A Scalable Approach for Re-Configuring Evolving Industrial Control Systems

Roopak Sinha
Computer & Mathematical Sciences
Auckland University of Technology
New Zealand
Email: roopak.sinha@aut.ac.nz

Kenneth Johnson
Computer & Mathematical Sciences
Auckland University of Technology
New Zealand
Email: kenneth.johnson@aut.ac.nz

Radu Calinescu
Department of Computer Science
University of York
United Kingdom
Email: radu.calinescu@york.ac.uk

***Abstract*—We present a scalable approach to automatically re-configure evolving IEC 61499 systems for deployment onto an available set of resources. We capture system architecture and high-level configuration requirements formally, and use an efficient SMT-based constraint resolution to generate a valid system configuration. Any changes in the system architecture, configuration requirements, or resources are automatically translated into a minimal set of updated constraints, allowing a faster reconfiguration as compared to a monolithic approach where the whole system is re-configured. We show the feasibility of our approach by studying an airport baggage handling system developed using the IEC 61499 standard.**

I. Introduction

The IEC 61499 [1] standard allows designers to write complex distributed industrial control software as a network of reusable software components called *function blocks*. Some examples of systems developed using function blocks are airport baggage handling systems [2] (BHS), power grid systems [3], and factory automation systems [4]. Such complex function block systems are *manually* distributed over several hardware devices like *programmable logic controllers* (PLCs) [1] or cloud services [5], [6]. The manual step, called *system configuration* can prove to be a bottleneck when the software architecture or device availability changes [7], [8].

Many function block systems must be *downtimeless*, or remain operational for extended periods of time with zero downtime [9], [10], [11]. We must handle changes in system architecture, configuration requirements, or device availability without halting operation. Several approaches have targeted this *dynamic configuration* problem. In [10], a framework to manage changes in communication protocols, architectural change and internal changes within individual components is proposed. This framework can check that key functional requirements are met after changes are applied but requires user-provided configurations. Likewise, other existing approaches deal only with functional requirements [12], [13] and/or require user-specified (re-)configurations [12], [14], [15].

We focus on two key questions of dynamic configuration:

- *How can a function block system be automatically configured such that high-level configuration requirements are satisfied?*
- *How can a function block system automatically reconfigure itself according to changes in system architecture, device availability, and configuration requirements?*

Our solution is a framework that allows users to formally specify high-level configuration requirements, and an algorithm that identifies configurations that satisfy these requirements. The architecture re-configuration process presented in this paper is based on the incremental verification approaches described in [16], [17], [18]. The process is depicted in Fig. 1 and comprises four steps:

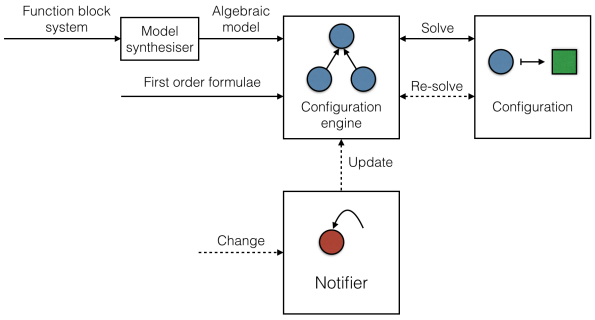


Fig. 1: Incremental re-configuration overview

1. *Algebraic model generation*: The *model synthesiser* accepts as input a function block system and generates an algebraic model of its architecture.
2. *Requirement specification*: Configuration requirements are specified as quantifier-free first order formulae and accepted as input into the configuration engine.
3. *Compositional resolution*: The *configuration engine* performs compositional resolution and computes a satisfiable configuration for the system. The actions associated with this step are depicted by solid lines in Fig. 1.
4. *Incremental re-resolution*: When a change in the system's architecture or requirements is detected, the *notifier* triggers an update to the algebraic model and associated first-order formulae. The configuration engine responds by carrying out incremental re-resolution whereby only the components affected by the change are re-configured. The actions associated with this step are depicted by dashed lines in Fig. 1.

The primary contributions of this paper are:

- a formal specification of the configuration problem for industrial control software as an SMT constraint problem,
- an application of the compositional SMT approach developed in [16] for configuring and an incremental reconfiguring method for function block systems, and
- an evaluation of our solution involving the re-configuration of a realistic airport baggage handling (BHS) system.

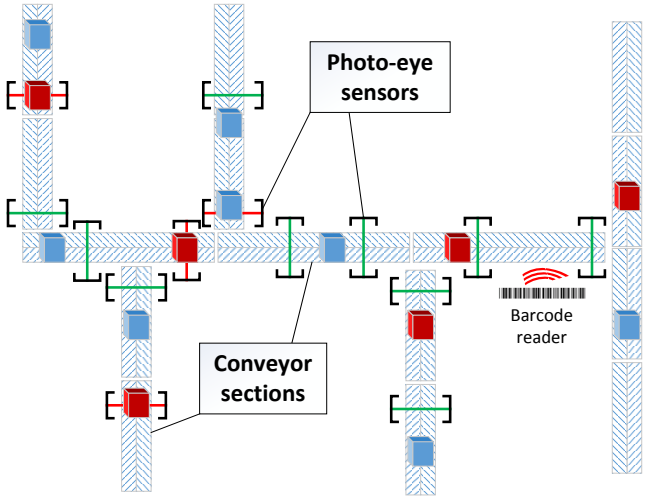


Fig. 2: A small baggage handling system

The rest of this paper is organised as follows. Sec. II describes the BHS case study and the extraction of its architectural model. Sec. III formulates an algebraic model for function block systems. Sec. IV shows how requirements are specified and formalised. Sec. V and Sec. VI provide details about automatically configuring and re-configuring function block systems. Sec. VII explores the scalability of our approach by analysing increasingly larger BHS systems. Finally, concluding remarks and future directions appear in Sec. VIII.

II. Case Study

Every large airport relies on complex *baggage handling system* (BHS) [2] to transport checked luggage from check-in points to specific zones where airport staff load the luggage onto air crafts. We use a simplified version of an airport BHS, shown in Fig. 2, to illustrate the key concepts introduced throughout the rest of this paper.

The BHS has four baggage entry points from check-in counters and two baggage exit points to airport gates. The system contains 15 conveyor sections to move baggage in predetermined directions, 14 photo-eye sensors to detect baggage at strategic points in the BHS, and a bar code reader to determine a bag's exit point. Each photo-eye sensor emits a light beam from a transmitter to a receiver. The sensor detects a bag when the emitted beam is broken and does not reach the receiver. The BHS presented here can be easily scaled up by introducing additional conveyors, photo-eye sensors and x-ray machines. In addition, the physical layout of the BHS is subject to change as airport facilities evolve.

A. Implementing the BHS using Function Blocks

The *IEC 61499* standard is an architectural framework that specifies control software for distributed automation systems in terms of reusable modules called *function blocks* [19], [1]. Fig. 3a uses this framework to describe the top-level BHS system as a function block *network* that contains two function blocks[1] called **ModelView** and **Mutex**. **ModelView** contains all individual conveyor and photo-eye controllers and *views* (to simulate the status of these components) in the system, while **Mutex** implements *mutual exclusion* logic which guarantees that merging luggage from two or more conveyors can never overlap. Blocks like **ModelView**, **Mutex**, and **Conveyors** (contained inside **ModelView**) are called *composite* function blocks as they contain (finitely nested) networks of function blocks. Networks are designed by manually inter-connecting the event and variable inputs and outputs of constituent blocks. As our approach is not affected by the interconnections between the blocks of a network, we do not discuss this aspect in detail. Interested readers will find details in [1].

In addition to composite blocks, a function block network can contain *basic* and *service-interface* function blocks. Both basic and service-interface blocks are *atomic* blocks and cannot be further broken down. The behaviour of basic blocks is modelled by finite state machines called *execution control charts*, while service-interface blocks contain platform-specific implementations of system services like network connections and timers. For illustration purposes, we assume that blocks **A** - **H** shown in Fig. 3a are atomic blocks. In reality, the BHS system contains 281 function blocks.

[1]Precisely, networks contain function block *instances* while we assume that all networked blocks are function blocks. This assumption does not affect the generality of our approach and is made purely to simplify the formalism.

B. Extracting the architecture of the BHS

Let FB denote the set of all function blocks. We can identify a subset $\{\mathbf{b}_1, \ldots, \mathbf{b}_n\} \subset FB$ as the set of *atomic* (basic or service-interface) blocks. A *composite* function block $\mathbf{cf} \in FB$ is composed of other function blocks. Mathematically, we define a compositional operation $comp : FB^m \rightarrow FB$ such that $\mathbf{cf} \equiv comp(\mathbf{f}_1, \ldots, \mathbf{f}_m)$ means that **cf** *contains* function blocks $\mathbf{f}_1, \ldots, \mathbf{f}_m$. Repeated application of the *comp* operation on basic function blocks enable more complex function blocks to be specified. In this way, we express the top-level system of the the baggage-handling case study shown in Fig. 3a as

$$\mathbf{BHS} \equiv comp(\mathbf{ModelView}, \mathbf{Mutex}) \quad (1)$$

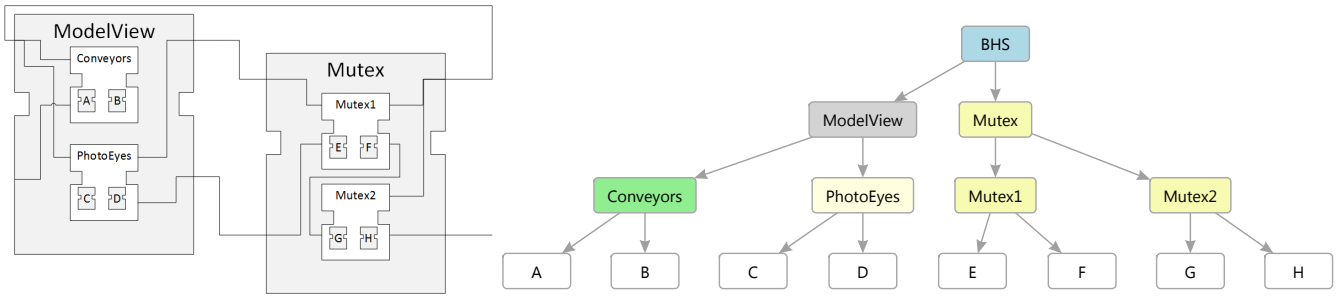
where $\mathbf{ModelView} \equiv comp(\mathbf{Conveyors}, \mathbf{PhotoEyes})$ and $\mathbf{Mutex} \equiv comp(\mathbf{Mutex1}, \mathbf{Mutex2})$. Each composite function block ultimately contains atomic blocks labelled **A** to **H**. Fig. 3b illustrates the hierarchical architecture of the BHS network expressed in (1). Nodes represent function blocks and edges denote compositional dependencies between nodes.

C. BHS Configuration Requirements

The IEC 61499 standard stipulates that the user must (manually) identify a **BHS** *configuration* which specifies a mapping of multiple copies of the basic function blocks **A** - **H** to computing resources residing in separate programmable-logic controllers (PLCs). PLCs are industrially-hardened computers with independent power supplies, networks and computing capacities and communications between function blocks executing on different PLCs are channelled via service-interface function blocks while intra-device or intra-resource communications use local mechanisms like pipes or shared memory.

We assume that **BHS** is to be distributed over PLCs numbered $1, \ldots, p$, where $p \geq 2$. Each PLC has a total of $q_i \geq 2$ resources for $1 \leq i \leq p$. Each resource can schedule and execute multiple function blocks. In accordance to standard practices in industrial system configuration to improving reliability, function blocks are copied and deployed multiple times across computing resources according to the following high-level requirements:

R1 All function blocks have between 1 and 5 copies.
R2 B has at least 3 copies, all of which must be distributed over exactly two different PLCs.



(a) Abstracted BHS function block network

(b) BHS architecture extracted from Fig. 3a

Fig. 3: Extracting the system architecture of a function block network

R3 D is deployed over four PLC resources.
R4 C has exactly 1 copy.
R5 The basic blocks of **Conveyors** are identically deployed.
R6 All instances of **Mutex2** are deployed on PLC 1.
R7 Each PLC 2 resource has a copy of **Mutex1**.

III. Algebraic Models of Function Blocks

The first step of the incremental re-resolution approach uses the *model synthesiser* to generate an algebraic *model* from an input system specified in the IEC 61499 standard. The algebraic model is used for incremental (re)-resolution.

Definition 3.1 (Function Block Models): Let Z be a set of variables. We define the model $Z_{\mathbf{f}} \subset Z$ of function block $\mathbf{f} \in FB$ by induction on the structure of function blocks.

Base case: $Z_{\mathbf{b}_i} \in \mathbb{P}(Z)$, for each basic block $\mathbf{b}_i$ in FB such that models are pairwise disjoint: $Z_{\mathbf{b}_i} \cap Z_{\mathbf{b}_j} = \emptyset$ for $i \neq j$.

Structural induction: $Z_{comp(\mathbf{f}_1,\ldots,\mathbf{f}_m)} = \cup_{i=1}^{m} Z_{\mathbf{f}_i}$, for the composition operation $comp$ and blocks $\mathbf{f}_1, \ldots, \mathbf{f}_m$ in FB.

For example, we model the basic function block **A** of the BHS case study introduced in Section II algebraically as the set

$$Z_{\mathbf{A}} = \{\mathbf{A}z_1^1, \ldots, \mathbf{A}z_{q_1}^1, \ldots, \mathbf{A}z_1^p, \ldots, \mathbf{A}z_{q_p}^p\}$$

of integer-typed variables $\mathbf{A}z_j^i$ such that the assignment $\mathbf{A}z_j^i = n$ of a value $n \in \mathbb{Z}$ to the variable $\mathbf{A}z_j^i$ corresponds to the scenario where n copies of block **A** are deployed on the j^{th} resource of the i^{th} PLC.

By Definition 3.1, the model for each function block is the union of the function block models from which it is composed. For example, the function block **Conveyors** $\equiv comp(\mathbf{A}, \mathbf{B})$ depicted in Fig. 3b is composed of two basic function blocks **A** and **B** and thus has the model $Z_{\mathbf{Conveyors}} = Z_{\mathbf{A}} \cup Z_{\mathbf{B}}$.

IV. Requirement Specification

The second step of our approach involves the specification of high-level configuration requirements of the system as quantifier-free first order formulae. The formulae are input to the configuration engine and subsequently solved as a constraint problem. Before describing this step in detail, it is useful to recall some basic results from universal algebra and formal logic. Interested readers may refer to the standard texts [20], [21] for a detailed discussion of these topics.

A. Quantifier-free first order formulae

We define a standard signature Σ of the integers and Booleans consisting of the sort $Integer$, a constant $0 :\rightarrow Integer$ and the operation symbols $+, - : Integer \times Integer \rightarrow Integer$ for addition and subtraction. We add the sort $Bool$ to Σ and include the constant Boolean truth symbols $true, false :\rightarrow Bool$. The standard logical operation symbols $\wedge, \vee : Bool \times Bool \rightarrow Bool$ and $\neg : Bool \rightarrow Bool$ are also included in the signature Σ.

We define the set of quantifier-free first order formulae as follows. Let Z be the set of integer-typed variables. The set of Σ-terms over Z are defined inductively by the rules

$$t ::= z \mid 0 \mid t_1 + t_2 \mid t_1 - t_2 \mid b_1 \wedge b_2 \mid b_1 \vee b_2 \mid \neg b_1 \qquad (2)$$

where z is an integer-typed variable and t_1, t_2 and b_1, b_2 are integer and Boolean-typed Σ-terms respectively. By adding relations $=: Integer \times Integer \rightarrow Bool$ and $>: Integer \times Integer \rightarrow Bool$ to the signature, we define the set F of quantifier-free first order formulae over the signature Σ inductively by the rules

$$\alpha ::= t_1 = t_2 \mid t_1 > t_2 \mid \neg\alpha_1 \mid \alpha_1 \wedge \alpha_2 \mid \alpha_1 \vee \alpha_2$$

where t_1 and t_2 are terms obtained from application of the rules specified by Equation (2) and α_1 and α_2 are quantifier-free first order formulae. When dealing with formulae arising in applications, we write $\alpha(\mathbf{z})$ to mean the formula α contains at least one instance of the variables in the tuple $\mathbf{z} = (z_1, \ldots, z_p)$.

Let $[Z \rightarrow \mathbb{Z}]$ denote the set of *assignment* functions mapping integer values to the variables in Z. Given an assignment $a : Z \rightarrow \mathbb{Z}$ term evaluation $[\![\alpha(\mathbf{z})]\!](a)$ works out the value of a logical formula by substituting the value $a(z)$ for each variable z in the formula $\alpha(\mathbf{z})$. The satisfiability relation $\models$ over $\mathbb{Z}$ is defined as $a \models \alpha(\mathbf{z}) \iff [\![\alpha(\mathbf{z})]\!](a) = \mathbf{t}$. In this case, we say that a is a *satisfiable assignment* of the formula $\alpha(\mathbf{z})$, where the satisfiability relationship $\models$ is defined inductively over the structure of the formulas in F.

Configuration requirements are specified in terms of logical formulae expressed over variables in Z used in Definition 3.1 to model function blocks within the system. Each function block $\mathbf{f}$ in the system is associated with a formula in F and contains variables from the function block model $Z_{\mathbf{f}} \subset Z$. Formally, we define a *requirement mapping* $\phi : FB \rightarrow F$ which associates each function block $\mathbf{f} \in FB$ (the system architecture) with a logical formulae $\phi_{\mathbf{f}} \in F$. This requirement mapping has the property $var(\phi_{\mathbf{f}}) \subseteq Z_{\mathbf{f}}$ meaning that the

variables appearing in logical formula formalising the requirements of **f** must be formed from variables within its function block model. When no requirements are needed for **f** we set $\phi_{\mathbf{f}} = \epsilon$, the empty formula which is trivially satisfiable by any variable assignment.

B. Baggage-Handling System Requirements

Suppose there are $p \geq 2$ PLCs available and that there are q_i resources available on the i^{th} PLC. We translate the high-level requirements (R1) to (R7) for the airline baggage-handling system described in Section II into logical formulae over integer-typed variables in the function block model of $Z_{\mathbf{BHS}}$. The translation outputs a requirement mapping $\phi^{\mathbf{BHS}} : FB \to F$ which associates each function block comprising **BHS** with a logical formula:

Requirement (R1) specifies that each function block has between one and five copies, and a negative number of copies may not appear on any PLC. This requirement is expressed by the formula

$$\alpha_{\mathbf{A}} := \bigwedge_{i=1}^{p} \bigwedge_{j=1}^{q_i} \mathbf{A}z_j^i \geq 0 \wedge \sum_{i=1}^{p}\sum_{j=1}^{q_i} \mathbf{A}z_j^i \geq 1 \wedge \sum_{i=1}^{p}\sum_{j=1}^{q_i} \mathbf{A}z_j^i \leq 5$$

for function block **A**. The requirements for each basic function blocks **B** to **H** have similar formulae $\alpha_{\mathbf{B}}$ to $\alpha_{\mathbf{H}}$.

Requirement (R2) specifies that the basic function block **B** has at least 3 copies distributed over two PLCs. This requirement is expressed by the formula

$$\phi_{\mathbf{B}} := \alpha_{\mathbf{B}} \wedge \sum_{i=1}^{p}\sum_{j=1}^{q_i} \mathbf{B}z_j^i \geq 3 \wedge \sum_{i=1}^{p} nz(\sum_{j=1}^{q_i} \mathbf{B}z_j^i) = 2$$

where $nz(n) = 1$ if $n \neq 0$ and 0 otherwise, for $n \in \mathbb{Z}$.

Requirement (R3) specifies the basic function block **D** is deployed over 4 PLC resources. It is expressed as

$$\phi_{\mathbf{D}} := \alpha_{\mathbf{D}} \wedge \sum_{i=1}^{p}\sum_{j=1}^{q_i} nz(\mathbf{D}z_j^i) = 4$$

Requirement (R4) further constrains the basic function block **C** by stipulating exactly 1 copy is to be deployed. This requirement is expressed by the formula

$$\phi_{\mathbf{C}} := \alpha_{\mathbf{C}} \wedge \sum_{i=1}^{p}\sum_{j=1}^{q_i} \mathbf{C}z_j^i = 1$$

Requirement (R5) states that the basic function blocks **A** and **B** must be identically deployed. This means that each PLC resource that hosts an copy of **A** must also host a copy of **B**. We formalise this requirement as the formula

$$\phi_{\textbf{Conveyors}} := \bigwedge_{i=1}^{p} \bigwedge_{j=1}^{q_i} (\mathbf{A}z_j^i = \mathbf{B}z_j^i).$$

Requirement (R6) states that all copies of function block **Multex2** are on PLC 1. As **Multex2** $\equiv comp(\mathbf{G}, \mathbf{H})$, we partition the requirement into two formulae:

$$\phi_{\mathbf{G}} := \alpha_{\mathbf{G}} \wedge \sum_{i=2}^{p}\sum_{j=1}^{q_i} \mathbf{G}z_j^i = 0,\ \phi_{\mathbf{H}} := \alpha_{\mathbf{H}} \wedge \sum_{i=2}^{p}\sum_{j=1}^{q_i} \mathbf{H}z_j^i = 0$$

Requirement (R7) specifies that each resource of PLC 2 must have a copy of **Multex1**. Since **Multex1** $\equiv comp(\mathbf{E}, \mathbf{F})$, we partition the requirement into the following two formulae

$$\phi_{\mathbf{E}} := \alpha_{\mathbf{E}} \wedge \bigwedge_{j=1}^{q_i} \mathbf{E}z_j^2 > 0 \text{ and } \phi_{\mathbf{F}} := \alpha_{\mathbf{F}} \wedge \bigwedge_{j=1}^{q_i} \mathbf{F}z_j^2 > 0.$$

As we have described above, high-level configuration requirements are specified as logical formulae. The formula constrain the **BHS** system, with each function block associated with a formula by the requirement mapping $\phi^{\mathbf{BHS}} : FB \to F$.

V. Compositional Resolution

Satisfiability modulo theories (SMTs) [22] represent a class of tools and techniques that determine the satisfiability of formulae expressed in a logical theory. An SMT *decision procedure* computes and returns an assignment of values to variables that satisfy the formula. For formulae over variables in Z, we model an SMT decision procedure as a total function $smt : F \to [Z \to \mathbb{Z}]$ such that $smt(\alpha) \models \alpha$ if $\alpha(\mathbf{z})$ is satisfiable in $\mathbb{Z}$. Otherwise, the procedure reports $\alpha(\mathbf{z})$ as unsatisfiable. We wish to apply the SMT decision procedure smt to determine *compliance* of a system **f** to the requirements specified by the requirement map ϕ. We formulae the *SMT constraint problem* comprising the logical conjunction of all formulae associated with the system function blocks according to ϕ. We make this mathematically precise in the following

Definition 5.1 (SMT constraint problem): Let $\phi : FB \to F$ be a requirement mapping and let **f** be a function block. We define the function $\rho : FB \times [FB \to F] \to F$ over the inductive structure of function blocks:

Base case: If **f** is a basic function block then $\rho(\mathbf{f}, \phi) = \phi_{\mathbf{f}}$.

Structural Induction: If $\mathbf{f} \equiv comp(\mathbf{f}_1, \ldots, \mathbf{f}_m)$ for $\mathbf{f}_1, \ldots, \mathbf{f}_m$ in FB then $\rho(\mathbf{f}, \phi) = \phi_{\mathbf{f}} \wedge (\wedge_{i=1}^{m} \phi_{\mathbf{f}_i})$.

From this definition, we formulate a mathematically precise notion of compliance, whereby a function block **f** is compliant with requirements ϕ if, and only if,

$$smt(\rho(\mathbf{f}, \phi)) \models \rho(\mathbf{f}, \phi). \tag{3}$$

Since modern industrial automations such as the **BHS** from the case study comprise large numbers of components, the SMT constraint problem as defined in (3) is often very large, making efficient resolution of constraints difficult. To overcome this difficulty, we describe a divide-and-conquer approach to resolving constraints.

The *compositional SMT approach* to constraint resolution is based on the observation that a logical formula $\alpha \in F$ may be syntactically decomposed into a sequence

$$\pi(\alpha) = (\alpha_1, \ldots, \alpha_n) \tag{4}$$

of *independent* formulae α_i such that no two formulae in the sequence share common variables. In symbols,

$$var(\alpha_i) \cap var(\alpha_j) = \emptyset,\ i \neq j, \tag{5}$$

for $1 \leq i, j \leq n$. The compositional approach $comp : Seq \to [Z \to \mathbb{Z}]$ is defined by the equation

$$comp(\pi(\alpha)) = smt(\alpha_1) \oplus \cdots \oplus smt(\alpha_n) \tag{6}$$

for the sequence $\pi(\alpha) \in Seq$ where the operation $\oplus$ combines assignment functions obtained from resolving smaller constraint satisfiability problems rather than solving the equivalent and potentially huge monolithic constraint problem (3) (cf. Theorem 4.1 [16]).

TABLE I: Single-step resolution of basic blocks **A** and **B**.

Step		z_1^1	z_1^2	z_2^1	z_2^2
1	**A**	0	1	0	2
	B	0	1	0	2

*a) Compositional SMT Analysis of **BHS**:* We apply the compositional SMT approach to compute a valid configuration of the function block **BHS** for the baggage-handling system case study over two PLCs, each with two resources. Mathematically, each basic function block is modelled by a set of four variables from Z, e.g. $Z_{\mathbf{A}} = \{\mathbf{A}z_1^1, \mathbf{A}z_2^1, \mathbf{A}z_1^2, \mathbf{A}z_2^2\}$ for the basic function block **A**. We construct the sequence from the requirement map $\phi^{\mathbf{BHS}}$ specified for the block **BHS** to obtain

$$\pi = (\phi_{\mathbf{Conveyors}} \wedge \phi_{\mathbf{A}} \wedge \phi_{\mathbf{B}}, \phi_{\mathbf{C}}, \phi_{\mathbf{D}}, \phi_{\mathbf{E}}, \phi_{\mathbf{F}}, \phi_{\mathbf{G}}, \phi_{\mathbf{H}}). \quad (7)$$

We note that the formulae in (7) are pairwise disjoint and satisfy (5). In particular, the first element $\phi_{\mathbf{Conveyors}} \wedge \phi_{\mathbf{A}} \wedge \phi_{\mathbf{B}}$ of the sequence comprises formulae that are not pairwise independent e.g. $var(\phi_{\mathbf{Conveyors}}) \cap var(\phi_{\mathbf{A}}) \cap var(\phi_{\mathbf{B}}) \neq \emptyset$ and thus cannot be decomposed in any way such that (5) is satisfied. Computing $smt(\phi_{\mathbf{Conveyors}} \wedge \phi_{\mathbf{A}} \wedge \phi_{\mathbf{B}})$ yields the assignment presented in Table I of integer values to function block model variables in $Z_{\mathbf{Conveyors}}$ satisfying $\phi_{\mathbf{Conveyors}} \wedge \phi_{\mathbf{A}} \wedge \phi_{\mathbf{B}}$. Similarly, the remaining basic function blocks **C** to **H** are resolved in a series of six SMT resolution steps. Table II lists the resolution results. The compositional approach (6) composes the results of each SMT analysis step in Tables I and II to obtain a system configuration of the function block **BHS** that is compliant to the high-level requirements (R1) - (R7).

VI. Incremental Re-resolution

While compositional analysis provides a speedup over a monolithic approach, it may still be unnecessary and infeasible to re-resolve every function block due system size and frequency of change. To address this limitation, an incremental re-resolution step re-resolves only affected function blocks after changes in the system architecture (addition/removal of components), and system requirements (addition/removal/update). We model change as a transformation $(\mathbf{f}, \phi) \xrightarrow{T} (\mathbf{f}', \phi')$, transforming the function block **f** and its requirement map ϕ to an updated function block **f′** and updated requirement map ϕ'. The transformation T of any function block **g** of **f** is a pair comprising $\mathbf{g}' \in FB$ and $\tau : FB \to F$ and is carried out in the following steps:

1. Perform function block substitution $\mathbf{f}' \equiv \mathbf{f}[\mathbf{g}/\mathbf{g}']$, substituting all instances of the function block **g** in **f** with the new function block **g′**,
2. Perform requirement substitution $\phi' = \phi[\tau]$, substituting a formula $\phi(\mathbf{f}_i)$ with a new formula $\tau(\mathbf{f}_i)$ for $\mathbf{f}_i \in dom(\tau)$.

Not all sub-blocks of block **f** must be re-resolved to ensure compliance after a transformation T. In order to identify those SMT analysis steps in the sequence π of the form (4) that must be re-resolved, we define the *characteristic set*

$$U = \bigcup_{\mathbf{f} \in dom(\tau)} var(\tau(\mathbf{f})) \quad (8)$$

which specifies the set of variables whose formulae have been affected by τ in transformation T. The characteristic set U contains all variables that are altered (directly or indirectly) by a transformation T in the function block **f**. We use U to identify

TABLE II: Six-step resolution of basic blocks **C** to **H**.

Step		z_1^1	z_1^2	z_2^1	z_2^2
2	**C**	1	0	0	0
3	**D**	1	1	1	1
4	**E**	0	0	1	1
5	**F**	0	0	1	1
6	**G**	1	0	0	0
7	**H**	1	0	0	0

the steps in the sequence π obtained from the compositional SMT analysis of **f** that have been modified by a change and thus require re-resolution. Each logical formulae $\alpha \in \pi$ is checked according to the criteria

$$\begin{aligned} \alpha \in \mu &\iff var(\alpha) \cap U \neq \emptyset \\ \alpha \in \sigma &\iff var(\alpha) \cap U = \emptyset \end{aligned} \quad (9)$$

partitioning π into two sequences μ and σ. If Criteria (9) is satisfied $\alpha \in \mu$, forming the *re-resolution sequence* μ. Otherwise, the step remain unchanged and $\alpha \in \sigma$. We define the incremental re-resolution approach as

$$comp(\mu) \oplus a \models \rho(\mathbf{f}', \tau') \quad (10)$$

where the assignment $a : Z \to \mathbb{Z}$ represents the satisfiable assignment of the logical formulae in the sequence σ obtained in previous SMT analysis steps and $comp(\mu)$ carries out compositional SMT analysis on the re-resolution sequence μ (cf. Theorem 4.2 [16]).

b) Basic Function Block Addition: We suppose that after composition resolution has been performed that a new basic block **I** is added to the composite block **Conveyor** in the baggage-handling system **BHS**. We assume at least four copies of **I** are to be deployed and Requirement (R5) requiring all basic blocks of **Conveyer** to be identically distributed must be maintained. We model this change as a transformation T involving the term substitution $\mathbf{BHS}' \equiv \mathbf{BHS}[comp(\mathbf{A}, \mathbf{B})/comp(\mathbf{A}, \mathbf{B}, \mathbf{I})]$ which updates the system architecture. The requirement map $\phi^{\mathbf{BHS}'} = \phi^{\mathbf{BHS}}[\tau_{add}]$ is updated, where $\tau_{add} : \{\mathbf{I}, \mathbf{Conveyors}\} \to F$ is the requirement map such that

$$\tau_{add}(\mathbf{I}) = \bigwedge_{i=1}^{p} \bigwedge_{j=1}^{q_i} \mathbf{I}z_j^i \geq 0 \wedge \sum_{i=1}^{p} \sum_{j=1}^{q_i} \mathbf{I}z_j^i \geq 4 \quad (11)$$

$$\tau_{add}(\mathbf{Conveyors}) = \phi_{\mathbf{Conveyors}} \wedge \bigwedge_{i=1}^{p} \bigwedge_{j=1}^{q_i} (\mathbf{B}z_j^i = \mathbf{I}z_j^i) \quad (12)$$

where $p = 2$, and $q_1 = q_2 = 2$ specifies two PLCs each with two resources. The requirement map $\tau_{add}(\mathbf{I})$ formalises the non-negative requirement and that at least four copies of **I** are to be deployed. The formula $\phi_{\mathbf{Conveyors}}$ is the original formula associated to **Conveyors** by the translation of Requirement (R5) and $\tau_{add}(\mathbf{Conveyors})$ ensures the identical distribution of basic function blocks **A**, **B** and **I**.

c) Characteristic set of τ_{add}: We compute the characteristic set U_{add} of τ_{add} containing new requirements (11) and (12) for function blocks **I** and **Conveyors** as $U_{add} = \{\mathbf{A}z_1^1, \mathbf{A}z_2^1, \mathbf{A}z_1^2, \mathbf{A}z_2^2\} \cup \{\mathbf{B}z_1^1, \mathbf{B}z_2^1, \mathbf{B}z_1^2, \mathbf{B}z_2^2\} \cup \{\mathbf{I}z_1^1, \mathbf{I}z_2^1, \mathbf{I}z_1^2, \mathbf{I}z_2^2\}$, comprising variables from the models of **A**, **B** and **I**

TABLE III: Re-resolution of function blocks **A**, **B** and **I**.

Step		z_1^1	z_1^2	z_2^1	z_2^2
$1'$	**A**	0	1	0	2
	B	0	1	0	2
	I	0	1	0	2

*d) Re-resolution of **BHS**:* Applying Criteria (9) to the compositional SMT sequence (7) of block **BHS** and the characteristic set U_{add}, we define

$$\mu = (\phi_{Conveyers} \wedge \phi_{\mathbf{A}} \wedge \phi_{\mathbf{B}} \wedge \phi_{\mathbf{I}})$$
$$\sigma = (\phi_{\mathbf{C}}, \phi_{\mathbf{D}}, \phi_{\mathbf{E}}, \phi_{\mathbf{F}}, \phi_{\mathbf{G}}, \phi_{\mathbf{H}})$$

and compositional SMT analysis on the subsequence μ yields the satisfiable solution $comp(\mu)$ presented in Table III.

By the incremental re-resolution approach (10) we have $comp(\mu) \oplus a \models \rho(\mathbf{BHS}', \phi^{\mathbf{BHS}'})$, where a is the assignment presented in Table II from previous resolution steps.

*e) Requirement update of function block **C**:* For the BHS case study, we suppose that a change in function block **C**'s requirement specifies a total of four copies to be distributed across PLCs instead of one. We model this change by the transformation T comprising $\tau_{update}(\mathbf{C}) = \alpha_{\mathbf{C}} \wedge \sum_{i=1}^{p} \sum_{j=1}^{q_i} \mathbf{C}z_j^i = 4$, where $p = 2$ and $q_1 = q_2 = 2$ and $\alpha_{\mathbf{C}}$ is the formula obtained in the translation of Requirement (R1). Requirement substitution yields a new requirement map $\phi^{\mathbf{BHS}'} = \phi^{\mathbf{BHS}}[\tau_{update}]$. Since T does not change the architecture of $\mathbf{BHS}'$ term substitution is unnecessary. We compute the characteristic set as $U_{update} = \{\mathbf{C}z_1^1, \mathbf{C}z_2^1, \mathbf{C}z_1^2, \mathbf{C}z_2^2\}$. Applying incremental re-resolution, we obtain the sequence $\mu = (\phi_{\mathbf{C}}^{\mathbf{BHS}'})$ and $comp(\mu)$ yields the satisfiable assignment shown in Tab. IV. Since no other steps are necessary, we conclude $comp(\mu) \oplus a' \models \rho(\mathbf{BHS}', \phi^{\mathbf{BHS}'})$, where a' is the satisfiable assignment obtained in the previous re-solution step in Ex. VI-0d.

f) Function block removal: Removing function blocks updates the system architecture but no re-resolution is required. Consider the change scenario in which the basic function block **H** is removed from **BHS**. This scenario is a transformation involving term substitution whereby $\mathbf{BHS}' \equiv \mathbf{BHS}[\mathbf{Mutex2}/\mathbf{G}]$. Since no new requirements are added, we specify $\tau_{remove} : \emptyset \rightarrow F$, which leaves the requirement map $\phi^{\mathbf{BHS}}$ unchanged. The characteristic set $U_{remove} = \emptyset$ and no resolution satisfy Criteria (9) and $\mu = ()$.

VII. Implementation and Scalability Experiments

To demonstrate the scalability of the incremental re-configuration approach described in this paper, we adapted the prototype tool implemented in [16]. The core component of the tool is the `Z3Engine` that realises the work flow presented in Figure 1 and carries out the compositional configuration and incremental re-configuration steps using the Z3 SMT tool. The model synthesis step generates models specified by the `Z3Model` class, adapted to suit the algebraic model (Def. 3.1) of the baggage-handling system **BHS**. The requirements-to-formula translation step was implemented by the `Requirement` class that automatically translates high-level system configuration requirements into logical formulae expressed in terms of the Z3 assertion language.

The `Z3Engine` features the following methods for configuration and re-configuration of industrial systems. `monolithicSolve()` computes a satisfiable configuration monolithically (3), `compositionalSolve()` computes a satisfiable configuration compositionally (6), and `incrementalSolve()` computes a re-configuration satisfying updated requirements (10). To compare the performance of the proposed approaches we carried out a range of experiments on a 2.3 GHz Intel Core i7 MacBook Pro computer with 16GB of memory.

TABLE IV: Re-resolution of **C**.

Step		z_1^1	z_1^2	z_2^1	z_2^2
$2'$	**C**	1	1	0	2

We created larger **BHS** systems by adding conveyors and photo-eye controllers and some mutual exclusion logic. This functionality added 25 blocks to the system including atomic and composite blocks (at the same level as blocks **Conveyors** and **Mutex1** in Fig. 3b). We used this step-size of 25 to create 20 variants of **BHS**, with the largest system containing 781 function blocks. We kept the number of PLCs to be constant at 5 for all the **BHS** variants, with each PLC assumed to contain 4 resources. The algebraic model of the smallest system containing 281 blocks had 5620 variables (281×5 PLCs× 4 resources per PLC) while the largest system containing 781 function blocks had a model with 15620 variables. Each newly added atomic block was constrained to have 1-5 copies. A quarter of the composite blocks added at the level of **Conveyors** or **Mutex1** in Fig. 3b were constrained to have identical deployment for their constituent blocks. Only one block at the level of **ModelView** in Fig. 3b was required to have identical deployment for all of its children blocks.

The performance of both the monolithic and compositional SMT methods to configure the **BHS** variants is presented in Fig. 4. On average, the compositional approach was 9.347 times faster than the monolithic approach, with the gap between the two approaches steadily increasing with larger system sizes. This clearly shows that the compositional approach was able to configure systems in a much more scalable manner.

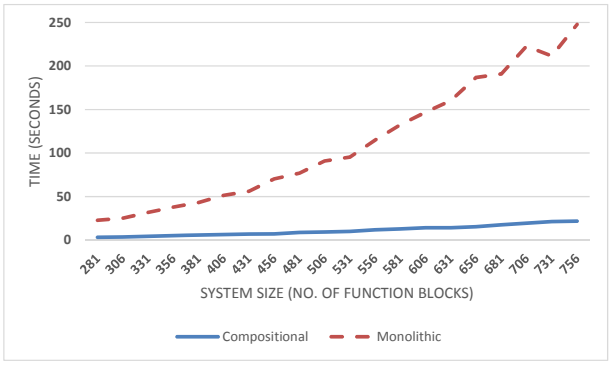
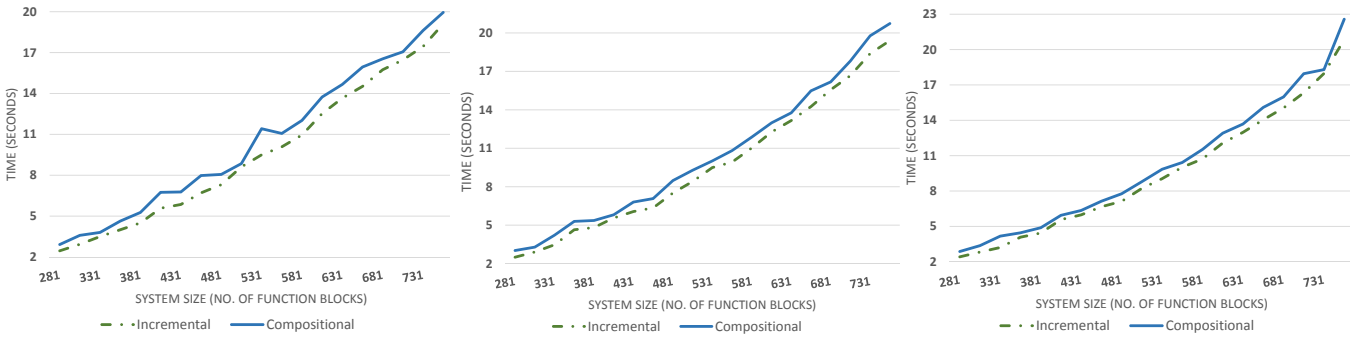


Fig. 4: Experiment results comparing monolithic and compositional configuration approaches

Next, we used incremental re-resolution and compositional resolution to re-configure all **BHS** variants in response to three kinds of changes. Fig. 5a shows that the incremental approach provided an average speedup of 1.12 seconds over the compositional approach when a single basic block was added to the system. Fig. 5b presents the results of an experiment in which a composite function block with two basic function blocks is



(a) Basic block addition (b) Composite block addition (c) 4+ copies of a basic block

Fig. 5: Experimental results comparing performance of compositional and incremental re-configuration

added whose requirements constrain each of its basic function blocks. The speedup obtained by the incremental approach over the compositional approach was 1.10 seconds. Finally, Figure 5c presents the results of an experiment in which the constraints of a basic function block are updated in accordance to changing system requirements. We observed an average speedup of 1.09 seconds when the incremental approach was used, as compared to the compositional approach. For function block removals, the incremental approach requires zero time while the compositional approach takes between 2 and 20 seconds. In a real-world scenario, a system may undergo many basic and composite block additions and removals, and thus the average speedup of incremental re-configuration would be much larger.

In summary, both the incremental and compositional approaches provide significant speedup over monolithic and manual reconfiguration. This enables automatic reconfiguration for a wider range of systems - those for which the range of response times exhibited by our approaches is acceptable. The performance of our approach will depend on how *decomposable* the (re)configuration requirements are. This provides yet another reason why building complex monolithic systems tends to be poor engineering practice.

VIII. Concluding Remarks

We present an approach that synthesises configurations that comply with the requirements of evolving control systems. These requirements can be expressed as SMT constraints. We described a compositional approach which improved the ability to determine re-configuration of function blocks affected by change. We applied our approach to a real-world airport baggage handling system whose configuration requirements were specified in terms of logical formulae to form a constraint satisfiability problem. Our experiments showed that the incremental approach performed consistently better than compositional SMT analysis in response to slight changes in the system. We extend the incremental framework of [16], to create new algebraic models for function blocks and used the Z3 SMT solver to determine valid configurations.

There are several directions in which we plan to extend our work. The framework we have implemented can be extended to a software tool which enables our approach to be used in existing frameworks for industrial automation and control. An important step in this process is the development of a domain-specific language to express requirements to be automatically translated into an SMT constraint problem. We also plan to extend the theoretical foundations of our approach to include the use of temporal logics to specify requirements for IEC 61499 systems to be verified during runtime.

References

[1] V. Vyatkin, *IEC 61499 function blocks for embedded and distributed control systems design*. International Society of Automation, 2007.

[2] G. Black and V. Vyatkin, "Intelligent component-based automation of baggage handling systems with IEC 61499," *T-ASE*, vol. 7, no. 2, pp. 337–351, 2010.

[3] N. Higgins, V. Vyatkin, N. Nair, and K. Schwarz, "Distributed power system automation with IEC 61850, IEC 61499, and intelligent control," *Transactions on Systems, Man, and Cybernetics*, vol. 41, no. 1, pp. 81–92, 2011.

[4] K. Thramboulidis, "IEC 61499 in factory automation," in *Advances in Computer, Information, and Systems Sciences, and Engineering*. Springer, 2006, pp. 115–124.

[5] P. Leitão, V. Marik, and P. Vrba, "Past, present, and future of industrial agent applications," *Transactions on Industrial Informatics*, vol. 9, no. 4, pp. 2360–2372, 2013.

[6] X. V. Wang and X. W. Xu, "Icms: a cloud-based manufacturing system," in *Cloud manufacturing*. Springer, 2013, pp. 1–22.

[7] A. Schimmel and A. Zoitl, "Distributed online change for iec 61499," in *Emerging Technologies & Factory Automation (ETFA), 2011 IEEE 16th Conference on*. IEEE, 2011, pp. 1–7.

[8] Y. Xu, R. Brennan, X. Zhang, and H. Norrie, "A reconfigurable concurrent function block model and its implementation in real-time java," 2002.

[9] C. Sünder, "Evaluation of downtimeless system evolution in automation and control systems," Ph.D. dissertation, Automation and Control Institute, Vienna University of Technology, 2008.

[10] C. Sünder, V. Vyatkin, and A. Zoitl, "Formal verification of downtimeless system evolution in embedded automation controllers," *TECS*, vol. 12, no. 1, p. 17, 2013.

[11] T. Strasser *et al.*, "Enhanced IEC 61499 device management execution and usage for downtimeless reconfiguration," in *INDIN*, vol. 2. IEEE, 2007, pp. 1163–1168.

[12] A. Tešanović, S. Nadjm-Tehrani, and J. Hansson, "Modular verification of reconfigurable components," in *Component-Based Software Development for Embedded Systems*. Springer, 2005, pp. 59–81.

[13] Z. E. Bhatti, R. Sinha, and P. S. Roop, "Observer based verification of IEC 61499 function blocks," in *INDIN*. IEEE, 2011, pp. 609–614.

[14] A. Zoitl, G. Grabmair, F. Auinger, and C. Sunder, "Executing real-time constrained control applications modelled in IEC 61499 with respect to dynamic reconfiguration," in *INDIN*. IEEE, 2005, pp. 62–67.

[15] I. Hegny *et al.*, "Integrating software agents and IEC 61499 realtime control for reconfigurable distributed manufacturing systems," in *Industrial Embedded Systems*. IEEE, 2008, pp. 249–252.

[16] K. Johnson and R. Calinescu, "Efficient re-resolution of SMT specifications for evolving software architectures," *QoSA*, 2014, to appear.

[17] K. Johnson, R. Calinescu, and S. Kikuchi, "An incremental verification framework for component-based software systems," in *CBSE'13*, 2013, pp. 33–42.

[18] R. Calinescu, S. Kikuchi, and K. Johnson, "Compositional reverification of probabilistic safety properties for large-scale complex it systems," in *Large-Scale Complex IT Systems. Development, Operation and Management*, ser. Lecture Notes in Computer Science, R. Calinescu and D. Garlan, Eds. Springer Berlin Heidelberg, 2012, vol. 7539, pp. 303–329.

[19] V. Vyatkin, "IEC 61499 as enabler of distributed and intelligent automation: State-of-the-art review," *Transactions on Industrial Informatics*, vol. 7, no. 4, pp. 768–781, 2011.

[20] K. Meinke and J. V. Tucker, "Universal algebra," in *Handbook of logic in computer science*. Oxford University Press, 1992, vol. 1, pp. 189–368.

[21] W. Rautenberg, "A concise introduction to mathematical logic," *Universitext Springer*, 2006.

[22] L. De Moura and N. Bjørner, "Z3: An efficient SMT solver," in *TACAS*. Springer, 2008, pp. 337–340.