference on requirements engineering* (pp. 9–18). IEEE Computer Society.

Liaskos, S., Lapouchnian, A., Yu, Y., Yu, E. & Mylopoulos, J. (2006a). On goal-based variability acquisition and analysis. In *Proceedings of the 14th ieee international requirements engineering conference* (pp. 79–88). IEEE Computer Society.

Liaskos, S., Lapouchnian, A., Yu, Y., Yu, E. & Mylopoulos, J. (2006b). On goal-based variability acquisition and analysis. In *Proceedings of the 14th ieee international conference on requirements engineering* (pp. 79–88). IEEE Computer Society.

Liaskos, S., McIlraith, S. A., Sohrabi, S. & Mylopoulos, J. (2011). Representing and reasoning about preferences in requirements engineering. *Requirements Engineering*, *16*(3), 227.

Liaskos, S. & Mylopoulos, J. (2010). On temporally annotating goal models. In *Proceedings of the 4th international i* workshop* (pp. 62–66). CEUR-WS.

Limam, N. & Boutaba, R. (2010). Assessing software service quality and trustworthiness at selection time. *IEEE Transactions on Software Engineering*, *36*(4), 559–574.

Linden, G., Smith, B. & York, J. (2003). Amazon.com recommendations: item-to-item collaborative filtering. *IEEE Internet Computing*(1), 76–80.

Liu, Y., Ngu, A. H. & Zeng, L. Z. (2004). Qos computation and policing in dynamic web service selection. In *Proceedings of the 13th international world wide web conference* (pp. 66–73). ACM.

Liu, Z.-Z., Song, C., Chu, D.-H., Hou, Z.-W. & Peng, W.-P. (2017). An approach for multipath cloud manufacturing services dynamic composition. *International Journal of Intelligent Systems*, *32*(4), 371–393.

Lloyd, E. A. & Gould, S. J. (1993). Species selection on variability. *Proceedings of the National Academy of Sciences*, *90*(2), 595–599.

Ludwig, H. & Petrie, C. (2006). Session summary – cross cutting concerns". In F. Cubera, B. J. Krämer & M. P. Papazoglou (Eds.), *Service oriented computing (soc).* Internationales Begegnungs- und Forschungszentrum für Informatik, Schloss Dagstuhl, Germany.

Ly, L. T., Rinderle, S. & Dadam, P. (2008). Integration and verification of semantic constraints in adaptive process management systems. *Data & Knowledge Engineering*, *64*(1), 3–23.

MacKenzie, C. M., Laskey, K., McCabe, F., Brown, P. F., Metz, R. & Hamilton, B. A. (2006). Reference model for service oriented architecture 1.0 [Computer software manual]. OASIS standard. (available at: https://www.oasis-open.org/committees/download.php/19679/soa-rm-cs.pdf)

Mahdavi-Hezavehi, S., Galster, M. & Avgeriou, P. (2013). Variability in quality attributes of service-based software systems: A systematic literature review. *Information and Software Technology*, *55*(2), 320–343.

Manes, A. T. (2001). Enabling open, interoperable, and smart web services [Computer software manual]. Sun Microsystems, Inc. (available at: https://www.w3.org/2001/03/WSWS-popa/paper29)

Mantere, T. & Alander, J. T. (2005). Evolutionary software engineering, a review. *Applied Soft Computing*, *5*(3), 315–331.

Martin, D., Paolucci, M., McIlraith, S., Burstein, M., McDermott, D., McGuinness, D., ... Sycara, K. (2004). Bringing semantics to web services: the owl-s approach. In *Proceedings of the 1st international workshop on semantic web services and web process composition* (pp. 26–42). Springer.

Masciadri, L. & Raibulet, C. (2009). Frameworks for the development of adaptive systems: evaluation of their adaptability feature through software metrics. In *Proceedings of the 4th international conference on software engineering advances* (pp. 309–312). IEEE Computer Society.

Mathew, S. S., Atif, Y., Sheng, Q. Z. & Maamar, Z. (2013). The web of things-challenges and enabling technologies. In *Internet of things and inter-cooperative computational technologies for collective intelligence* (pp. 1–23). Heidelberg: Springer.

Maximilien, E. M. & Singh, M. P. (2004). A framework and ontology for dynamic web services selection. *IEEE Internet Computing*, *8*(5), 84–93.

McIlraith, S. A., Son, T. C. & Zeng, H. (2001). Semantic web services. *IEEE Intelligent Systems*, *16*(2), 46–53.

McKinley, P. K., Sadjadi, S. M., Kasten, E. P. & Cheng, B. H. (2004). Composing adaptive software. *IEEE Computer*, *37*(7), 56–64.

Medvidovic, N. (2017). Foreword by nenad medvidovic: behold the golden age of software architecture. In I. Mistrik, N. Ali, R. Kazman, J. Grundy & B. Schmerl (Eds.), *Managing trade-offs in adaptable software architectures* (p. xxi-xxiii). Boston: Morgan Kaufmann.

Menasce, D. A., Gomaa, H. & Sousa, J. P. (2011). Sassy: A framework for self-architecting service-oriented systems. *IEEE software*, *28*(6), 78–85.

Mens, K., Capilla, R., Cardozo, N. & Dumas, B. (2016). A taxonomy of context-aware software variability approaches. In *Proceedings of the 15th international conference on modularity* (pp. 119–124). ACM.

Metzger, A. & Pohl, K. (2014). Software product line engineering and variability management: achievements and challenges. In *Proceedings of the on future of software engineering* (pp. 70–84). ACM.

Metzger, A., Pohl, K., Papazoglou, M., Di Nitto, E., Marconi, A., Karastoyanova, D., ... Bucchiarone, A. (2012). Research challenges on adaptive software and services in the future internet: towards an s-cube research roadmap. In *Proceedings of the 1st international workshop on european software services and systems research: Results and challenges* (pp. 1–7). IEEE.

Meyer, S., Ruppen, A. & Magerkurth, C. (2013). Internet of things-aware process modeling: integrating iot devices as business process resources. In *Proceedings of the 25th international conference on advanced information systems engineering* (pp. 84–98). Springer.

Miorandi, D., Sicari, S., De Pellegrini, F. & Chlamtac, I. (2012). Internet of things: Vision, applications and research challenges. *Ad hoc networks*, *10*(7), 1497–1516.

Mirandola, R., Perez-Palacin, D., Scandurra, P., Brignoli, M. & Zonca, A. (2015).

Business process adaptability metrics for qos-based service compositions. In *European conference on service-oriented and cloud computing* (pp. 110–124). Springer.

Mistrik, I., Ali, N., Kazman, R., Grundy, J. & Schmerl, B. (2017). *Managing trade-offs in adaptable software architectures*. Morgan Kaufmann.

Modafferi, S., Mussi, E. & Pernici, B. (2006). Sh-bpel: a self-healing plug-in for ws-bpel engines. In *Proceedings of the 1st workshop on middleware for service oriented computing* (pp. 48–53). ACM.

Morandini, M., Penserini, L., Perini, A. & Marchetto, A. (2017). Engineering requirements for adaptive systems. *Requirements Engineering*, *22*(1), 77–103.

Morin, B., Barais, O., Jézéquel, J.-M., Fleurey, F. & Solberg, A. (2009). Models@ run.time to support dynamic adaptation. *Computer*, *42*(10), 44–51.

Mosincat, A. & Binder, W. (2011). Automated maintenance of service compositions with sla violation detection and dynamic binding. *International Journal on Software Tools for Technology Transfer*, *13*(2), 167–179.

Motahari Nezhad, H. R., Benatallah, B., Martens, A., Curbera, F. & Casati, F. (2007). Semi-automated adaptation of service interactions. In *Proceedings of the 16th international conference on world wide web* (pp. 993–1002). ACM.

Mousa, A., Bentahar, J. & Alam, O. (2019). Context-aware composite saas using feature model. *Future Generation Computer Systems*, *99*, 376–390.

Muñoz, J. & Pelechano, V. (2006). Applying software factories to pervasive systems: a platform specific framework. In *Proceedings of the 8th international conference on enterprise information systems* (pp. 337–342). Springer.

Munoz, J., Valderas, P., Pelechano, V. & Pastor, O. (2006). Requirements engineering for pervasive systems. a transformational approach. In *Proceedings of the 14th ieee international requirements engineering conference* (pp. 351–352). IEEE Computer Society.

Murguzur, A., Intxausti, K., Urbieta, A., Trujillo, S. & Sagardui, G. (2014). Process flexibility in service orchestration: A systematic literature review. *International Journal of Cooperative Information Systems*, *23*(03), 1430001.

Mylopoulos, J., Chung, L., Liao, S., Wang, H. & Yu, E. (2001). Exploring alternatives during requirements analysis. *IEEE Software*, *18*(1), 92–96.

Mylopoulos, J., Chung, L. & Nixon, B. (1992). Representing and using nonfunctional requirements: A process-oriented approach. *IEEE Transactions on Software Engineering*, *18*(6), 483–497.

Nagrath, I. (2006). *Control systems engineering*. New Age International.

Naveed, N., Gottron, T., Kunegis, J. & Alhadi, A. C. (2011). Searching microblogs: coping with sparsity and document quality. In *Proceedings of the 20th international conference on information and knowledge management* (pp. 183–188). ACM.

Negri, P. P., Souza, V. E. S., de Castro Leal, A. L., de Almeida Falbo, R. & Guizzardi, G. (2017). Towards an ontology of goal-oriented requirements. In *Proceedings of the 20th conferencia iberoamericana en software engineering* (pp. 469–482). CIbSE.

Nguyen, C. M., Sebastiani, R., Giorgini, P. & Mylopoulos, J. (2018). Multi-objective reasoning with constrained goal models. *Requirements Engineering*, *23*(2), 189–225.

Nguyen, T., Colman, A., Talib, M. A. & Han, J. (2011). Managing service variability: state of the art and open issues. In *Proceedings of the 5th workshop on variability modeling of software-intensive systems* (pp. 165–173). ACM.

Nikravesh, A., Choffnes, D. R., Katz-Bassett, E., Mao, Z. M. & Welsh, M. (2014). Mobile network performance from user devices: A longitudinal, multidimensional analysis. In *Proceedings of the 15th international conference on passive and active network measurement* (pp. 12–22). Springer.

Noor, T. H., Sheng, Q. Z., Ngu, A. H. & Dustdar, S. (2014). Analysis of web-scale cloud services. *IEEE Internet Computing*, *18*(4), 55–61.

Oppenheim, D., Varshney, L. & Chee, Y.-M. (2014). Work as a service. In *Advanced web services* (p. 409-430). New York: Springer.

Oreizy, P., Gorlick, M. M., Taylor, R. N., Heimhigner, D., Johnson, G., Medvidovic, N., ... Wolf, A. L. (1999). An architecture-based approach to self-adaptive software. *IEEE Intelligent Systems and their Applications*, *14*(3), 54–62.

Oster, Z. J., Santhanam, G. R. & Basu, S. (2015). Scalable modeling and analysis of requirements preferences: A qualitative approach using ci-nets. In *Proceedings of the 23rd ieee international requirements engineering conference* (pp. 214–219). IEEE Computer Society.

Paik, H.-y., Lemos, A. L., Barukh, M. C., Benatallah, B. & Natarajan, A. (2017). *Web service implementation and composition techniques*. Cham, Switzerland: Springer.

Pan, R., Zhou, Y., Cao, B., Liu, N. N., Lukose, R., Scholz, M. & Yang, Q. (2008). One-class collaborative filtering. In *Proceedings of the 8th ieee international conference on data mining* (pp. 502–511).

Papazoglou, M. P. (2003). Service-oriented computing: Concepts, characteristics and directions. In *Proceedings of the 4th international conference on web information systems engineering* (pp. 3–12). IEEE Computer Society.

Papazoglou, M. P. & Van Den Heuvel, W.-J. (2006). Service-oriented design and development methodology. *International Journal of Web Engineering and Technology*, *2*(4), 412–442.

Papazoglou, M. P. & Van Den Heuvel, W.-J. (2007). Service oriented architectures: approaches, technologies and research issues. *The VLDB Journal*, *16*(3), 389–415.

Parhi, M., Pattanayak, B. K. & Patra, M. R. (2018). A multi-agent-based framework for cloud service discovery and selection using ontology. *Service Oriented Computing and Applications*, *12*(2), 137–154.

Paschke, A. & Teymourian, K. (2009). Rule based business process execution with bpel+. In *Proceedings of the 5th international conference on semantic systems* (pp. 588–601). Verlag der Technischen Universität Graz.

Pastore, S. (2008). The service discovery methods issue: A web services uddi specification framework integrated in a grid environment. *Journal of Network and*

*Computer Applications*, *31*(2), 93–107.

Paucar, L. H. G., Bencomo, N. & Yuen, K. K. F. (2019). Arrow: automatic runtime reappraisal of weights for self-adaptation. In *Proceedings of the 34th acm/sigapp symposium on applied computing* (pp. 1584–1591). ACM.

Pautasso, C. (2009). Restful web service composition with bpel for rest. *Data & Knowledge Engineering*, *68*(9), 851–866.

Pautasso, C., Zimmermann, O. & Leymann, F. (2008). Restful web services vs. big web services: making the right architectural decision. In *Proceedings of the 17th international conference on world wide web* (pp. 805–814). ACM.

Peltz, C. (2003). Web services orchestration and choreography. *IEEE Computer*, *36*(10), 46–52.

Perera, C., Zaslavsky, A., Christen, P. & Georgakopoulos, D. (2014). Context aware computing for the internet of things: a survey. *IEEE Communications Surveys & Tutorials*, *16*(1), 414–454.

Perez-Palacin, D., Mirandola, R. & Merseguer, J. (2014). On the relationships between qos and software adaptability at the architectural level. *Journal of Systems and Software*, *87*, 1–17.

Pernici, B. & Rosati, A. M. (2007). Automatic learning of repair strategies for web services. In *Proceedings of the 5th ieee european conference on web services* (pp. 119–128). IEEE Computer Society.

Petrie, C. (2016). *Web service composition*. Springer.

Raibulet, C. & Masciadri, L. (2009). Evaluation of dynamic adaptivity through metrics: an achievable target? In *Proceedings of the wicsa & european conference on software architecture* (pp. 341–344). IEEE Computer Society.

Ralyte, J. (2012). Viewpoints and issues in requirements engineering for services. In *Proceedings of the 36th ieee annual computer software and applications conference workshops* (pp. 341–346). IEEE Computer Society.

Rao, J. & Su, X. (2004). A survey of automated web service composition methods. In *Proceedings of the 1st international workshop on semantic web services and web process composition* (pp. 43–54). Springer.

Reinecke, P., Wolter, K. & Van Moorsel, A. (2010). Evaluating the adaptivity of computing systems. *Performance Evaluation*, *67*(8), 676–693.

Rendle, S., Freudenthaler, C., Gantner, Z. & Schmidt-Thieme, L. (2009). Bpr: Bayesian personalized ranking from implicit feedback. In *Proceedings of the 25th conference on uncertainty in artificial intelligence* (pp. 452–461). AUAI Press.

Ricca, F. (2003). A java wrapper for dlv. In *Proceedings of the 2nd international answer set programming workshop*. CEUR-WS.org.

Robinson, W. N. (1989). Integrating multiple specifications using domain goals. In *Proceedings of the 5th international workshop on software specification and design* (pp. 219 – 226). ACM.

Robinson, W. N. (1990). Negotiation behavior during requirement specification. In *Proceedings of the 12th international conference on software engineering* (pp. 268–276). IEEE Computer Society.

Rodrigues, G. N., Tavares, C. J., Watanabe, N., Alves, C. & Ali, R. (2018). A persona-based modelling for contextual requirements. In *Proceedings of the 24th international working conference on requirements engineering: Foundation for software quality* (pp. 352–368). Springer.

Rodriguez-Mier, P., Pedrinaci, C., Lama, M. & Mucientes, M. (2015). An integrated semantic web service discovery and composition framework. *IEEE Transactions on Services Computing*, *9*(4), 537–550.

Rolland, C. & Salinesi, C. (2005). Modeling goals and reasoning with them. In *Engineering and managing software requirements* (pp. 189–217). Berlin, Heidelberg: Springer.

Rong, W. & Liu, K. (2010). A survey of context aware web service discovery: from user's perspective. In *Proceedings of the 5th ieee international symposium on service oriented system engineering* (pp. 15–22). IEEE Computer Society.

Rosenberg, F. & Dustdar, S. (2005). Business rules integration in bpel – a service-oriented approach. In *Proceedings of the 7th ieee international conference on e-commerce technology* (pp. 476–479). IEEE Computer Society.

Rutten, E., Marchand, N. & Simon, D. (2017). Feedback control as mape-k loop in autonomic computing. In *Software engineering for self-adaptive systems iii. assurances* (pp. 349–373). Cham, Switzerland: Springer.

Ryu, D., Lee, K. & Baik, J. (2018). Location-based web service qos prediction via preference propagation to address cold start problem. *IEEE Transactions on Services Computing*, *DOI: 10.1109/TSC.2018.2821686*.

Saaty, T. L. (2008). Decision making with the analytic hierarchy process. *International Journal of Services Sciences*, *1*(1), 83–98.

Sadat, H. & Ghorbani, A. A. (2004). On the evaluation of adaptive web systems. In *Proceedings of the 2nd international workshop on web-based support systems* (pp. 127–136). Halifax, Nova Scotia: Saint Mary's University.

Salakhutdinov, R. & Mnih, A. (2007). Probabilistic matrix factorization. In *Proceedings of the 20th international conference on neural information processing systems* (pp. 1257–1264). Curran Associates Inc.

Salifu, M., Yu, Y. & Nuseibeh, B. (2007). Specifying monitoring and switching problems in context. In *Proceedings of the 15th ieee international requirements engineering conference* (pp. 211–220). IEEE Computer Society.

Salton, G. & Buckley, C. (1988). Term-weighting approaches in automatic text retrieval. *Information Processing & Management*, *24*(5), 513–523.

Sam, Y., Boucelma, O. & Hacid, M.-S. (2006). Web services customization: a composition-based approach. In *Proceedings of the 6th international conference on web engineering* (pp. 25–31). ACM.

Sarwar, B., Karypis, G., Konstan, J. & Riedl, J. (2001). Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on world wide web* (pp. 285–295). ACM.

Sawyer, P., Bencomo, N., Whittle, J., Letier, E. & Finkelstein, A. (2010). Requirements-aware systems: a research agenda for re for self-adaptive systems. In *Proceedings of the 18th ieee international requirements engineering conference* (pp. 95–103).

IEEE Computer Society.

Schall, D., Truong, H.-L. & Dustdar, S. (2008). Unifying human and software services in web-scale collaborations. *IEEE Internet Computing*, *12*(3), 62–68.

Schmerl, B., Kazman, R., Ali, N., Grundy, J. & Mistrik, I. (2017). Managing trade-offs in adaptable software architectures. In *Managing trade-offs in adaptable software architectures* (pp. 1–13). Amsterdam: Elsevier.

Schmidt, M.-T., Hutchison, B., Lambros, P. & Phippen, R. (2005). The enterprise service bus: making service-oriented architecture real. *IBM Systems Journal*, *44*(4), 781–797.

Schröder, G., Thiele, M. & Lehner, W. (2011). Setting goals and choosing metrics for recommender system evaluations. In *Proceedings of the ucersti-2 workshop of the 5th international conference on recommender systems* (pp. 78–85). ACM.

Sebastiani, R., Giorgini, P. & Mylopoulos, J. (2004). Simple and minimum-cost satisfiability for goal models. In *Proceedings of the 16th international conference on advanced information systems engineering* (pp. 20–35). Springer.

Shao, L., Zhang, J., Wei, Y., Zhao, J., Xie, B. & Mei, H. (2007). Personalized qos prediction for web services via collaborative filtering. In *Proceedings of the ieee international conference on web services* (pp. 439–446). IEEE Computer Society.

Shen, J., Grossmann, G., Yang, Y., Stumptner, M., Schrefl, M. & Reiter, T. (2007). Analysis of business process integration in web service context. *Future Generation Computer Systems*, *23*(3), 283–294.

Shen, W., Hao, Q., Wang, S., Li, Y. & Ghenniwa, H. (2007). An agent-based service-oriented integration architecture for collaborative intelligent manufacturing. *Robotics and Computer-Integrated Manufacturing*, *23*(3), 315–325.

Sheng, Q. Z. & Benatallah, B. (2005). Contextuml: a uml-based modeling language for model-driven development of context-aware web services. In *Proceedings of the international conference on mobile business* (pp. 206–212). IEEE Computer Society.

Sheng, Q. Z., Qiao, X., Vasilakos, A. V., Szabo, C., Bourne, S. & Xu, X. (2014). Web services composition: A decade's overview. *Information Sciences*, *280*, 218–238.

Shi, Y., Larson, M. & Hanjalic, A. (2014). Collaborative filtering beyond the user-item matrix: A survey of the state of the art and future challenges. *ACM Computing Surveys*, *47*(1), 3.

Sievert, C. & Shirley, K. (2014). Ldavis: A method for visualizing and interpreting topics. In *Proceedings of the workshop on interactive language learning, visualization, and interfaces* (pp. 63–70). Association for Computational Linguistics.

Siljee, J., Bosloper, I., Nijhuis, J. & Hammer, D. (2005). Dysoa: making service systems self-adaptive. In *Proceedings of the international conference on service-oriented computing* (pp. 255–268). Springer.

Sim, K. M. (2011). Agent-based cloud computing. *IEEE Transactions on Services Computing*, *5*(4), 564–577.

Somu, N., MR, G. R., Kirthivasan, K. & VS, S. S. (2018). A trust centric optimal service ranking approach for cloud service selection. *Future Generation Computer*

*Systems*, *86*, 234–252.

Song, S. & Lee, S.-W. (2013). A goal-driven approach for adaptive service composition using planning. *Mathematical and Computer Modelling*, *58*(1-2), 261–273.

Srebro, N. & Jaakkola, T. (2003). Weighted low-rank approximations. In *Proceedings of the 20th international conference on machine learning* (pp. 720–727). AAAI.

Stefanidis, K., Pitoura, E. & Vassiliadis, P. (2011). Managing contextual preferences. *Information Systems*, *36*(8), 1158–1180.

Steyvers, M. & Griffiths, T. (2006). Probabilistic topic models. In T. Landauer, D. McNamara, S. Dennis & W. Kintsch (Eds.), *Latent semantic analysis: A road to meaning.* Hillsdale, NJ: Laurence Erlbaum.

Subramanian, N. & Chung, L. (2001a). Metrics for software adaptability. In *Proceedings of the software quality management (sqm) conference.*

Subramanian, N. & Chung, L. (2001b). Software architecture adaptability: an nfr approach. In *Proceedings of the 4th international workshop on principles of software evolution* (pp. 52–61). ACM.

Sun, C.-a., Rossing, R., Sinnema, M., Bulanov, P. & Aiello, M. (2010). Modeling and managing the variability of web service-based systems. *Journal of Systems and Software*, *83*(3), 502–516.

Sutcliffe, A., Fickas, S. & Sohlberg, M. M. (2005). Personal and contextual requirements engineering. In *Proceedings of the 13th ieee international conference on requirements engineering* (pp. 19–28). IEEE Computer Society.

Sutcliffe, A. & Sawyer, P. (2013). Modeling personalized adaptive systems. In *Proceedings of the 25th international conference on advanced information systems engineering* (pp. 178–192). Springer.

Szvetits, M. & Zdun, U. (2016). Systematic literature review of the objectives, techniques, kinds, and architectures of models at runtime. *Software & Systems Modeling*, *15*(1), 31–69.

Tahamtan, A., Beheshti, S. A., Anjomshoaa, A. & Tjoa, A. M. (2012). A cloud repository and discovery framework based on a unified business and cloud service ontology. In *Proceedings of the 8th ieee world congress on services* (pp. 203–210). IEEE Computer Society.

Tan, W., Fan, Y., Ghoneim, A., Hossain, M. A. & Dustdar, S. (2016). From the service-oriented architecture to the web api economy. *IEEE Internet Computing*, *20*(4), 64-68.

Tang, M., Zhang, T., Liu, J. & Chen, J. (2015). Cloud service qos prediction via exploiting collaborative filtering and location-based data smoothing. *Concurrency and Computation: Practice and Experience*, *27*(18), 5826–5839.

Tekinerdogan, B. & Aksit, M. (1996). Adaptability in object-oriented software development: Workshop report. In *Proceedings of the 10th annual european conference on object-oriented programming.* Springer.

Trainotti, M., Pistore, M., Calabrese, G., Zacco, G., Lucchese, G., Barbon, F., ... Traverso, P. (2005). Astro: supporting composition and execution of web services. In *Proceedings of the international conference on service-oriented computing* (pp. 495–501). Springer.

Tripathy, A. K. & Tripathy, P. K. (2018). Fuzzy qos requirement-aware dynamic service discovery and adaptation. *Applied Soft Computing*, *68*, 136–146.

Truong, H.-L. & Dustdar, S. (2009). A survey on context-aware web service systems. *International Journal of Web Information Systems*, *5*(1), 5–31.

Tsai, W.-T., Sun, X. & Balasooriya, J. (2010). Service-oriented cloud computing architecture. In *Proceedings of the 7th international conference on information technology: New generations* (pp. 684–689). IEEE.

Van Lamsweerde, A. (2001). Goal-oriented requirements engineering: a guided tour. In *Proceedings of the 5th ieee international symposium on requirements engineering* (pp. 249–262). IEEE Computer Society.

van Lamsweerde, A. (2009). Reasoning about alternative requirements options. In *Conceptual modeling: Foundations and applications* (pp. 380–397). Berlin, Heidelberg: Springer.

Van Lamsweerde, A., Darimont, R. & Letier, E. (1998). Managing conflicts in goal-driven requirements engineering. *IEEE Transactions on Software Engineering*, *24*(11), 908–926.

Van Lamsweerde, A. & Letier, E. (2000). Handling obstacles in goal-oriented requirements engineering. *IEEE Transactions on Software Engineering*, *26*(10), 978–1005.

Vasilecas, O., Kalibatiene, D. & Lavbič, D. (2016). Rule-and context-based dynamic business process modelling and simulation. *Journal of Systems and Software*, *122*, 1–15.

Verma, K., Gomadam, K., Sheth, A. P., Miller, J. & Wu, Z. (2005). *The meteor-s approach for configuring and executing dynamic web processes* (Tech. Rep.). Fairborn, Ohio: Wright State University.

von Rosing, M., White, S., Cummins, F. & de Man, H. (2015). Business process model and notation-bpmn. In *The complete business process handbook* (pp. 429–453). Amsterdam: Elsevier.

Wang, H., Wang, L., Yu, Q., Zheng, Z. & Yang, Z. (2018). A proactive approach based on online reliability prediction for adaptation of service-oriented systems. *Journal of Parallel and Distributed Computing*, *114*, 70–84.

Wang, J., Zhang, J., Hung, P. C., Li, Z., Liu, J. & He, K. (2011). Leveraging fragmental semantic data to enhance services discovery. In *Proceedings of the 13th ieee international conference on high performance computing and communications* (pp. 687–694). IEEE.

Wang, L., Shen, J. & Yong, J. (2012). A survey on bio-inspired algorithms for web service composition. In *Proceedings of the 16th ieee international conference on computer supported cooperative work in design* (pp. 569–574). IEEE.

Wang, Y. & Vassileva, J. (2007). Toward trust and reputation based web service selection: a survey. *International Transactions on Systems Science and Applications*, *3*(2), 118–132.

Wei, Y. & Blake, M. B. (2010). Service-oriented computing and cloud computing: Challenges and opportunities. *IEEE Internet Computing*, *14*(6), 72–75.

Weibelzahl, S. (2002). *Evaluation of adaptive systems* (Unpublished doctoral dissertation). University of Education Freiburg, Germany.

Weiser, M. (1993). Hot topics - ubiquitous computing. *IEEE Computer*, *26*(10), 71-72.

Weiser, M. (1999). The computer for the 21st century. *SIGMOBILE Mobile Computing and Communications Review*, *3*(3), 3–11.

Weng, J., Lim, E.-P., Jiang, J. & He, Q. (2010). Twitterrank: finding topic-sensitive influential twitterers. In *Proceedings of the 3rd international conference on web search and data mining* (pp. 261–270). ACM.

Weyns, D. (2019). Software engineering of self-adaptive systems. In *Handbook of software engineering* (pp. 399–443). Cham, Switzerland: Springer.

Wieringa, R. (2005). Requirements researchers: are we really doing research? *Requirements Engineering*, *10*(4), 304–306.

Williams, S. K., Battle, S. A. & Cuadrado, J. E. (2006). Protocol mediation for adaptation in semantic web services. In *Proceedings of the 3rd european semantic web conference* (pp. 635–649). Springer.

Wishart, R., Robinson, R., Indulska, J. & Jøsang, A. (2005). Superstringrep: reputation-enhanced service discovery. In *Proceedings of the 28th australasian conference on computer science-volume 38* (pp. 49–57). Australian Computer Society, Inc.

Wu, G., Wei, J., Qiao, X. & Li, L. (2007). A bayesian network based qos assessment model for web services. In *Proceedings of the ieee international conference on services computing* (pp. 498–505). IEEE Computer Society.

Wu, H., Yue, K., Li, B., Zhang, B. & Hsu, C.-H. (2018). Collaborative qos prediction with context-sensitive matrix factorization. *Future Generation Computer Systems*, *82*, 669–678.

Wu, X., Cheng, B. & Chen, J. (2017). Collaborative filtering service recommendation based on a novel similarity computation method. *IEEE Transactions on Services Computing*, *10*(3), 352–365.

Wu, Y. & Doshi, P. (2007). Regret-based decentralized adaptation of web processes with coordination constraints. In *Proceedings of the ieee international conference on services computing* (pp. 262–269). IEEE Computer Society.

Xu, Y., Yin, J., Deng, S., Xiong, N. N. & Huang, J. (2016). Context-aware qos prediction for web service recommendation and selection. *Expert Systems with Applications*, *53*, 75–86.

Yang, Z., Li, Z., Jin, Z. & Chen, Y. (2014). A systematic literature review of requirements modeling and analysis for self-adaptive systems. In *Proceedings of the 20th international working conference on requirements engineering: Foundation for software quality* (pp. 55–71). Springer.

Yao, L., Sheng, Q. Z., Ngu, A. H., Yu, J. & Segev, A. (2015). Unified collaborative and content-based web service recommendation. *IEEE Transactions on Services Computing*, *8*(3), 453–466.

Yao, L., Wang, X., Sheng, Q. Z., Benatallah, B. & Huang, C. (2018). Mashup recommendation by regularizing matrix factorization with api co-invocations. *IEEE Transactions on Services Computing*, *DOI: 10.1109/TSC.2018.2803171*.

Yen, J. & Tiao, W. A. (1997). A systematic tradeoff analysis for conflicting imprecise requirements. In *Proceedings of the 3rd ieee international symposium on requirements engineering* (pp. 87–96). IEEE Computer Society.

Yu, E. & Mylopoulos, J. (1998). Why goal-oriented requirements engineering. In *Proceedings of the 4th international workshop on requirements engineering: Foundations of software quality* (Vol. 15, pp. 15–22). Presses Universitaires de Namur.

Yu, E. S. (1997). Towards modelling and reasoning support for early-phase requirements engineering. In *Proceedings of the 3rd ieee international symposium on requirements engineering* (pp. 226–235). IEEE.

Yu, J., Sheng, Q. Z., Swee, J. K., Han, J., Liu, C. & Noor, T. H. (2015). Model-driven development of adaptive web service processes with aspects and rules. *Journal of Computer and System Sciences*, *81*(3), 533–552.

Yu, Q., Zheng, Z. & Wang, H. (2013). Trace norm regularized matrix factorization for service recommendation. In *Proceedings of the 20th ieee international conference on web services* (pp. 34–41). IEEE Computer Society.

Yu, T., Zhang, Y. & Lin, K.-J. (2007). Efficient algorithms for web services selection with end-to-end qos constraints. *ACM Transactions on the Web*, *1*(1), 6.

Yu, W. D., Radhakrishna, R. B., Pingali, S. & Kolluri, V. (2007). Modeling the measurements of qos requirements in web service systems. *Simulation*, *83*(1), 75–91.

Zeng, L., Benatallah, B., Dumas, M., Kalagnanam, J. & Sheng, Q. Z. (2003). Quality driven web services composition. In *Proceedings of the 12th international conference on world wide web* (pp. 411–421). ACM.

Zeng, L., Benatallah, B., Ngu, A. H., Dumas, M., Kalagnanam, J. & Chang, H. (2004). Qos-aware middleware for web services composition. *IEEE Transactions on Software Engineering*, *30*(5), 311–327.

Zhang, P., Jin, H., He, Z., Leung, H., Song, W. & Jiang, Y. (2018). Igs-wbsrm: A time-aware web service qos monitoring approach in dynamic environments. *Information and software technology*, *96*, 14–26.

Zhang, Q., Cheng, L. & Boutaba, R. (2010). Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications*, *1*(1), 7–18.

Zhang, Y., Wang, K., He, Q., Chen, F., Deng, S., Zheng, Z. & Yang, Y. (2019). Covering-based web service quality prediction via neighborhood-aware matrix factorization. *IEEE Transactions on Services Computing*, *DOI: 10.1109/TSC.2019.2891517*.

Zheng, Z., Ma, H., Lyu, M. R. & King, I. (2009). Wsrec: A collaborative filtering based web service recommender system. In *Proceedings of the ieee international conference on web services* (pp. 437–444). IEEE Computer Society.

Zheng, Z., Ma, H., Lyu, M. R. & King, I. (2010). Qos-aware web service recommendation by collaborative filtering. *IEEE Transactions on Services Computing*, *4*(2), 140–152.

Zheng, Z., Ma, H., Lyu, M. R. & King, I. (2013). Collaborative web service qos prediction via neighborhood integrated matrix factorization. *IEEE Transactions on Services Computing*, *6*(3), 289–299.

Zheng, Z., Zhang, Y. & Lyu, M. R. (2014). Investigating qos of real-world web services. *IEEE Transactions on Services Computing*, *7*(1), 32–39.

Zhong, Y., Fan, Y., Huang, K., Tan, W. & Zhang, J. (2015). Time-aware service recommendation for mashup creation. *IEEE Transactions on Services Computing*, *8*(3), 356–368.

Zhou, Q., Wu, H., Yue, K. & Hsu, C.-H. (2019). Spatio-temporal context-aware collaborative qos prediction. *Future Generation Computer Systems*, *100*, 46–57.

Zhou, Y. & Chen, T. (2017). *Software adaptation in an open environment: A software architecture perspective* (1st ed.). Auerbach Publications.

Zhu, X., Jing, X.-Y., Wu, D., He, Z., Cao, J., Yue, D. & Wang, L. (2018). Similarity-maintaining privacy preservation and location-aware low-rank matrix factorization for qos prediction based web service recommendation. *IEEE Transactions on Services Computing*, *DOI: 10.1109/TSC.2018.2839741*.

Zou, G., Xiang, Y., Gan, Y., Wang, D. & Liu, Z. (2009). An agent-based web service selection and ranking framework with qos. In *Proceedings of the 2nd ieee international conference on computer science and information technology* (pp. 37–42). IEEE.

Zuse, H. (1998). *A framework of software measurement*. Berlin, Germany: Walter de Gruyter & Co.