A General Framework for Architecture Composability

Paul Attie[1], Eduard Baranov[2], Simon Bliudze[2], Mohamad Jaber[1], and Joseph Sifakis[2]

[1] American University of Beirut, Lebanon
{pa07,mj54}@aub.edu.lb
[2] École Polytechnique Fédérale de Lausanne, Station 14, 1015 Lausanne, Switzerland
firstname.lastname@epfl.ch

Abstract. Architectures depict design principles: paradigms that can be understood by all, allow thinking on a higher plane and avoiding low-level mistakes. They provide means for ensuring correctness by construction by enforcing global properties characterizing the coordination between components. An architecture can be considered as an operator A that, applied to a set of components $\mathcal{B}$, builds a composite component $A(\mathcal{B})$ meeting a characteristic property Φ. Architecture composability is a basic and common problem faced by system designers. In this paper, we propose a formal and general framework for architecture composability based on an associative, commutative and idempotent architecture composition operator '$\oplus$'. The main result is that if two architectures A_1 and A_2 enforce respectively safety properties Φ_1 and Φ_2, the architecture $A_1 \oplus A_2$ enforces the property $\Phi_1 \wedge \Phi_2$, that is both properties are preserved by architecture composition. We also establish preservation of liveness properties by architecture composition. The presented results are illustrated by a running example and a case study.

1 Introduction

Architectures depict design principles: paradigms that can be understood by all, allow thinking on a higher plane and avoiding low-level mistakes. They provide means for ensuring correctness by construction by enforcing global properties characterizing the coordination between components.

Using architectures largely accounts for our ability to master complexity and develop systems cost-effectively. System developers extensively use libraries of reference architectures ensuring both functional and non-functional properties, for example fault-tolerant architectures, architectures for resource management and QoS control, time-triggered architectures and security architectures. Nonetheless, we still lack theory and methods for combining architectures in principled and disciplined fully correct-by-construction design flows.

Informally speaking, an architecture can be considered as an operator A that, applied to a set of components $\mathcal{B}$ builds a composite component $A(\mathcal{B})$ meeting a characteristic property Φ. In a design process, it is often necessary to combine more than one architectural solution on a set of components to achieve a

D. Giannakopoulou and G. Salaün (Eds.): SEFM 2014, LNCS 8702, pp. 128–143, 2014.

global property. System engineers use libraries of solutions to specific problems and they need methods for combining them without jeopardizing their characteristic properties. For example, a fault-tolerant architecture combines a set of features building into the environment protections against trustworthiness violations. These include 1) triple modular redundancy mechanisms ensuring continuous operation in case of single component failure; 2) hardware checks to be sure that programs use data only in their defined regions of memory, so that there is no possibility of interference; 3) default to least privilege (least sharing) to enforce file protection. Is it possible to obtain a single fault-tolerant architecture consistently combining these features? The key issue here is *architecture composability* in the integrated solution, which can be formulated as follows.

Consider two architectures A_1 and A_2, enforcing respectively properties Φ_1 and Φ_2 on a set of components $\mathcal{B}$. That is, $A_1(\mathcal{B})$ and $A_2(\mathcal{B})$ satisfy respectively the properties Φ_1 and Φ_2. Is it possible to find an architecture $A_1 \oplus A_2$ such that the composite component $(A_1 \oplus A_2)(\mathcal{B})$ meets $\Phi_1 \wedge \Phi_2$? For instance, if A_1 ensures mutual exclusion and A_2 enforces a scheduling policy is it possible to find an architecture on the same set of components that satisfies both properties?

Architecture composability is a very basic and common problem faced by system designers. Manifestations of lack of composability are also known as feature interaction in telecommunication systems [1].

The development of a formal framework dealing with architecture composability implies a rigorous definition of the concept of architecture as well as of the underlying concepts of components and their interaction. The paper proposes such a framework based on results showing how architectures can be used for achieving correctness by construction in a rigorous component-based design flow [2]. The underlying theory is inspired from BIP [3]. BIP is a component framework rooted in well-defined operational semantics. It proposes an expressive and elegant notion of interaction models for component composition. Interaction models can be studied as sets of Boolean constraints expressing interactions between components. BIP has been fully implemented in a language and supporting toolset, including compilers and code generators [4].

BIP allows the description of composite components as an expression $\gamma(\mathcal{B})$, where $\mathcal{B}$ is a set of atomic components and γ is an interaction model. Atomic components are characterized by their behaviour specified as transition systems.

An *interaction model* γ is a set of *interactions*. Each interaction is a set of actions of the composed components, executed synchronously. The meaning of γ can be specified by using operational semantics rules defining the transition relation of the composite component $\gamma(\mathcal{B})$ in terms of transition relations of the composed components $\mathcal{B}$. Intuitively, for each interaction $a \in \gamma$, $\gamma(\mathcal{B})$ can execute a transition labelled by a iff the components involved in a can execute the corresponding transitions labelled by the actions composing a, whereas other components do not move. A formal definition is given in Sect. 2 (Def. 2).

Given a set of components $\mathcal{B}$ an *architecture* is an operator A such that $A(\mathcal{B}) = \gamma(\mathcal{C}, \mathcal{B})$, where γ is an interaction model and $\mathcal{C}$ a set of coordinating components, and $A(\mathcal{B})$ satisfies a characteristic property Φ_A.

According to this definition, an architecture A is a solution to a specific coordination problem, specified by Φ_A, by using an interaction model specified by γ and $\mathcal{C}$. For instance, for distributed architectures, interactions are point-to-point by asynchronous message passing. Other architectures adopt a specific topology (e.g. ring architectures, hierarchically structured architectures). These restrictions entail reduced expressiveness of the interaction model γ that must be compensated by using the additional set of components $\mathcal{C}$ for coordination. The characteristic property Φ_A assigns a meaning to the architecture that can be informally understood without the need for explicit formalization (e.g. mutual exclusion, scheduling policy, clock synchronization).

Our Contributions. We propose a general formal framework for architecture composability based on a composition operator '$\oplus$' which is associative, commutative and idempotent. We consider that characteristic properties are the conjunction of safety properties and liveness properties. We show that if two architectures A_1 and A_2 enforce respectively safety properties Φ_1 and Φ_2, the architecture $A_1 \oplus A_2$ enforces $\Phi_1 \wedge \Phi_2$, that is both properties are preserved by architecture composition. The concept of liveness for architectures derives from the Büchi-acceptance condition. We designate a subset of states of each coordinator as "idle", meaning that it is permissible for the coordinator to remain in such a state forever. Otherwise, the controller must execute infinitely often. The main result guaranteeing liveness preservation is based on a "pairwise noninterference" check of the composed architectures that can be performed algorithmically.

The paper is structured as follows. Sect. 2 introduces the notions of components and architecture, as well as the corresponding composition operators. Sect. 3 presents the key results about the preservation of safety and liveness properties. Sect. 4 illustrates the application of our framework on an Elevator control use case. Some related work is discussed in Sect. 5, and Sect. 6 concludes.

2 The Theory of Architectures

2.1 Components and Architectures

Definition 1 (Components). *A* component *is a Labelled Transition System* $B = (Q, q^0, P, \rightarrow)$*, where* Q *is a set of* states*,* $q^0 \in Q$ *is the* initial state*,* P *is a set of* ports *and* $\rightarrow \subseteq Q \times 2^P \times Q$ *is a* transition relation*. Each transition is labelled by an* interaction $a \subseteq P$*. We call* P *the* interface *of* B*. Notations* $q \xrightarrow{a} q'$*,* $q \xrightarrow{a}$ *and* $q \not\xrightarrow{a}$ *are as usual;* Q_B*,* q^0_B*,* P_B *and* $\rightarrow_B$ *denote the constituents of* B*.*

Definition 2 (Interaction model). *Let* $\mathcal{B} = \{B_1, \ldots, B_n\}$ *be a finite set of components with* $B_i = (Q_i, q_i^0, P_i, \rightarrow)$*,*[1] *such that all* P_i *are pairwise disjoint, i.e.* $\forall i \neq j,\ P_i \cap P_j = \emptyset$*. Let* $P = \bigcup_{i=1}^{n} P_i$*. An* interaction model over P *is a subset* $\gamma \subseteq 2^P$*. We call the set of ports* P *the* domain *of the interaction model.*

[1] Here and below, we skip the index on the transition relation $\rightarrow$, since it is always clear from the context.

The composition of $\mathcal{B}$ with the interaction model γ is given by the component $\gamma(\mathcal{B}) = (Q, q^0, P, \rightarrow)$, where $Q = \prod_{i=1}^{n} Q_i$, $q^0 = q_1^0 \dots q_n^0$ and $\rightarrow$ is the minimal transition relation inductively defined by the following rules:

$$\frac{q_i \xrightarrow{\emptyset} q_i'}{q_1 \dots q_i \dots q_n \xrightarrow{\emptyset} q_1 \dots q_i' \dots q_n}, \qquad \frac{a \in \gamma \quad \begin{array}{r} q_i \xrightarrow{a \cap P_i} q_i' \ (\textit{if } a \cap P_i \neq \emptyset) \\ q_i = q_i' \quad (\textit{if } a \cap P_i = \emptyset) \end{array}}{q_1 \dots q_n \xrightarrow{a} q_1' \dots q_n'}.$$

In the sequel, when speaking of a set of components $\mathcal{B} = \{B_1, \dots, B_n\}$, we will always assume that it satisfies all assumptions of Def. 2.

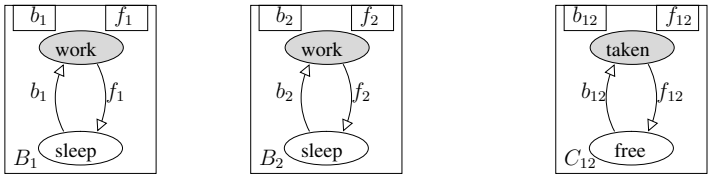
Definition 3 (Architecture). *An* architecture *is a tuple $A = (\mathcal{C}, P_A, \gamma)$, where $\mathcal{C}$ is a finite set of* coordinating components *with pairwise disjoint sets of ports, $\bigcup_{C \in \mathcal{C}} P_C \subseteq P_A$, and $\gamma \subseteq 2^{P_A}$ is an interaction model over P_A.*

Definition 4 (Application of an architecture). *Let $A = (\mathcal{C}, P_A, \gamma)$ be an architecture and let $\mathcal{B}$ be a set of components, such that $\bigcup_{B \in \mathcal{B}} P_B \cap \bigcup_{C \in \mathcal{C}} P_C = \emptyset$ and $P_A \subseteq P \triangleq \bigcup_{B \in \mathcal{B} \cup \mathcal{C}} P_B$. The* application of an architecture *A to the components $\mathcal{B}$ is the component*

$$A(\mathcal{B}) \triangleq \left(\gamma \parallel 2^{P \setminus P_A}\right)(\mathcal{C} \cup \mathcal{B}), \tag{1}$$

where, for interaction models γ' and γ'' over disjoint domains P' and P'' respectively, $\gamma' \parallel \gamma'' \triangleq \{a' \cup a'' \mid a' \in \gamma', a'' \in \gamma''\}$ is an interaction model over $P' \cup P''$.

Architecture A enforces coordination constraints on the components in $\mathcal{B}$. The interface P_A of an architecture A contains all ports of the coordinating components $\mathcal{C}$ and some additional ports, which must belong to the components in $\mathcal{B}$. In the application $A(\mathcal{B})$, the ports belonging to P_A can only participate in the interactions defined by the interaction model γ of A. Ports which do not belong to P_A are not restricted and can participate in any interaction. In particular, they can join the interactions in γ (see (1)). If the interface of the architecture covers all ports of the system, i.e. $P = P_A$, we have $2^{P \setminus P_A} = \{\emptyset\}$ and the only interactions allowed in $A(\mathcal{B})$ are those belonging to γ. Finally, the definition of $\gamma' \parallel \gamma''$, above, requires that an interaction from each of γ' and γ'' be involved in every interaction belonging to $\gamma' \parallel \gamma''$. To enable independent progress in (1), one must have $\emptyset \in \gamma$. (Notice that $\emptyset \in 2^{P \setminus P_A}$ holds always.)



(a) (b)

Fig. 1. Component (*a*) and coordinator (*b*) for Ex. 1.

Example 1 (Mutual exclusion). Consider the components B_1 and B_2 in Fig. 1(a). In order to ensure mutual exclusion of their `work` states, we apply the architecture $A_{12} = (\{C_{12}\}, P_{12}, \gamma_{12})$, where C_{12} is shown in Fig. 1(b), $P_{12} = \{b_1, b_2, b_{12}, f_1, f_2, f_{12}\}$ and $\gamma_{12} = \{\emptyset, b_1 b_{12}, b_2 b_{12}, f_1 f_{12}, f_2 f_{12}\}$.

The interface P_{12} of A_{12} covers all ports of B_1, B_2 and C_{12}. Hence, the only possible interactions are those explicitly belonging to γ_{12}. Assuming that the initial states of B_1 and B_2 are `sleep`, and that of C_{12} is `free`, neither of the two states (`free`, `work`, `work`) and (`taken`, `work`, `work`) is reachable, i.e. the mutual exclusion property $(q_1 \neq \texttt{work}) \vee (q_2 \neq \texttt{work})$ holds in $A_{12}(B_1, B_2)$

Let B_3 be a third component, similar to B_1 and B_2, with the interface $\{b_3, f_3\}$. Since $b_3, f_3 \notin P_{12}$, the interaction model of the application $A_{12}(B_1, B_2, B_3)$ is $\gamma_{12} \parallel \{\emptyset, b_3, f_3\}$. (We omit the interaction $b_3 f_3$, since b_3 and f_3 are never enabled in the same state and, therefore, cannot be fired simultaneously.) Thus, the component $A_{12}(B_1, B_2, B_3)$ is the unrestricted product of the components $A_{12}(B_1, B_2)$ and B_3. The application of A_{12} enforces mutual exclusion between the `work` states of B_1 and B_2, but does not affect the behaviour of B_3.

2.2 Composition of Architectures

Architectures can be intuitively understood as enforcing constraints on the global state space of the system [3, 5]. More precisely, component coordination is realised by limiting the allowed interactions, thus enforcing constraints on the transitions components can take. From this perspective, architecture composition can be understood as the conjunction of their respective constraints. This intuitive notion is formalised by the two definitions below.

Definition 5 (Characteristic predicates [6]). *Denote* $\mathbb{B} = \{\texttt{tt}, \texttt{ff}\}$ *and let* $\gamma \subseteq 2^P$ *be an interaction model over a set of ports* P. *Its* characteristic predicate $(\varphi_\gamma : \mathbb{B}^P \to \mathbb{B}) \in \mathbb{B}[P]$ *is defined by letting* $\varphi_\gamma \stackrel{\Delta}{=} \bigvee_{a \in \gamma} \left(\bigwedge_{p \in a} p \wedge \bigwedge_{p \notin a} \overline{p} \right)$. *For any valuation* $v : P \to \mathbb{B}$, $\varphi_\gamma(v) = \texttt{tt}$ *if and only if* $\{p \in P \mid v(p) = \texttt{tt}\} \in \gamma$. *A predicate* $\varphi \in \mathbb{B}[P]$ *uniquely defines an interaction model* γ_φ, *such that* $\varphi_{\gamma_\varphi} = \varphi$.

Example 2 (Mutual exclusion (contd.)). Consider the interaction model $\gamma_{12} = \{\emptyset, b_1 b_{12}, b_2 b_{12}, f_1 f_{12}, f_2 f_{12}\}$ from Ex. 1. Since the domain of γ_{12} is $P_{12} = \{b_1, b_2, b_{12}, f_1, f_2, f_{12}\}$, its characteristic predicate is (omitting '$\wedge$'):

$$\begin{aligned} \varphi_{\gamma_{12}} &= \overline{b_1}\,\overline{b_2}\,\overline{b_{12}}\,\overline{f_1}\,\overline{f_2}\,\overline{f_{12}} \;\vee\; b_1\,\overline{b_2}\,b_{12}\,\overline{f_1}\,\overline{f_2}\,\overline{f_{12}} \;\vee\; \overline{b_1}\,b_2\,b_{12}\,\overline{f_1}\,\overline{f_2}\,\overline{f_{12}} \\ &\quad \vee\; \overline{b_1}\,\overline{b_2}\,\overline{b_{12}}\,f_1\,\overline{f_2}\,f_{12} \;\vee\; \overline{b_1}\,\overline{b_2}\,\overline{b_{12}}\,\overline{f_1}\,f_2\,f_{12} \\ &= (b_1 \Rightarrow b_{12}) \wedge (f_1 \Rightarrow f_{12}) \wedge (b_2 \Rightarrow b_{12}) \wedge (f_2 \Rightarrow f_{12}) \\ &\quad \wedge\; (b_{12} \Rightarrow b_1 \,\texttt{XOR}\, b_2) \wedge (f_{12} \Rightarrow f_1 \,\texttt{XOR}\, f_2) \wedge (b_{12} \Rightarrow \overline{f_{12}})\,. \end{aligned} \tag{2}$$

Intuitively, the implication $b_1 \Rightarrow b_{12}$, for instance, means that, for the port b_1 to be fired, it is necessary that b_{12} be fired in the same interaction [6].

Definition 6 (Architecture composition). *Let* $A_j = (\mathcal{C}_j, P_j, \gamma_j)$, *for* $j = 1, 2$ *be two architectures. The* composition *of* A_1 *and* A_2 *is an architecture* $A_1 \oplus A_2 = (\mathcal{C}_1 \cup \mathcal{C}_2, P_1 \cup P_2, \gamma_\varphi)$, *where* $\varphi = \varphi_{\gamma_1} \wedge \varphi_{\gamma_2}$.

The following lemma states that the interaction model of the composed component consists precisely of the interactions a such that the projections of a onto the interfaces of both of the composed architectures (A_1, A_2, resp.) belong to the corresponding interaction models (γ_1, γ_2 resp.). In other words, these are precisely the interactions that satisfy the coordination constraints enforced by both composed architectures.

Lemma 1. *Consider two interaction models $\gamma_i \subseteq 2^{P_i}$, for $i = 1, 2$, and let $\varphi = \varphi_{\gamma_1} \wedge \varphi_{\gamma_2}$. For an interaction $a \subseteq P_1 \cup P_2$, $a \in \gamma_\varphi$ iff $a \cap P_i \in \gamma_i$, for $i = 1, 2$.*

Proposition 1. *Architecture composition '$\oplus$' is commutative, associative and idempotent; $A_{id} = \big(\emptyset, \emptyset, \{\emptyset\}\big)$ is its neutral element, i.e. for any architecture A, we have $A \oplus A_{id} = A$. Furthermore, for any component B, we have $A_{id}(B) = B$.*

Notice that, for an arbitrary set of components $\mathcal{B}$ with $P = \bigcup_{B \in \mathcal{B}} P_B$, we have, by (1), $A_{id}(\mathcal{B}) = \big(2^P\big)(\mathcal{B})$ (cf. Def. 2).

Example 3 (Mutual exclusion (contd.)). Building upon Ex. 1, let B_3 be a third component, similar to B_1 and B_2, with the interface $\{b_3, f_3\}$. We define two additional architectures A_{13} and A_{23} similar to A_{12}: for $i = 1, 2$, $A_{i3} = (\{C_{i3}\}, P_{i3}, \gamma_{i3})$, where, up to the renaming of ports, C_{i3} is the same as C_{12} in Fig. 1(b), $P_{i3} = \{b_i, b_3, b_{i3}, f_i, f_3, f_{i3}\}$ and $\gamma_{i3} = \{\emptyset, b_i b_{i3}, b_3 b_{i3}, f_i f_{i3}, f_3 f_{i3}\}$.

By considering, for $\varphi_{\gamma_{13}}$ and $\varphi_{\gamma_{23}}$, expressions similar to (2), it is easy to compute $\varphi_{\gamma_{12}} \wedge \varphi_{\gamma_{13}} \wedge \varphi_{\gamma_{23}}$ as the conjunction of the following implications:

$$
\begin{array}{llllll}
b_1 \Rightarrow b_{12} \wedge b_{13}\,, & f_1 \Rightarrow f_{12} \wedge f_{13}\,, & b_{12} \Rightarrow b_1 \,\texttt{XOR}\, b_2\,, & f_{12} \Rightarrow f_1 \,\texttt{XOR}\, f_2\,, & b_{12} \Rightarrow \overline{f_{12}}\,, \\
b_2 \Rightarrow b_{12} \wedge b_{23}\,, & f_2 \Rightarrow f_{12} \wedge f_{23}\,, & b_{13} \Rightarrow b_1 \,\texttt{XOR}\, b_3\,, & f_{13} \Rightarrow f_1 \,\texttt{XOR}\, f_3\,, & b_{13} \Rightarrow \overline{f_{13}}\,, \\
b_3 \Rightarrow b_{13} \wedge b_{23}\,, & f_3 \Rightarrow f_{13} \wedge f_{23}\,, & b_{23} \Rightarrow b_2 \,\texttt{XOR}\, b_3\,, & f_{23} \Rightarrow f_2 \,\texttt{XOR}\, f_3\,, & b_{23} \Rightarrow \overline{f_{23}}\,.
\end{array}
$$

It is now straightforward to obtain the interaction model for $A_{12} \oplus A_{13} \oplus A_{23}$, i.e. $\{\emptyset,\ b_1 b_{12} b_{13},\ f_1 f_{12} f_{13},\ b_2 b_{12} b_{23},\ f_2 f_{12} f_{23},\ b_3 b_{13} b_{23},\ f_3 f_{13} f_{23}\}$. Notice that this is different from the union of the interaction models of the three architectures.

Assuming that the initial states of B_1, B_2 and B_3 are `sleep`, whereas those of C_{12}, C_{13} and C_{23} are `free`, one can observe that none of the states $(\cdot, \cdot, \cdot, \texttt{work}, \texttt{work}, \cdot)$, $(\cdot, \cdot, \cdot, \texttt{work}, \cdot, \texttt{work})$ and $(\cdot, \cdot, \cdot, \cdot, \texttt{work}, \texttt{work})$ are reachable in $(A_{12} \oplus A_{13} \oplus A_{23})(B_1, B_2, B_3)$. Thus, we conclude that the composition of the three architectures, $(A_{12} \oplus A_{13} \oplus A_{23})(B_1, B_2, B_3)$, enforces mutual exclusion among the `work` states of all three components. In Sect. 3.1, we provide a general result stating that architecture composition preserves the enforced state properties.

2.3 Hierarchical Composition of Architectures

Proposition 2. *Let $\mathcal{B}$ be a set of components and let $A_1 = (\mathcal{C}_1, P_{A_1}, \gamma_1)$ and $A_2 = (\mathcal{C}_2, P_{A_2}, \gamma_2)$ be two architectures, such that 1) $P_{A_1} \subseteq \bigcup_{B \in \mathcal{B} \cup \mathcal{C}_1} P_B$ and 2) $P_{A_2} \subseteq \bigcup_{B \in \mathcal{B} \cup \mathcal{C}_1 \cup \mathcal{C}_2} P_B$. Then $A_2(A_1(\mathcal{B}))$ is defined and equal to $(A_1 \oplus A_2)(\mathcal{B})$.*

Condition 1 states that A_1 can be applied to the components in $\mathcal{B}$ (cf. Def. 4); condition 2 states that A_2 can be applied to $A_1(\mathcal{B})$. Note that, when $P_{A_i} \subseteq \bigcup_{B \in \mathcal{B} \cup \mathcal{C}_i} P_B$ holds for both $i \in \{1,2\}$—for $i = 1$, this is the condition 1—and none of the architectures involves the ports of the other, i.e. $P_{A_i} \cap \bigcup_{C \in \mathcal{C}_j} P_C = \emptyset$, for $i \neq j \in \{1,2\}$, then the two architectures are independent and their composition is commutative: $A_2(A_1(\mathcal{B})) = (A_1 \oplus A_2)(\mathcal{B}) = A_1(A_2(\mathcal{B}))$.

Proposition 3. *Let $\mathcal{B}_1, \mathcal{B}_2$ be two sets of components, such that $\bigcup_{B \in \mathcal{B}_1} P_B \cap \bigcup_{B \in \mathcal{B}_2} P_B = \emptyset$. Let $A_1 = (\mathcal{C}_1, P_{A_1}, \gamma_1)$ and $A_2 = (\mathcal{C}_2, P_{A_2}, \gamma_2)$ be two architectures, such that $P_{A_1} \subseteq \bigcup_{B \in \mathcal{B}_1 \cup \mathcal{C}_1} P_B$ and $P_{A_2} \subseteq \bigcup_{B \in \mathcal{B}_1 \cup \mathcal{B}_2 \cup \mathcal{C}_1 \cup \mathcal{C}_2} P_B$. Then $A_2(A_1(\mathcal{B}_1, \mathcal{B}_2)) = A_2(A_1(\mathcal{B}_1), \mathcal{B}_2)$.*

Intuitively, Prop. 3 states that one only has to apply the architecture A_1 to those components that have ports involved in its interface. Notice that, in order to compare the semantics of two sets of components, one has to compose them into compound components, by applying *some* architecture. Hence the need for A_2 in Prop. 3. As a special case, one can consider the "most liberal" identity architecture A_{id} (see Prop. 1). A_{id} does not impose any coordination constraints, allowing all possible interactions between the components it is applied to.

Example 4 (Mutual exclusion (contd.)). Ex. 3 can be generalised to an arbitrary number n of components. However, this solution requires $n(n-1)/2$ architectures, and so does not scale well. Instead, we apply architectures hierarchically.

Let $n = 4$ and consider two architectures A_{12}, A_{34}, with the respective coordination components C_{12}, C_{34}, that respectively enforce mutual exclusion between B_1, B_2 and B_3, B_4 as in Ex. 3. Assume furthermore, that an architecture A enforces mutual exclusion between the `taken` states of C_{12} and C_{34}. It is clear that the system $A(A_{12}(B_1, B_2), A_{34}(B_3, B_4))$ ensures mutual exclusion between all four components $(B_i)_{i=1}^4$. Furthermore, by the above propositions,

$$A(A_{12}(B_1, B_2), A_{34}(B_3, B_4)) = A(A_{12}(B_1, B_2, A_{34}(B_3, B_4))) = \\ A(A_{12}(A_{34}(B_1, B_2, B_3, B_4))) = (A \oplus A_{12} \oplus A_{34})(B_1, B_2, B_3, B_4)\,.$$

3 Property Preservation

Throughout this section we use several classical notions, which we recall here.

Definition 7 (Paths, path fragments, and reachable states). *Let $B = (Q, q^0, P, \to)$ be a component. A finite or infinite sequence $q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \cdots \xrightarrow{a_{i-1}} q_{i-1} \xrightarrow{a_i} q_i \cdots$ is a* path *in B if $q_0 = q^0$, otherwise it is a* path fragment. *A state $q \in Q$ is* reachable *iff there exists a finite path in B terminating in q.*

3.1 Safety Properties

Definition 8 (Properties and invariants). *Let $B = (Q, q^0, P, \to)$ be a component. A* property *of B is a state predicate $\Phi : Q \to \mathbb{B}$. Φ is* initial *if $\Phi(q^0) = \mathtt{tt}$.*

Definition 9 (Enforcing properties). *Let $A = (\mathcal{C}, P_A, \gamma)$ be an architecture; let $\mathcal{B}$ be a set of components and Φ be an initial property of their parallel composition $A_{id}(\mathcal{B})$ (see Prop. 1). We say that A* enforces *Φ on $\mathcal{B}$ iff, for every state $q = (q_b, q_c)$ reachable in $A(\mathcal{B})$, with $q_b \in \prod_{B\in\mathcal{B}} Q_B$ and $q_c \in \prod_{C\in\mathcal{C}} Q_C$, we have $\Phi(q_b) = \mathtt{tt}$.*

Example 5. Consider again mutual exclusion in Ex. 1. The component $A_{12}(B_1, B_2)$ is shown in Fig. 2 (we abbreviate `sleep`, `work`, `free` and `taken` to `s`, `w`, `f` and `t` respectively). Clearly A_{12} enforces on $\{B_1, B_2\}$ the property $\Phi_{12} = (q_1 \neq \mathtt{w}) \vee (q_2 \neq \mathtt{w})$, where q_1 and q_2 are state variables of B_1 and B_2 respectively.

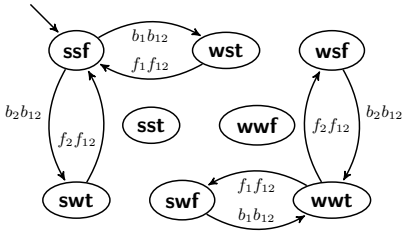
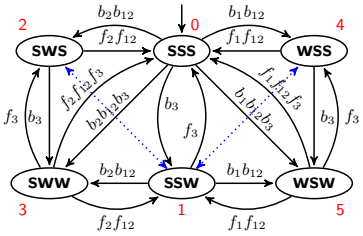


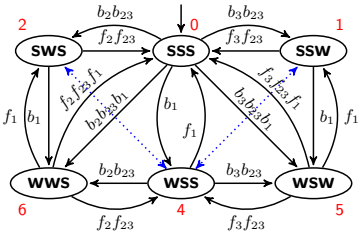
Fig. 2. Component from Ex. 5

According to the above definition, when we say that an architecture enforces some property Φ, it is implicitly assumed that Φ is initial for the coordinated components. Below, we omit mentioning this explicitly.

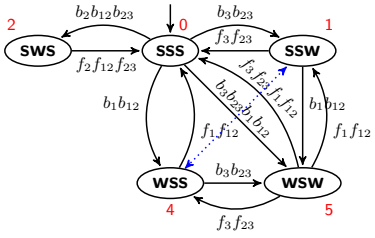
Theorem 1 (Preserving enforced properties). *Let $\mathcal{B}$ be a set of components; let A_1 and A_2 be two architectures enforcing on $\mathcal{B}$ the properties Φ_1 and Φ_2 respectively. The composition $A_1 \oplus A_2$ enforces on $\mathcal{B}$ the property $\Phi_1 \wedge \Phi_2$.*



(a) $A_{12}(B_1, B_2, B_3)$



(b) $A_{23}(B_1, B_2, B_3)$



(c) $(A_{12} \oplus A_{23})(B_1, B_2, B_3)$

For ease of reading, we omit the transitions indicated in blue and additionally label each state with a red number, whereof the main label is the binary representation with $\mathtt{s} = 0$ and $\mathtt{w} = 1$.

Fig. 3. Projections of reachable states of Ex. 6 components onto $A_{id}(B_1, B_2, B_3)$

Example 6. In the context of Ex. 3, consider the application of architectures A_{12} and A_{23} to the components B_1, B_2 and B_3. The former enforces the property $\Phi_{12} = (q_1 \neq \mathtt{w}) \vee (q_2 \neq \mathtt{w})$ (the projections of reachable states of $A_{12}(B_1, B_2, B_3)$

onto the state-space of the atomic components are shown in Fig. 3(a)), whereas the latter enforces $\Phi_{23} = (q_2 \neq \texttt{w}) \vee (q_3 \neq \texttt{w})$ (the projections of reachable states of $A_{23}(B_1, B_2, B_3)$ onto the state-space of the atomic components are shown in Fig. 3(b)). By Th. 1, the composition $A_{12} \oplus A_{23}$ enforces $\Phi_{12} \wedge \Phi_{23} = (q_2 \neq \texttt{w}) \vee \big((q_1 \neq \texttt{w}) \wedge (q_3 \neq \texttt{w})\big)$, i.e. mutual exclusion between, on one hand, the `work` state of B_2 and, on the other hand, the `work` states of B_1 and B_3 (see Fig. 3(c)). Mutual exclusion between the `work` states of B_1 and B_3 is not enforced. Furthermore, it is easy to check that $A_1 \oplus A_2 \oplus A_3$ enforces mutual exclusion between the `work` states of B_1, B_2 and B_3 as $\Phi_{12} \wedge \Phi_{13} \wedge \Phi_{23} = \big((q_1 \neq \texttt{w}) \wedge (q_2 \neq \texttt{w})\big) \vee \big((q_1 \neq \texttt{w}) \wedge (q_3 \neq \texttt{w})\big) \vee \big((q_1 \neq \texttt{w}) \wedge (q_3 \neq \texttt{w})\big)$.

3.2 Liveness Properties

Our treatment of liveness properties is based on the idea that each coordinator C must be "invoked sufficiently often", so that the liveness properties inherent in C are imposed on the system as a whole. So, what does sufficiently often mean? A reasonable initial idea is to require that each controller is executed infinitely often (along an infinite path). But that turns out to be too strong. For example, a mutual exclusion controller should not be invoked infinitely often if no process that it controls requests the critical resource. So, we add "idle states", so that it is permitted for a coordinator to remain forever in an idle state. A coordinator not in an idle state must eventually be executed. We will use the equivalent formulation: an (infinite) path is live with respect to a coordinator C iff either C is executed infinitely often, or is in an idle state infinitely often. A live path is one that is live with respect to all coordinators. An architecture A is live with respect to a set of components $\mathcal{B}$ iff every finite path of $A(\mathcal{B})$ can be extended to an infinite live one.

A transition $q \xrightarrow{a} q'$ *executes a coordinator* C iff $a \cap P_C \neq \emptyset$. An infinite path α executes C infinitely often iff α contains an infinite number of transitions that execute C. An infinite path $q_0 \xrightarrow{a_1} q_1 \cdots$ *visits an idle state* of coordinator C infinitely often iff, for infinitely many $i \geq 0$, $q_i \restriction C$ (the state component of C in q_i) is an idle state of C.

Definition 10 (Architecture with liveness conditions). *An* architecture with liveness conditions *is a tuple* $A = (\mathcal{C}, P_A, \gamma)$*, where* $\mathcal{C}$ *is a set of coordinating components with liveness condition,* P_A *is a set of ports, such that* $\bigcup_{C \in \mathcal{C}} P_C \subseteq P_A$*, and* $\gamma \subseteq 2^{P_A}$ *is an interaction model. A* coordinating component with liveness condition *is* $C = (Q, q^0, Q_{idle}, P_C, \rightarrow)$*, where* $(Q, q^0, P_C, \rightarrow)$ *is a component (Def. 1) and* $Q_{idle} \subseteq Q$*.*

Hence, we augment each coordinator with a *liveness condition*: a subset Q_{idle} of its states Q, which are considered "idle", and in which it can remain forever without violating liveness.

Definition 11 (Live path). *Let* $A = (\mathcal{C}, P_A, \gamma)$ *be an architecture with liveness conditions and* $\mathcal{B}$ *a set of components. An infinite path* α *in* $A(\mathcal{B})$ *is* live *iff, for every* $C \in \mathcal{C}$*,* α *contains infinitely many occurrences of interactions containing*

some port of C, or α contains infinitely many states whose projection onto C is an idle state of C.

That is, if $\alpha \stackrel{\Delta}{=} q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \cdots \xrightarrow{a_i} q_i \cdots$ then, for every $C \in \mathcal{C}$, for infinitely many i: $a_i \cap P \neq \emptyset$ or $q_i \restriction C \in Q_{idle}$, where $C = (Q, q^0, Q_{idle}, P, \rightarrow)$, and $q_i \restriction C$ denotes the local state of C in q_i.

The intuition behind this definition is that each liveness condition guarantees that its coordinator executes "sufficiently often", i.e. infinitely often unless it is in an idle state. When architectures are composed, we take the union of all the coordinators. Since each coordinator carries its liveness condition with it, we obtain that each coordinator is also executed sufficiently often in the composed architecture. We also obtain that architecture composition is as before, i.e. we use Def. 6, with the understanding that we compose two architectures with liveness conditions. For the rest of this section, we use "architecture" to mean "architecture with liveness conditions".

When we apply an architecture with liveness conditions to a set of components, thereby obtaining a system, we need the notion of machine closure [7]: every finite path can be extended to a live one.

Definition 12 (Live w.r.t. a set of components). *Let A be an architecture with liveness conditions and $\mathcal{B}$ be a set of components. A is* live *w.r.t. $\mathcal{B}$ iff every finite path in $A(\mathcal{B})$ can be extended to a live path.*

Even if $A_1, \ldots, A_m$ are each live w.r.t. $\mathcal{B}$, it is still possible for $(A_1 \oplus \cdots \oplus A_m)(\mathcal{B})$ to be not live w.r.t. $\mathcal{B}$, due to "interference" between the coordinators of the A_i. For example, consider two architectures that enforce mutually inconsistent scheduling policies, e.g. they require two conflicting interactions (those that share a component) to both be executed. Hence, we define a notion of "noninterference" which guarantees that $(A_1 \oplus \cdots \oplus A_m)(\mathcal{B})$ is live w.r.t. $\mathcal{B}$.

A system is free of *global deadlock* iff, in every reachable global state, there is at least one enabled interaction. We show in [8] how to verify that a system is free of global deadlock, using a sufficient but not necessary condition that, in many cases, can be evaluated quickly, without state-explosion. Essentially, we check, for every interaction a in the system, that the execution of a cannot possibly lead to a deadlock state. The check can often be discharged within a "small subsystem," which contains all of the components that participate in a.

We now give a criterion for liveness that can be evaluated without state-explosion w.r.t. the number of architectures. For simplicity, we assume in the sequel that each architecture A_i has exactly one coordinating component C_i.

Definition 13 (Noninterfering live architectures). *Let architectures $A_i = (\{C_i\}, P_{A_i}, \gamma_i)$, for $i = 1, 2$ be live w.r.t. a set of components $\mathcal{B}$. Then A_1 is* noninterfering *with respect to A_2 and components $\mathcal{B}$ iff, for every infinite path α in $(A_1 \oplus A_2)(\mathcal{B})$ which executes C_1 infinitely often: either α executes C_2 infinitely often or α visits an idle state of C_2 infinitely often.*

A set of architectures $A_i = (\{C_i\}, P_{A_i}, \gamma_i)$, for $i \in \{1, \ldots, m\}$ is *pairwise-noninterfering* w.r.t. components $\mathcal{B}$, iff for all $j, k \in \{1, \ldots, m\}, j \neq k$: A_j is noninterfering w.r.t A_k and components $\mathcal{B}$.

Theorem 2 (Pairwise noninterfering live architectures). *Let architectures* $A_i = (\{C_i\}, P_{A_i}, \gamma_i)$*, for* $i \in \{1, \ldots, m\}$ *be live and pairwise-noninterfering w.r.t. a set of components* $\mathcal{B}$*. Assume also that* $(\bigoplus_{i=1}^{m} A_i)(\mathcal{B})$ *is free of global deadlock. Then* $(\bigoplus_{i=1}^{m} A_i)$ *is live w.r.t.* $\mathcal{B}$*.*

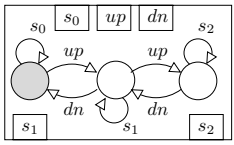
Example 7 (Noninterference in mutual exclusion). Consider the system $(A_{12} \oplus A_{23} \oplus A_{13})(B_1, B_2, B_3)$, as in Ex. 3. Let each coordinator have a single idle state, namely the `free` state. Consider the applications of each pair of coordinators, i.e. $(A_{12} \oplus A_{23})(B_1, B_2, B_3)$, $(A_{23} \oplus A_{13})(B_1, B_2, B_3)$ and $(A_{12} \oplus A_{13})(B_1, B_2, B_3)$. For $(A_{12} \oplus A_{23})(B_1, B_2, B_3)$, we observe that along any infinite path, either C_{12} executes infinitely often, or remains forever in its idle state after some point. Hence A_{23} is noninterfering w.r.t. A_{12} and B_1, B_2, B_3. Likewise for the five other ordered pairs of coordinators. We verify that $(A_{12} \oplus A_{23} \oplus A_{13})(B_1, B_2, B_3)$ is free from global deadlock using the method of [8]. Hence by Th. 2, we conclude that $(A_{12} \oplus A_{23} \oplus A_{13})$ is live w.r.t. $\{B_1, B_2, B_3\}$.

For a finite-state system $(A_1 \oplus A_2)(\mathcal{B})$, we verify noninterference by checking for infinite paths along which C_1 (the coordinator of A_1) executes forever, while C_2 (the coordinator of A_2) does not execute and is not in an idle state. Our algorithm is: (1) generate the state-transition graph M_{12} of $(A_1 \oplus A_2)(\mathcal{B})$, by starting with the initial state and repeatedly applying all enabled interactions until there is no further change; (2) let M'_{12} result from M_{12} by removing all transitions of C_2 and all global states (and their incident transitions) whose C_2-component is an idle state of C_2; (3) find all non-trivial maximal strongly connected components of M'_{12}, if any. A strongly connected component is nontrivial if it is either a single state with a self-loop, or it contains at least two states. Existence of such a component CC of M'_{12} certifies the existence of an infinite path along which C_1 executes forever, while C_2 does not execute and is not in an idle state. Conversely, the non-existence of such a CC certifies that A_1 is noninterfering w.r.t. A_2 and $\mathcal{B}$ (Def. 13).

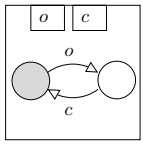
Let $|M_{12}|$ denote the number of nodes (states) plus the number of edges (transitions) in M_{12}. Let $|\gamma_1 \cup \gamma_2|$ denote the number of interactions in $|(A_1 \oplus A_2)(\mathcal{B})|$. Step (1) takes time $O(|M_{12}| * |\gamma_1 \cup \gamma_2|)$, since every interaction is checked in every state. Step (2) takes time $O(|M_{12}|)$, since it can be implemented using a depth-first (or breadth-first) search of M_{12}. Step (3) takes time $O(|M_{12}|)$, using [9]. Hence our algorithm has time complexity $O(|M_{12}| * |\gamma_1 \cup \gamma_2|)$.

4 Case Study: Control of an Elevator Cabin

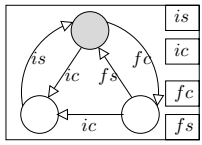
We illustrate our results with the Elevator case study adapted from the literature [10, 11], for a building with three floors. Control of the elevator cabin is modelled as a set of coordinated atomic components shown in Fig. 4. Each floor of the building has a separate caller system, which allows floor selection inside the elevator and calling from the floor. Ports *ic* and *fc* respectively represent calls made within the elevator and calls from a floor. Ports *is* and *fs* represent cabin stops in response to these calls. Furthermore, port names, *m*, *c*, *o*, *s*, *dn*, *up*, *nf*,



(a) Elevator Engine E



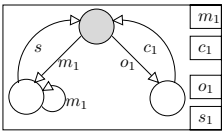
(b) Elevator Door D



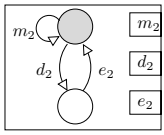
(c) Caller System CS

Fig. 4. Elevator atomic components

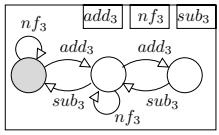
fn and fr stand respectively for "move", "call", "open", "stop", "move down", "move up", "not full", "finish" and "free". Caller system components and their ports are indexed by floor numbers. $\mathcal{B} = \{E, D, CS_0, CS_1, CS_2\}$ denotes the set of atomic components.



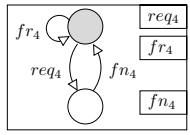
(a) C_1



(b) C_2



(c) C_3



(d) C_4

Fig. 5. Coordinating components for the elevator example

To enforce required properties, we successively apply and compose architectures. First, apply architecture $A_1 = (\{C_1\}, P_1, \gamma_1)$ to $\mathcal{B}$. C_1 is shown in Fig. 5(a). P_1 contains all ports of C_1 and all ports of $\mathcal{B}$. γ_1 comprises the empty interaction $\emptyset$ and the following interactions (for $i \in [0, 2]$):

- Door control: $\{\{o, o_1\}, \{c, c_1\}\}$
- Floor selection control: $\{\{fc_i\}, \{ic_i\}\}$
- Moving control: $\{\{s_i, s, fs_i\}, \{s_i, s, is_i\}, \{up, m_1\}, \{dn, m_1\}\}$

System $A_1(\mathcal{B})$ provides basic elevator functionality, i.e., moving up and down, stopping only at the requested floors, and door control. Architecture A_1 enforces the safety property: *the elevator does not move with open doors*. Nonetheless, $A_1(\mathcal{B})$ allows the elevator to stop at a floor, and then to leave without having opened the door. To prevent this, we apply architecture $A_2 = (\{C_2\}, P_2, \gamma_2)$ where C_2 is shown in Fig. 5(b), $P_2 = \{e_2, d_2, m_2, c_1, s, m_1\}$, and $\gamma_2 = \{\emptyset, \{s, d_2\}, \{c_1, e_2\}, \{m_1, m_2\}\}$. This grants priority to the door controller after an s action. By Prop. 2, $A_2(A_1(\mathcal{B})) = (A_1 \oplus A_2)(\mathcal{B})$. $(A_2 \oplus A_1)(\mathcal{B})$ provides the same functionality as $A_1(\mathcal{B})$, and also this additional property. The property *"if the elevator is full, it must stop only at floors selected from the cabin and ignore outside calls"* [10, 11], is enforced by applying architecture $A_3 = (\{C_3\}, P_3, \gamma_3)$ with C_3 shown in Fig. 5(c), $P_3 = \{add_3, sub_3, nf_3, s, fs_i \mid i \in [0, 2]\}$ and $\gamma_3 = \{\emptyset, \{add_3\}, \{sub_3\}\} \cup \{\{s, nf_3, fs_i\} \mid i \in [0, 2]\}$. A elevator is full in our example if it contains two passengers. By Prop. 2, $A_3((A_1 \oplus A_2)(\mathcal{B})) = (A_1 \oplus A_2 \oplus A_3)(\mathcal{B})$. By Th. 1, $(A_1 \oplus A_2 \oplus A_3)(\mathcal{B})$ satisfies all three properties.

We specify liveness properties for $(A_1 \oplus A_2 \oplus A_3)(\mathcal{B})$ by choosing idle states for the coordinators. C_1 and C_2 have only their initial states idle, since a moving

elevator must eventually stop, and an open door must eventually close. C_3 has all of its states idle, since C_3 enforces a pure safety property. We implemented our algorithm for checking noninterference, and used the implementation to verify that C_1 and C_2 are mutually noninterfering w.r.t. to $\mathcal{B}$. Our implementation showed, however, that C_3 interferes with both C_1 and C_2, since it allows an infinite sequence of add_3 and sub_3 interactions. This reflects the absence of an environment component: in reality, one assumes that clients will not hold up the elevator indefinitely by continuously moving in and out. This shows that we can detect shortcomings in component models w.r.t. liveness: they manifest as violations of noninterference.

Finally, we consider the additional property: *"requests from the second floor have priority over all other requests"* [10, 11]. This is enforced by the architecture $A_4 = (\{C_4\}, P_4, \gamma_4)$ with C_4 shown in Fig. 5(d). P_4 consists of ports of C_4, CS_2, and o, s and dn; $\gamma_4 = \{\emptyset, \{fc_2, req_4\}, \{ic_2, req_4\}, \{o, fr_4\}, \{dn, fr_4\}, \{fs_2, fn_4\}, \{is_2, fn_4\}\}$. The system obtained by application of A_4 to $(A_1 \oplus A_2 \oplus A_3)(\mathcal{B})$ has a local deadlock, which was detected by using the deadlock analysis tool presented in [8]. This deadlock occurs when a full elevator is called from the second floor. In fact, A_4 enforces the constraint of not going down, while A_3 forbids stopping at this floor. Thus, the only choice is to move upward, which is impossible. Hence the system is in a local deadlock state involving the elevator engine.

$(A_1 \oplus A_2 \oplus A_4)(\mathcal{B})$, obtained by applying A_4 to $(A_1 \oplus A_2)(\mathcal{B})$, is verified to be deadlock-free, using [8]. $\{A_1, A_2, A_4\}$ are pairwise-noninterfering w.r.t. $\mathcal{B}$, using our implementation. So by Th. 2, $(A_1 \oplus A_2 \oplus A_4)$ is live w.r.t. $\mathcal{B}$.

5 Related Work

A number of paradigms for unifying component composition have been studied in [12–14]. These achieve unification by reduction to a common low-level semantic model. Coordination mechanisms and their properties are not studied independently of behaviour. This is also true for the numerous compositional and algebraic frameworks [15–23]. Most of these frameworks are based on a single operator for concurrent composition. This entails poor expressiveness, which results in overly complex architectural designs. In contrast, BIP allows expression of general multiparty interaction and strictly respects separation of concerns. Coordination can be studied as a separate entity that admits a simple Boolean characterisation that is instrumental for expressing composability.

BIP has some similarities with CSP, which can directly express multiparty interaction by using composition operators parameterized by channel names. For example, $B|\{a\}|B'$ is the system that enforces synchronisation of a-actions of components B and B'. Nonetheless, CSP is not adequate for architecture composition as the components must be modified when additional architecture constraints are applied. Consider for example the components $B_i = a_i \rightarrow STOP$ for $i = 1, 2, 3$. To model the system described in BIP by $\{\{a_1, a_2\}, \{a_2, a_3\}\}\{B_1, B_2, B_3\}$, two channels α and β must be defined representing respectively interactions $\{a_1, a_2\}$

and $\{a_2, a_3\}$ and the components modified as follows: $B_1 = \alpha \rightarrow STOP$, $B_2 = \alpha \rightarrow STOP \,\Box\, \beta \rightarrow STOP, B_3 = \beta \rightarrow STOP$. That is, in addition to renaming, B_2 must be modified to show explicitly the conflict between α and β.

Existing research on architecture composability deals mainly with resource composability for particular types of architectures, e.g. [23]. The feature interaction problem is how to rapidly develop and deploy new features without disrupting the functionality of existing features. It can be considered as an architecture composability problem to the extent that features can be modelled as architectural constraints. A survey on feature interaction research is provided in [1]. Existing results focus mainly on modelling aspects and checking feature interaction by using algorithmic verification techniques with well-known complexity limitations. Our work takes a constructive approach. It has some similarities to [24] which presents a formal framework for detecting and avoiding feature interactions by using priorities. Nonetheless, these results do not deal with property preservation through composition. Similarly, existing work on service interaction mainly focuses on modelling and verification aspects, e.g. [25, 26].

6 Conclusion

Our work makes two novel contributions towards correct-by-construction system design. First, it proposes a general concept of architecture. Architectures are operators restricting the behaviour of their arguments by enforcing a characteristic property. They can be composed and studied independently. Composition of architectures can be naturally expressed as the conjunction of the induced synchronisation constraints. This implies nice properties such as associativity, commutativity and idempotence. Nonetheless, it is not easy to understand it as an operation on interaction models. Using BIP to describe architectures proves to be instrumental for achieving this. In contrast to other formalisms, BIP is expressive enough and keeps a strict separation between behaviour and coordination aspects. *Application of architectures does not require any modification of the atomic components.* The second contribution is preservation of properties enforced by architectures. The preservation of state predicates is guaranteed by the very nature of architecture composition. This result is different from existing results stipulating the preservation of invariants of components when composed by using parallel composition operators e.g., an invariant of B_1 is also an invariant of $B_1||B_2$ for some parallel composition operator $||$. Our result is about preservation of properties over the *same* state-space, which is the Cartesian product of the atomic components. That is, a property of $A_1(\mathcal{B})$ is also a property of $(A_1 \oplus A_2)(\mathcal{B})$, and so the state-space of the components $\mathcal{B}$ is unchanged.

Our work pursues similar objectives as the research on interaction of features or services, insofar as they can be modelled as architectural constraints. Nonetheless, it adopts a radically different approach. It privileges constructive techniques to avoid costly and intractable verification. It proposes a concept of composability focusing on property preservation.

References

1. Calder, M., Kolberg, M., Magill, E.H., Reiff-Marganiec, S.: Feature interaction: a critical review and considered forecast. Computer Networks 41(1), 115–141 (2003)
2. Sifakis, J.: Rigorous System Design. Foundations and Trends in Electronic Design Automation 6(4), 293–362 (2012)
3. Bliudze, S., Sifakis, J.: Synthesizing glue operators from glue constraints for the construction of component-based systems. In: Apel, S., Jackson, E. (eds.) SC 2011. LNCS, vol. 6708, pp. 51–67. Springer, Heidelberg (2011)
4. Basu, A., Bensalem, S., Bozga, M., Combaz, J., Jaber, M., Nguyen, T.H., Sifakis, J.: Rigorous component-based system design using the BIP framework. IEEE Software 28(3), 41–48 (2011)
5. Wegner, P.: Coordination as constrained interaction (extended abstract). In: Ciancarini, P., Hankin, C. (eds.) COORDINATION 1996. LNCS, vol. 1061, pp. 28–33. Springer, Heidelberg (1996)
6. Bliudze, S., Sifakis, J.: Causal semantics for the algebra of connectors. Formal Methods in System Design 36(2), 167–194 (2010)
7. Abadi, M., Lamport, L.: Composing specifications. ACM Trans. Program. Lang. Syst. 15(1), 73–132 (1993)
8. Attie, P.C., Bensalem, S., Bozga, M., Jaber, M., Sifakis, J., Zaraket, F.A.: An abstract framework for deadlock prevention in BIP. In: Beyer, D., Boreale, M. (eds.) FMOODS/FORTE 2013. LNCS, vol. 7892, pp. 161–177. Springer, Heidelberg (2013)
9. Tarjan, R.: Depth-first search and linear graph algorithms. SIAM Journal on Computing 1(2), 146–160 (1972)
10. D'Souza, D., Gopinathan, M.: Conflict-tolerant features. In: Gupta, A., Malik, S. (eds.) CAV 2008. LNCS, vol. 5123, pp. 227–239. Springer, Heidelberg (2008)
11. Plath, M., Ryan, M.: Feature integration using a feature construct. Science of Computer Programming 41(1), 53–84 (2001)
12. Balarin, F., Watanabe, Y., Hsieh, H., Lavagno, L., Passerone, C., Sangiovanni-Vincentelli, A.: Metropolis: an integrated electronic system design environment. IEEE Computer 36(4), 45–52 (2003)
13. Balasubramanian, K., Gokhale, A., Karsai, G., Sztipanovits, J., Neema, S.: Developing applications using model-driven design environments. IEEE Computer 39(2), 33–40 (2006)
14. Eker, J., Janneck, J., Lee, E., Liu, J., Liu, X., Ludvig, J., Neuendorffer, S., Sachs, S., Xiong, Y.: Taming heterogeneity: The Ptolemy approach. Proceedings of the IEEE 91(1), 127–144 (2003)
15. Arbab, F.: Reo: a channel-based coordination model for component composition. Mathematical Structures in Computer Science 14(3), 329–366 (2004)
16. Fiadeiro, J.L.: Categories for Software Engineering. Springer (April 2004)
17. Ray, A., Cleaveland, R.: Architectural interaction diagrams: AIDs for system modeling. In: ICSE 2003: Proceedings of the 25th International Conference on Software Engineering, pp. 396–406. IEEE Computer Society, Washington, DC (2003)
18. Spitznagel, B., Garlan, D.: A compositional formalization of connector wrappers. In: ICSE, pp. 374–384. IEEE Computer Society (2003)
19. Bernardo, M., Ciancarini, P., Donatiello, L.: On the formalization of architectural types with process algebras. In: SIGSOFT FSE, pp. 140–148 (2000)
20. Bruni, R., Lanese, I., Montanari, U.: A basic algebra of stateless connectors. Theoretical Computer Science 366(1), 98–120 (2006)

21. Hoare, C.A.R.: Communicating Sequential Processes. Prentice Hall International Series in Computer Science. Prentice Hall (April 1985)
22. Milner, R.: Communication and Concurrency. Prentice Hall International Series in Computer Science. Prentice Hall (1989)
23. Liu, I., Reineke, J., Lee, E.A.: A PRET architecture supporting concurrent programs with composable timing properties. In: 2010 Conference Record of the Forty Fourth Asilomar Conference on Signals, Systems and Computers (ASILOMAR), pp. 2111–2115 (2010)
24. Hay, J.D., Atlee, J.M.: Composing features and resolving interactions. SIGSOFT Softw. Eng. Notes 25(6), 110–119 (2000)
25. Decker, G., Puhlmann, F., Weske, M.: Formalizing service interactions. In: Dustdar, S., Fiadeiro, J.L., Sheth, A.P. (eds.) BPM 2006. LNCS, vol. 4102, pp. 414–419. Springer, Heidelberg (2006)
26. Li, Z., Jin, Y., Han, J.: A runtime monitoring and validation framework for web service interactions. In: ASWEC, pp. 70–79 (2006)