Configuration Logics: Modelling Architecture Styles

Anastasia Mavridou, Eduard Baranov, Simon Bliudze, and Joseph Sifakis

École polytechnique fédérale de Lausanne, Station 14, 1015 Lausanne, Switzerland;
firstname.lastname@epfl.ch

Abstract. We study a framework for the specification of architecture styles as families of architectures involving a common set of types of components and coordination mechanisms. The framework combines two logics: 1) interaction logics for the specification of architectures as generic coordination schemes involving a configuration of interactions between typed components; 2) configuration logics for the specification of architecture styles as sets of interaction configurations. The presented results build on previous work on architecture modelling in BIP. We show how propositional interaction logic can be extended into a corresponding configuration logic by adding new operators on sets of interaction configurations. We provide a complete axiomatisation of the propositional configuration logic, as well as a decision procedure for checking that an architecture satisfies given logical specifications. To allow genericity of specifications, we study first-order and second-order extensions of the propositional configuration logic. We provide examples illustrating the application of the results to the characterization of architecture styles. Finally, we provide an experimental evaluation using the Maude rewriting system to implement the decision procedure for the propositional logic.

1 Introduction

Architectures are common means for organizing coordination between components in order to build complex systems and to make them manageable. They depict generic coordination principles between components and embody design rules that can be understood by all. Architectures allow thinking on a higher plane and avoiding low-level mistakes. They are a means for ensuring global coordination properties between components and thus, achieving correctness by construction [1]. Using architectures largely accounts for our ability to master complexity and develop systems cost-effectively. System developers extensively use reference architectures ensuring both functional and non-functional properties, e.g. fault-tolerant, time-triggered, adaptive, security architectures.
Informally architectures are characterized by the structure of the interactions between a set of typed components. The structure is usually specified as a relation, e.g. connectors between component ports.
Architecture styles characterize not a single architecture but a family of architectures sharing common characteristics such as the type of the involved components and the topology induced by their coordination

structure. Simple examples of architecture styles are Pipeline, Ring, Master/Slave, Pipe and Filter. For instance, Master/Slave architectures integrate two types of components, masters and slaves, such that each slave can interact only with one master. Figure 1 depicts four Master/Slave architectures involving master components M_1, M_2 and slave components S_1, S_2. Their communication ports are respectively m_1, m_2 and s_1, s_2. The architectures correspond to interaction configurations: $\{\{s_1, m_1\}, \{s_2, m_2\}\}$, $\{\{s_1, m_1\}, \{s_2, m_1\}\}$, $\{\{s_1, m_2\}, \{s_2, m_1\}\}$ and $\{\{s_1, m_2\}, \{s_2, m_2\}\}$. The set $\{s_i, m_j\}$ denotes an interaction between ports s_i and m_j. A configuration is a non-empty set of interactions. The Master/Slave architecture style characterizes all the Master/Slave architectures for arbitrary numbers of masters and slaves.

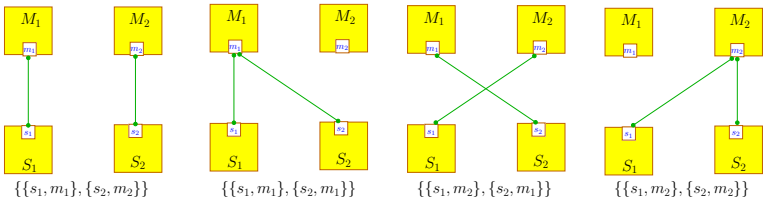


Fig. 1: Master/Slave architectures

The paper studies the relation between architectures and architecture styles. This relation is similar to the relation between programs and their specifications. As program specifications can be expressed by using logics, e.g. temporal logics, architecture styles can be specified by configuration logics characterizing classes of architectures.

First, we propose a propositional configuration logic whose formulas represent, for a given set of components, the allowed configuration sets. Then, we introduce first-order and second-order logics as extensions of the propositional logic. These allow genericity of description as they are defined for types of components.

The meaning of a configuration logic formula is a configuration set. A configuration on a set of components represents a particular architecture. Defining configuration logics requires considering three hierarchically structured semantic domains:

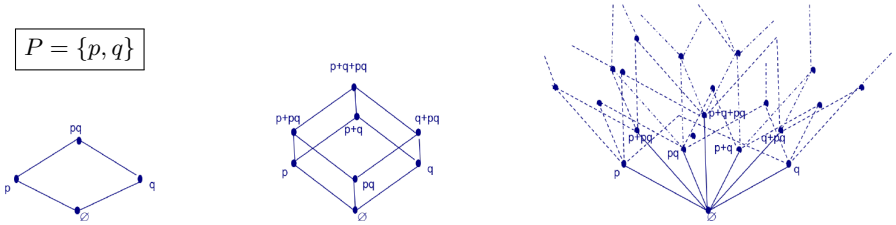
The lattice of interactions. An interaction a is a non-empty subset of P, the set of ports of the integrated components. Its execution implies the atomic synchronization of all component actions (at most one action per component) associated with the ports of a.

The lattice of configurations. Configurations are non-empty sets of interactions characterizing architectures.

The lattice of configuration sets. Sets of configurations are properties described by the configuration logic.

Figure 2 shows the three lattices for $P = \{p, q\}$. For the lattice of configuration sets, we show only how it is generated.

This work consistently extends results on modelling architectures by using propositional interaction logic [2–4], which are Boolean algebras on the set of ports P of the composed components. Their semantics is defined via a satisfaction relation between interactions and formulas. An



(a) $I(P) = 2^P$ (b) $C(P) = 2^{I(P)\setminus\{\emptyset\}}$ (c) $CS(P) = 2^{C(P)\setminus\{\emptyset\}}$

Fig. 2: Lattices of interactions (a), configurations (b) and configuration sets (c).

interaction $a \subseteq P$ satisfies a formula ϕ (we write $a \models_i \phi$) if ϕ evaluates to *true* for the valuation that assigns *true* to the ports belonging to a and *false* otherwise. It is characterized exactly by the formula $\bigwedge_{p\in a} p \wedge \bigwedge_{p\notin a} \overline{p}$.

Configuration logic is a powerset extension of the interaction logic. Its formulas are generated from the formulas of the propositional interaction logic by using the operators union, intersection and complementation as well as a *coalescing operator* $+$. To avoid ambiguity, we refer to the formulas of the configuration logic that syntactically are also formulas of the interaction logics as *interaction formulas*. The semantics of the configuration logic is defined via a satisfaction relation $\models$ between configurations $\gamma = \{a_1, ..., a_n\}$ and formulas. An interaction formula f represents any configuration consisting of interactions satisfying it; that is $\gamma \models f$ if, for all $a \in \gamma$, $a \models_i f$. For set-theoretic operators we take the standard meaning. The meaning of formulas of the form $f_1 + f_2$ is all configurations γ that can be decomposed into γ_1 and γ_2 ($\gamma = \gamma_1 \cup \gamma_2$) satisfying respectively f_1 and f_2. The formula $f_1 + f_2$ represents configurations obtained as the union of configurations of f_1 with configurations of f_2.

Despite its apparent complexity, configuration logic is easy to use because of its stratified construction. From interaction logic it inherits the Boolean connectives of conjunction ($\wedge$), disjunction ($\vee$) and negation ($\bar{}$). It also uses the set-theoretic operations of union ($\sqcup$), complementation ($\neg$) and coalescing ($+$). It can be shown that intersection coincides with conjunction.

Formulas of the form $f + true$, denoted $\sim f$, present a particular interest for writing specifications. Their characteristic configuration set is the largest set containing configurations satisfying f.

We provide a full axiomatisation of the propositional configuration logic and a normal form similar to the disjunctive normal form in Boolean algebras. The existence of such normal form implies the decidability of formula equality and of satisfaction of a formula by an architecture model.

To allow genericity of specifications, we study first-order and second-order extensions of the propositional configuration logic. First-order logic formulas involve quantification over component variables. Second-order logic formulas involve additionally quantification over sets of components. Second-order logic is needed to express interesting topological properties, e.g. the existence of interaction cycles.

A complete presentation, with proofs and additional examples, of the results in this paper can be found in the technical report [22].
The paper is structured as follows. Section 2 recalls some basic facts about the interaction logic. Section 3 presents the propositional configuration logic, its properties and the definition of a normal form. Section 4 proposes a methodology for the specification of architecture styles. Section 5 presents first-order and second-order extensions of the logic and illustrates their use by several architecture style examples. Section 6 presents the results of an implementation of the decision procedure in the Maude rewriting system. Section 7 discusses related work. Section 8 concludes the paper.

2 Propositional interaction logic

The propositional interaction logic (PIL), studied in [2, 3], is a Boolean logic used to characterize the interactions between components on a global set of ports P. In this section, we present only the results needed to introduce the propositional configuration logic (Sect. 3). Below, we assume that the set P is given.

Definition 1. *An* interaction *is a set of ports $a \subseteq P$ such that $a \neq \emptyset$.*

Syntax. The propositional interaction logic is defined by the grammar:

$$\phi ::= \mathit{true} \mid p \mid \overline{\phi} \mid \phi \vee \phi\,, \qquad \text{with any } p \in P.$$

Conjunction is defined as usual: $\phi_1 \wedge \phi_2 \stackrel{def}{=} \overline{(\overline{\phi_1} \vee \overline{\phi_2})}$. To simplify the notation, we omit it in monomials, e.g. writing pqr instead of $p \wedge q \wedge r$.

Semantics. The meaning of a PIL formula ϕ is defined by the following satisfaction relation. Let $a \subseteq P$ be a non-empty interaction. We define: $a \models_i \phi$ iff ϕ evaluates to *true* for the valuation $p = \mathit{true}$, for all $p \in a$, and $p = \mathit{false}$, for all $p \notin a$. Thus, the semantic domain of PIL is the lattice of configurations $C(P) = 2^{I(P)\setminus\{\emptyset\}}$, where $I(P) = 2^P$ (Fig. 2).

The operators meet the usual Boolean axioms and the additional axiom $\bigvee_{p\in P} p = \mathit{true}$ meaning that interactions are non-empty sets of ports.
An interaction a can be associated to a characteristic monomial $m_a = \bigwedge_{p\in a} p \wedge \bigwedge_{p\notin a} \overline{p}$ such that $a' \models_i m_a$ iff $a' = a$.

Example 1. Consider a system consisting of three components: a sender with port p and two receivers with ports q and r respectively. We can express the following interaction patterns:

- *Strong synchronization* between the components is specified by a single interaction involving all components. This is represented by the single monomial pqr.
- *Broadcast* defines weak synchronization among the sender and any number of the receivers: $\big\{\{p\}, \{p, q\}, \{p, r\}, \{p, q, r\}\big\}$, represented by the formula p, which can be expanded to $p\overline{q}\,\overline{r} \vee pq\overline{r} \vee p\overline{q}\,r \vee pqr$.

3 Propositional configuration logic

3.1 Syntax and semantics

Syntax. The propositional configuration logic (PCL) is a powerset extension of PIL defined by the following grammar:

$$f ::= \mathit{true} \mid \phi \mid \neg f \mid f + f \mid f \sqcup f\,,$$

where ϕ is a PIL formula; $\neg$, $+$ and $\sqcup$ are respectively the *complementation*, *coalescing* and *union* operators.
We define the usual notation for intersection and implication: $f_1 \sqcap f_2 \stackrel{def}{=} \neg(\neg f_1 \sqcup \neg f_2)$ and $f_1 \Rightarrow f_2 \stackrel{def}{=} \neg f_1 \sqcup f_2$.

The language of PCL formulas is generated from PIL formulas by using union, coalescing and complementation operators. The binding strength of the operators is as follows (in decreasing order): PIL negation, complementation, PIL conjunction, PIL disjunction, coalescing, union. Henceforth, to avoid confusion, we refer as *interaction formulas* to the subset of PCL formulas that syntactically are also PIL formulas. Furthermore, we will use Latin letters f, g, h for general PCL formulas and Greek letters ϕ, ψ, ξ for interaction formulas. Interaction formulas inherit all axioms of PIL.

Semantics. Let P be a set of ports. The semantic domain of PCL is the lattice of configuration sets $CS(P) = 2^{C(P)\setminus\{\emptyset\}}$ (Fig. 2(c)). The meaning of a PCL formula f is defined by the following satisfaction relation. Let $\gamma \in C(P)$ be a non-empty configuration. We define:

$\gamma \models \mathit{true}\,,$	always,
$\gamma \models \phi\,,$	if $\forall a \in \gamma, a \models_i \phi$, where ϕ is an interaction formula and $\models_i$ is the satisfaction relation of PIL,
$\gamma \models f_1 + f_2\,,$	if there exist $\gamma_1, \gamma_2 \in C(P) \setminus \{\emptyset\}$, such that $\gamma = \gamma_1 \cup \gamma_2$, $\gamma_1 \models f_1$ and $\gamma_2 \models f_2$,
$\gamma \models f_1 \sqcup f_2\,,$	if $\gamma \models f_1$ or $\gamma \models f_2$,
$\gamma \models \neg f\,,$	if $\gamma \not\models f$ (i.e. $\gamma \models f$ does not hold).

In particular, the meaning of an interaction formula ϕ in PCL is the set $2^{I_a} \setminus \{\emptyset\}$, with $I_a = \{a \in I(P) \mid a \models_i \phi\}$, of all configurations involving any number of interactions satisfying ϕ in PIL.
We say that two formulas are equivalent $f_1 \equiv f_2$ iff, for all $\gamma \in C(P)$ such that $\gamma \neq \emptyset$, $\gamma \models f_1 \Leftrightarrow \gamma \models f_2$.

Proposition 1. *Equivalence $\equiv$ is a congruence w.r.t. all PCL operators.*

Example 2. The Master/Slave architecture style for two masters M_1, M_2 and two slaves S_1, S_2 with ports m_1, m_2, s_1 and s_2 respectively characterizes the four configurations of Fig. 1 as the union:

$$\bigsqcup_{i,j\in\{1,2\}} (\phi_{1,i} + \phi_{2,j}),$$

where, for $i \neq i'$ and $j \neq j'$, the monomial $\phi_{i,j} = s_i\, m_j\, \overline{s_{i'}}\ \overline{m_{j'}}$ defines a binary interaction between ports s_i and m_j.

3.2 Conservative extension of PIL operators

Notice that from the PCL semantics of interaction formulas, it follows immediately that PCL is a conservative extension of PIL. Below we extend the PIL conjunction and disjunction operators to PCL.
PCL intersection is a conservative extension of PIL conjunction.

Proposition 2. *$\phi_1 \wedge \phi_2 \equiv \phi_1 \sqcap \phi_2$, for any interaction formulas ϕ_1, ϕ_2.*

Thus, conjunction and intersection coincide on interaction formulas. In the rest of the paper, we use the same symbol $\wedge$ to denote both operators. Disjunction can be conservatively extended to PCL with the following semantics: for any PCL formulas f_1 and f_2,

$$\gamma \models f_1 \vee f_2\,, \quad \text{if } \gamma \models f_1 \sqcup f_2 \sqcup f_1 + f_2. \tag{1}$$

Proposition 3. *For any interaction formulas ϕ_1 and ϕ_2 and any $\gamma \in C(P)$ such that $\gamma \neq \emptyset$, we have $\gamma \models \phi_1 \vee \phi_2$ iff $\forall a \in \gamma, a \models_i \phi_1 \vee \phi_2$.*

3.3 Properties of PCL operators

Union, complementation and conjunction operators have the standard set-theoretic meaning and consequently, they satisfy the usual axioms of propositional logic.
The coalescing operator $+$ combines configurations, as opposed to the union operator $\sqcup$, which combines configuration sets. Coalescing has the following properties:

Proposition 4. *$+$ is associative, commutative and has an absorbing element false $\stackrel{def}{=} \neg true$.*

Proposition 5. *For any formulas f, f_1, f_2 and any interaction formula ϕ, we have the following distributivity results:*
1. *$f \vee (f_1 \sqcup f_2) \equiv (f \vee f_1) \sqcup (f \vee f_2)$,*
2. *$f + (f_1 \vee f_2) \equiv (f + f_1) \vee (f + f_2)$,*
3. *$f + (f_1 \sqcup f_2) \equiv f + f_1 \sqcup f + f_2$,*
4. *$\phi \wedge (f_1 + f_2) \equiv (\phi \wedge f_1) + (\phi \wedge f_2)$.*

Associativity of coalescing and union, together with the distributivity of coalescing over union, immediately imply the following generalisation of the extended semantics of disjunction (1).

Corollary 1. *For any set of formulas $\{f_i\}_{i \in I}$, we have*

$$\bigvee_{i \in I} f_i \equiv \bigsqcup_{\emptyset \neq J \subseteq I} \sum_{j \in J} f_j\,,$$

where $\sum_{j \in J} f_j$ denotes the coalescing of formulas f_j, for all $j \in J$.

Example 3. A configuration γ satisfying the formula $f = f_1 \vee f_2 \vee f_3$ can be partitioned into $\gamma = \gamma_1 \cup \gamma_2 \cup \gamma_3$, such that $\gamma_i \models f_i$. However, by the semantics of disjunction, some γ_i can be empty. On the contrary, the semantics of coalescing requires all elements of such partition to be non-empty. Hence, in order to rewrite f without the disjunction operator, we take the union of all possible coalescings of f_1, f_2 and f_3. Thus, we have $f \equiv f_1 \sqcup f_2 \sqcup f_3 \sqcup (f_1 + f_2) \sqcup (f_1 + f_3) \sqcup (f_2 + f_3) \sqcup (f_1 + f_2 + f_3)$.

Notice that in general coalescing does not distribute over conjunction.

Example 4. Let $P = \{p, q\}$ and consider $f = p \sqcup q$, $f_1 = p$ and $f_2 = q$. Configuration $\big\{\{p\}, \{q\}\big\}$ satisfies $(f + f_1) \wedge (f + f_2)$, but not $f + (f_1 \wedge f_2)$.

Coalescing with *true* presents a particular interest for writing specifications, since they allow adding any set of interactions to the configurations satisfying f. Notice that *true* is not a neutral element of coalescing: only the implication $f \Rightarrow f + \mathit{true}$ holds in general.

Definition 2. *For any formula f, the* closure operator $\sim$ *is defined by putting* $\sim f \stackrel{def}{=} f + \mathit{true}$. *We give* $\sim$ *the same binding power as* $\neg$.

Example 5. For $P = \{p, q, r\}$ the formula f characterizing all the configurations such that p must interact with both q and r, is $f = \sim(pq + qr) = pq + pr + \mathit{true}$. Notice that the only constraint imposed by the formula f is that configurations that satisfy it must contain an interaction pqr or both interactions pq and qr. Configurations satisfying f can contain any additional interactions.

Proposition 6. *For any formula f, we have* $\sim\sim f \equiv \sim f$.

The closure operator can be interpreted as a modal operator with existential quantification. The formula $\sim f$ characterizes configurations γ, such that there *exists* a sub-configuration of γ satisfying f. Thus, $\sim f$ means "possible f". Dually $\neg \sim \neg f$ means "always f" in the following sense: if a configuration γ satisfies $\neg \sim \neg f$, *all* sub-configurations of γ satisfy f. Below, we show that, for an interaction formula ϕ, holds the equivalence $\sim \neg\phi \equiv \neg\phi$, which implies $\neg \sim \neg\phi \equiv \neg\neg\phi \equiv \phi$. However, this is not true in general. Consider $f = m_a + m_b$, where m_a and m_b are characteristic monomials of interactions a and b respectively. The only configuration satisfying f is $\gamma = \{a, b\}$. In particular, none of the sub-configurations $\{a\}, \{b\} \subset \gamma$ satisfies f. Thus, $\neg \sim \neg(m_a + m_b) \equiv \mathit{false}$.

Proposition 7. *For any f_1 and f_2, we have*
1. $\sim(f_1 \sqcup f_2) \equiv \sim f_1 \sqcup \sim f_2 \equiv \sim(f_1 \vee f_2)$,
2. $\sim(f_1 + f_2) \equiv \sim f_1 + \sim f_2 \equiv \sim f_1 \wedge \sim f_2$.

The following proposition allows us to address the relation between complementation and negation.

Proposition 8. *For any interaction formula ϕ, we have*

$$\phi \sqcup \overline{\phi} \sqcup (\overline{\phi} + \phi) \equiv \mathit{true}\,.$$

Notice that the three terms on the left are mutually disjoint and therefore, for any interaction formula ϕ, we have

$$\neg\phi \equiv \overline{\phi} \sqcup (\phi + \overline{\phi}) \equiv \overline{\phi} + \mathit{true} \equiv \sim\overline{\phi}\,. \tag{2}$$

In particular, this means that complementation can also be interpreted as a modality. Prop. 8 shows that the complementation of an interaction formula ϕ represents all configurations that contain $\overline{\phi}$. Equivalences $\neg\overline{\phi} \equiv \sim\phi$, $\neg \sim \phi \equiv \overline{\phi}$, $\neg \sim \overline{\phi} \equiv \phi$ and $\sim \neg\phi \equiv \neg\phi$, for interaction formulas ϕ, are direct corollaries of Prop. 8 and, for the latter, Prop. 6. The following proposition generalises (2) to coalescings of interaction formulas.

Proposition 9. *For any set of interaction formulas Φ, we have*

$$\neg\Big(\sum_{\phi\in\Phi}\phi\Big) \equiv \bigsqcup_{\phi\in\Phi}\overline{\phi}\sqcup\sim\Big(\bigwedge_{\phi\in\Phi}\overline{\phi}\Big).$$

Example 6. Consider a formula $f=\neg(pq+pr)$ and a configuration $\gamma\models f$. The PCL semantics requires that γ cannot be split into two non-empty parts $\gamma_1\models pq$ and $\gamma_2\models pr$. This can happen in two cases: 1) there exists $a\in\gamma$ such that a does not satisfy neither pq nor pr; 2) one of the monomials is not satisfied by any interaction in γ. The former case can be expressed as $\sim(\overline{pq}\ \overline{pr}\,)$ and the latter as $\overline{pq}\sqcup\overline{pr}$. The union of these formulas gives the equivalence $\neg(pq+pr)\equiv\overline{pq}\sqcup\overline{pr}\sqcup\sim(\overline{pq}\ \overline{pr}\,)$.

Prop. 9 allows the elimination of complementation. It is also possible to eliminate conjunction of coalescings by using the following distributivity results to push it down within the formula.

Proposition 10. *For two sets of interaction formulas Φ and Ψ, we have*

$$\sum_{\phi\in\Phi}\phi\wedge\sum_{\psi\in\Psi}\psi\ \equiv\sum_{\xi\in\Phi\cup\Psi}\Big(\xi\wedge\bigvee_{(\phi,\psi)\in\Phi\times\Psi}(\phi\wedge\psi)\Big).$$

Example 7. Consider a formula $f=(\phi_1+\phi_2)\wedge(\phi_3+\phi_4)$, where ϕ_1, ϕ_2, ϕ_3 and ϕ_4 are interaction formulas, and a configuration $\gamma\models f$. The semantics requires that there exists two partitions of γ: $\gamma=\gamma_1\cup\gamma_2$ and $\gamma=\gamma_3\cup\gamma_4$, such that $\gamma_i\models\phi_i$ for $i\in[1,4]$. Considering an intersection $\gamma_{i,j}=\gamma_i\cap\gamma_j$ we have $\gamma_{i,j}\models\phi_i\wedge\phi_j$. Thus, $\gamma=\bigcup\gamma_{i,j}$ satisfies $\phi_1\phi_3\vee\phi_1\phi_4\vee\phi_2\phi_3\vee\phi_2\phi_4$ even if some $\gamma_{i,j}$ are empty. However, disjunction allows configurations such that no interaction satisfy one of the disjunction terms and consequently some ϕ_i. A coalescing of ϕ_i allows only configurations such that each ϕ_i is satisfied by at least one interaction. Thus, the conjunction of these formulas gives the equivalent representation:

$$\begin{aligned}f &\equiv(\phi_1\phi_3\vee\phi_1\phi_4\vee\phi_2\phi_3\vee\phi_2\phi_4)\wedge(\phi_1+\phi_2+\phi_3+\phi_4)\\ &\equiv\phi_1\wedge(\phi_1\phi_3\vee\phi_1\phi_4\vee\phi_2\phi_3\vee\phi_2\phi_4)\\ &\quad+\phi_2\wedge(\phi_1\phi_3\vee\phi_1\phi_4\vee\phi_2\phi_3\vee\phi_2\phi_4)\\ &\quad+\phi_3\wedge(\phi_1\phi_3\vee\phi_1\phi_4\vee\phi_2\phi_3\vee\phi_2\phi_4)\\ &\quad+\phi_4\wedge(\phi_1\phi_3\vee\phi_1\phi_4\vee\phi_2\phi_3\vee\phi_2\phi_4)\,.\end{aligned}$$

The PCL lattice is illustrated in Fig. 3. The circle nodes represent interaction formulas, whereas the red dot nodes represent all other formulas. Notice that the PCL lattice has two sub-lattices generated by monomials:

- through disjunction and negation (isomorphic to the PIL lattice);
- through union and complementation (disjunction is not expressible).

Notice that coalescing cannot be expressed in any of these two sub-lattices. Although some formulas involving the closure operator can be expressed in the second sub-lattice, e.g. $\sim\overline{\phi}\equiv\neg\phi$, in general this is not the case, e.g. the formulas $\sim(\overline{\phi}\wedge\overline{\psi}\,)$ and $\sim\phi\sqcup\sim\psi$ are not part of either sub-lattice. However, the closure operator is expressible by taking as generators the interaction formulas:

Proposition 11. *The lattice generated by interaction formulas through union and complementation is closed under the closure operator $\sim$.*

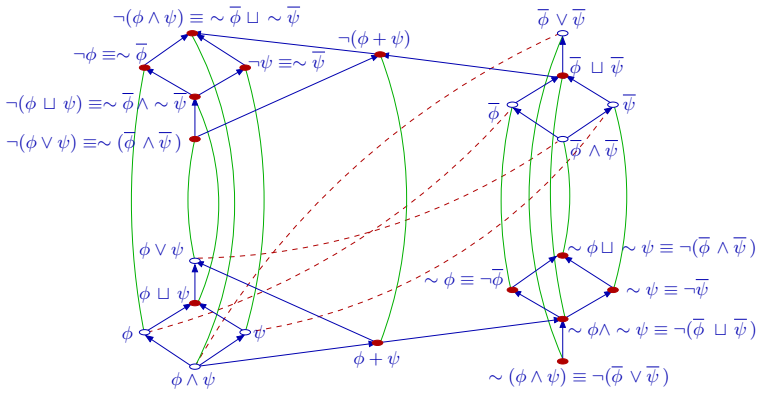


Fig. 3: PCL lattice (the blue arrows represent implications; red dashed and green solid lines represent, respectively, PIL negation and complementation).

3.4 Deciding equivalence and satisfaction

In this subsection, we present an axiomatisation of the PCL equivalence $\equiv$, which is sound and complete with respect to the definition in Sect. 3.1. This axiomatisation allows us to define a normal form for PCL formulas, similar to the disjunctive normal form in Boolean algebras. The existence of such a normal form immediately implies the decidability of 1) the equivalence of two PCL formulas and 2) the satisfaction of a formula by a configuration.

Axioms. PCL operators satisfy the following axioms (for any formulas f, f_1 and f_2 and any sets of interaction formulas Φ and Ψ):

1. The PIL axioms for interaction formulas.
2. The usual axioms of propositional logic for $\sqcup$, $\wedge$, $\neg$.
3. $+$ is associative, commutative and has an absorbing element *false*.
4. $f + (f_1 \sqcup f_2) \equiv f + f_1 \sqcup f + f_2$.
5. $\sum_{\phi\in\Phi} \phi \wedge \sum_{\psi\in\Psi} \psi \equiv \sum_{\xi\in\Phi\cup\Psi} \Big(\xi \wedge \bigvee_{(\phi,\psi)\in\Phi\times\Psi} (\phi \wedge \psi)\Big)$.
6. $\neg\Big(\sum_{\phi\in\Phi} \phi\Big) \equiv \bigsqcup_{\phi\in\Phi} \overline{\phi} \sqcup \sim\Big(\bigwedge_{\phi\in\Phi} \overline{\phi}\Big)$.

Theorem 1. *The above set of axioms is sound and complete for the equivalence $\equiv$ in PCL.*

Applying the axioms above, one can remove or push PCL operators down in the expression tree of the formula. For instance, Ax. 5 allows one to push the conjunction down, Ax. 6 removes the complementation.[1]

Definition 3. *A PCL formula is in* normal form *iff it has the form $\bigsqcup_{i\in I} \sum_{j\in J_i} \bigvee_{k\in K_{i,j}} m_{i,j,k}$, where all $m_{i,j,k}$ are monomials.*

Theorem 2. *Any PCL formula has an equivalent normal form formula.*

[1] Full details of the normal form derivation can be found in the technical report [22].

Example 8. The following example illustrates the normalization process:

$$
\begin{aligned}
&(pq \sqcup r) \wedge (pr + \neg q) \equiv (pq \sqcup r) \wedge (pr + \overline{q} + true) && \text{// Ax. 6}\\
&\quad \equiv (pq \wedge (pr + \overline{q} + true)) \sqcup (r \wedge (pr + \overline{q} + true)) && \text{// Ax. 2}\\
&\quad \equiv ((pq \wedge pr) + (pq \wedge \overline{q}) + (pq \wedge true)) && \text{// Ax. 5}\\
&\qquad \sqcup ((r \wedge pr) + (r \wedge \overline{q}) + (r \wedge true))\\
&\quad \equiv (pqr + false + pq) \sqcup (pr + r\overline{q} + r) && \text{// Ax. 1}\\
&\quad \equiv pr + r\overline{q} + r\,. && \text{// Ax. 2, 3}
\end{aligned}
$$

The first step removes the complementation. Then the application of distributivity rules pushes conjunction down in the expression tree of the formula, to the level of monomials. Finally, the formula is simplified, by observing that *false* is the absorbing element of coalescing and the identity of union.

4 Architecture style specification methodology

The methodology for writing architecture style specifications can be conceptually simplified due to the fact that an architecture can be considered as a hypergraph whose vertices are ports and edges are interactions. If a is an interaction then, its characteristic monomial m_a specifies in PCL a single configuration (hypergraph) that contains only the interaction (edge) a. The formula $\sim m_a$ specifies all the configurations (hypergraphs) that contain the interaction (edge) a. It can be considered as a predicate on ports expressing their connectivity.
A key idea in writing architecture style specifications is that these can be expressed as logical relations between connectivity formulas of the form $\sim\phi$ where ϕ is an interaction formula. This allows simplification through separation of concerns: first configurations are specified as the conjunction of formulas on Boolean variables representing connectivity formulas; then, after simplification, the connectivity formulas are replaced. This may require another round of simplifications based on specific properties of PCL. This idea is illustrated in the following example.

Example 9. Consider a system with three ports p, q, r and the following connectivity constraint: *If any port is connected to the two others, the latter have to be connected between themselves.* In order to specify this constraint in PCL, we first define three predicates $X = \sim(pq)$, $Y = \sim(qr)$ and $Z = \sim(pr)$. The constraint we wish to impose is then specified by the conjunction of the three implications: $(X \wedge Y \Rightarrow Z) \wedge (Y \wedge Z \Rightarrow X) \wedge (Z \wedge X \Rightarrow Y) \equiv \neg Z \wedge \neg Y \sqcup \neg Y \wedge \neg X \sqcup \neg X \wedge \neg Z \sqcup X \wedge Y \wedge Z$. Substituting $\sim(pq)$, $\sim(qr)$, $\sim(pr)$ for X, Y, Z, respectively, we obtain

$$
\begin{aligned}
&(\overline{p} \vee \overline{r}) \wedge (\overline{q} \vee \overline{r}) \sqcup (\overline{q} \vee \overline{r}) \wedge (\overline{p} \vee \overline{q}) \sqcup (\overline{p} \vee \overline{q}) \wedge (\overline{p} \vee \overline{r})\\
&\qquad \sqcup \sim(pq) \wedge \sim(qr) \wedge \sim(pr)\\
&\equiv \neg(\overline{r} \vee \overline{p}\ \overline{q}) \wedge \neg(\overline{q} \vee \overline{p}\ \overline{r}) \wedge \neg(\overline{p} \vee \overline{q}\ \overline{r}) \Rightarrow \sim(pq) \wedge \sim(qr) \wedge \sim(pr)\\
&\equiv \sim(pr \vee qr) \wedge \sim(qr \vee pq) \wedge \sim(pq \vee pr) \Rightarrow \sim(pq) \wedge \sim(qr) \wedge \sim(pr)\\
&\equiv \sim(pr) \wedge \sim(qr) \sqcup \sim(qr) \wedge \sim(pq) \sqcup \sim(pq) \wedge \sim(pr)\\
&\qquad \Rightarrow \sim(pq) \wedge \sim(qr) \wedge \sim(pr)\\
&\equiv \sim(pr + qr) \sqcup \sim(pq + qr) \sqcup \sim(pq + pr) \Rightarrow \sim(pq + qr + pr)\,.
\end{aligned}
$$

5 First and second order extensions of PCL

PCL is defined for a given set of ports and a given set of components. In order to specify architecture styles, we need quantification over component variables. We make the following assumptions:

- A finite set of component types $\mathcal{T} = \{T_1, \ldots, T_n\}$ is given. Instances of a component type have the same interface and behaviour. We write $c{:}T$ to denote a component c of type T.
- The interface of each component type has a distinct set of ports. We write $c.p$ to denote the port p of component c and $c.P$ to denote the set of ports of component c.

5.1 First-order configuration logic

Syntax. The language of the formulas of the first-order configuration logic extends the language of PCL by allowing Boolean expressions on component variables, universal quantification and a specific coalescing quantifier $\Sigma c{:}T$. Let ϕ denote any interaction formula:

$$F ::= \mathit{true} \mid \phi \mid \forall c{:}T\big(\Phi(c)\big).F \mid \Sigma c{:}T\big(\Phi(c)\big).F \mid F \sqcup F \mid \neg F \mid F + F\,,$$

where $\Phi(c)$ is some set-theoretic predicate on c (omitted when $\Phi = \mathit{true}$).

Semantics. The semantics is defined for closed formulas, where, for each variable in the formula, there is a quantifier over this variable in a higher nesting level. As above, we assume that the finite set of component types $\mathcal{T} = \{T_1, \ldots, T_n\}$ is given. Models are pairs $\langle B, \gamma\rangle$, where B is a set of component instances of types from $\mathcal{T}$ and γ is a configuration on the set of ports P of these components. For quantifier-free formulas, the semantics is the same as for PCL formulas. For formulas with quantifiers, the satisfaction relation is defined by the following rules:

$$\langle B, \gamma\rangle \models \forall c{:}T\big(\Phi(c)\big).F\,, \qquad \text{iff } \gamma \models \bigwedge_{c'{:}T \in B \,\wedge\, \Phi(c')} F[c'/c],$$

$$\langle B, \gamma\rangle \models \Sigma c{:}T\big(\Phi(c)\big).F\,, \qquad \text{iff } \gamma \models \sum_{c'{:}T \in B \,\wedge\, \Phi(c')} F[c'/c],$$

where $c' : T$ ranges over all component instances of type $T \in \mathcal{T}$ satisfying Φ and $F[c'/c]$ is obtained by replacing all occurrences of c in F by c'.

For a more concise representation of formulas, we introduce the notation $\sharp(c_1.p_1, \ldots, c_n.p_n)$, which expresses an exact interaction, i.e. all ports in the arguments and only they participate in the interaction:

$$\sharp(c_1.p_1, \ldots, c_n.p_n) \stackrel{def}{=} \bigwedge_{i=1}^{n} c_i.p_i \;\wedge \bigwedge_{i=1}^{n} \bigwedge_{p \in c_i.P \setminus \{p_i\}} \overline{c_i.p} \;\wedge \bigwedge_{T \in \mathcal{T}} \bigwedge_{c{:}T \notin \{c_1, \ldots, c_n\}} \bigwedge_{p \in c.P} \overline{c.p}\,. \quad (3)$$

Example 10. The Star architecture style is defined for a set of components of the same type. One central component s is connected to every other component through a binary interaction, and there are no other interactions. It can be specified as follows:

$$\exists s{:}T.\ \forall c{:}T(c \neq s).\ \big(\sim(c.p\ s.p)\ \wedge\ \forall c'{:}T(c' \notin \{c,s\}).\ (\overline{c'.p\ c.p})\big) \\ \wedge\ \big(\forall c{:}T.\ \neg \sim \sharp(c.p)\big)\,. \quad (4)$$

The three conjuncts of this formula express respectively the properties: 1) any component is connected to the center; 2) components other than the center are not connected; and 3) unary interactions are forbidden. Notice that the semantics of the first part of the specification, $\forall c{:}T(c \neq s).\ \sim(c.p\ s.p)$, is a conjunction of closure formulas. In this conjunction, the closure operator also allows interactions in addition to the ones explicitly defined. Therefore, to correctly specify this style, we need to forbid all other interactions with the second and third conjuncts of the specification. A simpler alternative specification uses the Σ quantifier:

$$\exists s{:}T.\ \Sigma c{:}T(c \neq s).\ \sharp(c.p,\ s.p)\,. \quad (5)$$

The $\sharp$ notation requires interactions to be binary and the Σ quantifier allows configurations that contain only interactions satisfying $\sharp(c.p,\ s.p)$, for some c. Thus, contrary to (4), we do not need to explicitly forbid unary interactions and connections between non-center components.

Example 11. The Pipes and Filters architecture style [13] involves two types of components, P and F, each having two ports *in* and *out*. Each input (resp. output) of a filter is connected to an output (resp. input) of a single pipe. The output of any pipe can be connected to at most one filter. This style can be specified as follows:

$$\forall f{:}F.\ \exists p{:}P.\ \sim(f.in\ p.out) \wedge \forall p'{:}P(p \neq p').\ \big(\overline{f.in\ p'.out}\big) \quad (6)$$

$$\wedge\ \forall f{:}F.\ \exists p{:}P.\ \sim(f.out\ p.in) \wedge \forall p'{:}P(p \neq p').\ \big(\overline{f.out\ p'.in}\big) \quad (7)$$

$$\wedge\ \forall p{:}P.\ \exists f{:}F.\ \forall f'{:}F(f \neq f').\ \big(\overline{p.out\ f'.in}\big) \quad (8)$$

$$\wedge\ \forall p{:}P.\ \Big(\overline{p.in\,p.out} \wedge \forall p'{:}P(p \neq p').\ \big(\overline{p.in\,p'.in} \wedge \overline{p.in\,p'.out}\big)\Big) \quad (9)$$

$$\wedge\ \forall f{:}F.\ \Big(\overline{f.in\,f.out} \wedge \forall f'{:}F(f \neq f').\ \big(\overline{f.in\,f'.in} \wedge \overline{f.in\,f'.out}\big)\Big)\,. \quad (10)$$

The first conjunct (6) requires that the input of each filter be connected to the output of a single pipe. The second conjunct (7) requires that the output of each filter be connected to the input of a single pipe. The third conjunct (8) requires that the output of a pipe be connected to at most one filter. Finally, the fourth and fifth conjuncts (9) and (10) require that pipes only be connected to filters and vice-versa.

Notice that (6) and (7) in Ex. 11 can be simplified by introducing the additional notation for "exists unique":

$$\exists!c{:}T\big(\Phi(c)\big).F(c) \ \stackrel{def}{=}\ \exists c{:}T\big(\Phi(c)\big).F(c) \wedge \forall c'{:}T\big(c \neq c' \wedge \Phi(c')\big).\neg F(c')\,. \quad (11)$$

Using this notation, (6) and (7) can be rewritten respectively as

$$\forall f\!:\!F.\ \exists!p\!:\!P.\ \sim(f.in\ p.out) \qquad \text{and} \qquad \forall f\!:\!F.\ \exists!p\!:\!P.\ \sim(f.out\ p.in)\,.$$

5.2 Second-order configuration logic

Properties stating that two components are connected through a chain of interactions, are essential for architecture style specification. For instance, the property that all components form a single ring and not several disjoint ones can be reformulated as such a property. In [18], it is shown that transitive closure, necessary to specify such reachability properties, cannot be expressed in the first-order logic. This motivates the introduction of the second-order configuration logic with quantification over sets of components.
This logic further extends PCL with variables ranging over component sets. We write $C\!:\!T$ to express the fact that all components belonging to C are of type T. Additionally, we denote C_T the set of all the components of type T. Finally, we assume the existence of the universal component type U, such that any component or component set is of this type. Thus, C_U represents all the components of a model.

Syntax. The syntax of the second-order configuration logic is defined by the following grammar (ϕ is an interaction formula):

$$\begin{aligned} S ::= {} & \mathit{true} \mid \phi \mid \forall c\!:\!T\big(\Phi(c)\big).S \mid \Sigma c\!:\!T\big(\Phi(c)\big).S \mid S \sqcup S \mid \neg S \mid S + S \\ & \mid \forall C : T\big(\Psi(C)\big).S \mid \Sigma C : T\big(\Psi(C)\big).S\,, \end{aligned}$$

where $\Phi(c)$, $\Psi(C)$ are some set-theoretic predicates (omitted when *true*).

Semantics. Models are pairs $\langle B, \gamma\rangle$, where B is a set of component instances of types from $\mathcal{T}$ and γ is a configuration on the set of ports P of these components. The meaning of quantifier-free formulas or formulas with quantification only over component variables is as for first-order logic. We define the meaning of quantifiers over component set variables:

$$\begin{aligned} \langle B, \gamma\rangle \models \forall C\!:\!T\big(\Psi(C)\big).S\,, \qquad & \text{iff } \gamma \models \bigwedge_{C'\!:T\subseteq B\wedge\Psi(C')} S[C'/C], \\ \langle B, \gamma\rangle \models \Sigma C\!:\!T\big(\Psi(C)\big).S\,, \qquad & \text{iff } \gamma \models \sum_{C'\!:T\subseteq B\wedge\Psi(C')} S[C'/C], \end{aligned}$$

where $C'\!:\!T$ ranges over all sets of components of type T that satisfy Ψ.

Example 12. The Repository architecture style [7] consists of a repository component r with a port p and a set of data-accessor components of type A with ports q. We provide below a list of increasingly strong properties that may be used to characterize this style:

1. The basic property *"there exists a single repository and all interactions involve it"* is specified as follows:
$$\exists r\!:\!R.\ (r.p) \wedge \forall r\!:\!R.\ \forall r'\!:\!R.\ (r = r'),$$
where the subterm $\forall r : R.\ \forall r' : R.\ (r = r')$ can be expressed in the logic as $\forall r\!:\!R.\ \forall r'\!:\!R(r' \neq r).\ \mathit{false}$.

2. The additional property *"there are some data-accessors and any data-accessor must be connected to the repository"* is enforced by extending the formula as follows:

$$\exists r\!:\!R.\ (r.p) \wedge \forall r\!:\!R.\ \forall r'\!:\!R.\ (r = r') \\ \wedge\ \exists a\!:\!A.\ true \wedge \forall a\!:\!A.\ \exists r\!:\!R.\ \sim(r.p\ a.q)$$

3. Finally, the additional property *"there are no components of other types than Repository and Data-accessor"* is enforced by the formula:

$$\exists r\!:\!R.\ (r.p) \wedge \exists a\!:\!A.\ true \wedge \forall a\!:\!A.\ \exists r\!:\!R.\ \sim(r.p\ a.q) \\ \wedge\ \forall r\!:\!R.\ \forall r'\!:\!R.\ (r = r') \wedge\ \forall c\!:\!U.\ (c \in C_R \sqcup c \in C_A)\,,$$

where the subterm $\forall c\!:\!U.\ (c \in C_R \sqcup c \in C_A)$ can be expressed as $\forall c\!:\!U(c \notin C_R \wedge c \notin C_A).\ false$.

Example 13. In the Ring architecture style (with only one component type T), all components form a single ring by connecting their *in* and *out* ports. This style can be specified as follows:

$$\Sigma c\!:\!T.\ \exists c'\!:\!T(c \neq c').\ \sharp(c.in,\ c'.out) \\ \wedge\ \Sigma c\!:\!T.\ \exists c'\!:\!T(c \neq c').\ \sharp(c.out,\ c'.in) \\ \wedge\ \forall C\!:\!T(C \neq U).\ (\exists c\!:\!T(c \in C).\ \exists c'\!:\!T(c' \notin C).\ \sim(c.in\ c'.out))\,.$$

The third conjunct ensures that there is a single ring and not several disjoint ones.

6 Implementation of the decision procedure

The PCL decision procedure is based on the computation of the normal form followed by a decision whether a model satisfies at least one union term of the normal form or not. For the first- and second-order extensions, satisfaction of a formula by a model can be decided by reduction to the decision procedure of PCL. Indeed, given a model, all quantifiers can be effectively eliminated, transforming a formula into a PCL one. Details of the procedure can be found in the technical report [22].

We implemented the decision procedure for PCL using Maude 2.0. Maude is a language and an efficient rewriting system supporting both equational and rewriting logic specification and programming for a wide range of applications. In the experimental evaluation we used a set of architecture styles including Star, Ring, Request-Response pattern [9], Pipes-Filters, Repository and Blackboard [8]. We used configuration logic formulas (all of them can be found in the technical report [22]) and models of different sizes, including both correct and incorrect models. Quantifiers were eliminated externally and the decision procedure was applied to quantifier-free formulas. All experiments have been performed on a 64-bit Linux machine with a 2.8 Ghz Intel i7-2640M CPU with a memory limit of 1Gb and time limit of 600 seconds.

Fig. 4 shows the average duration of the decision procedure for the six examples, as a function of the total number of ports involved in the formula. Simple architecture styles like Star are decidable within seconds

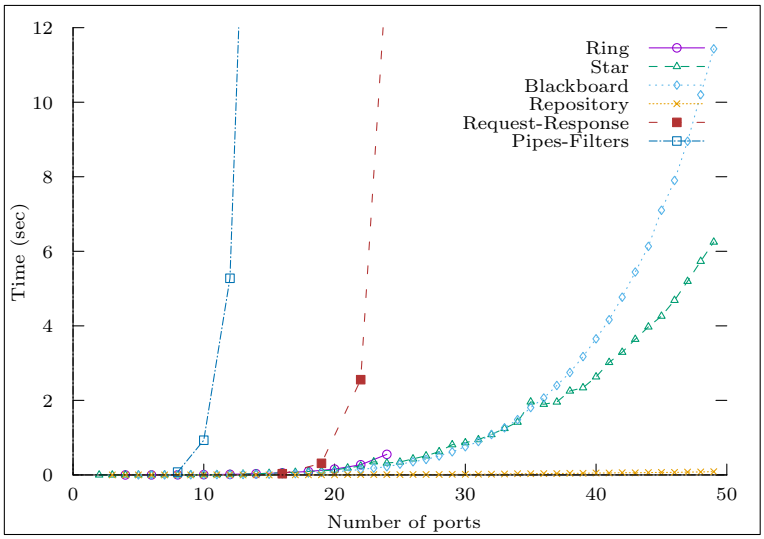


Fig. 4: Decision procedure for architecture styles

even for 50 ports. For architecture styles requiring more complex specifications, the number of ports that can be managed in 600 seconds is smaller. For the Ring architecture the memory limit is attained for the model with 24 ports.

7 Related work

A plethora of approaches exist for characterizing architecture styles. Patterns [9, 16] are commonly used for this purpose in practical applications. They incorporate explicit constructs for architecture modelling but, lacking formal semantics, are not amenable to formal analysis.

Among the formal approaches for representing and analysing architecture styles, we distinguish two main categories:

- *Extensional approaches*, where one explicitly specifies all interactions among the components (cf. the specification (5) of the Star pattern). All connections, other than the ones specified, are excluded.
- *Intentional approaches*, where one does not explicitly specify all the connections among the components, but these are derived from a set of logical constraints, formulating the intentions of the designer (cf. the specification (4) of the Star pattern). In this case specifications are conjunctions of logical formulas.

The proposed framework encompasses both approaches. It allows explicit specification of individual interactions, e.g. by using interaction formulas, as well as explicit specification of configuration sets, e.g. by using formulas of the form $\sim f$.

A large body of literature, originating in [15, 21], studies the use of graph grammars and transformations [24] to define software architectures. Although this work focuses mainly on dynamic reconfiguration of architectures, e.g. [6, 19, 20], graph grammars can be used to extensionally

define architecture styles: a style admits all the configurations that can be derived by its defining grammar. The main limitations, outlined already in [21], are the following: 1) the difficulty of understanding the architecture style defined by a grammar; 2) the fact that the restriction to context-free grammars precludes the specification of certain styles (e.g. trees with unbounded number of components or interactions, square grids); 3) the impossibility of combining several styles in a homogeneous manner. To some extent, the latter two are addressed, respectively, by considering synchronised hyperedge replacement [11], context-sensitive grammars [10, 27] and architecture views [23]. Our approach avoids these problems. Combining the extensional and intentional approaches allows intuitive specification of architecture styles. The higher-order extensions of PCL allow imposing global constraints necessary to specify styles that are not expressible by context-free graph grammars. Finally, the combination of several architecture styles is defined by the conjunction of the corresponding PCL formulas.

The proposed framework has similarities, but also significant differences, with the use of Alloy [17] and OCL [26] for intentional specification of architecture styles, respectively, in ACME and Darwin [12, 14] and in UML [5]. Our approach achieves a strong semantic integration between architectures and architecture styles. Moreover, configuration logic allows a fine characterization of the coordination structure by using n-ary connectivity predicates. On the contrary, the connectivity primitives in [12, 14, 26] are binary predicates and cannot tightly characterize coordination structures involving multiparty interaction. To specify an n-ary interaction, these approaches require an additional entity connected by n binary links with the interacting ports. Since the behaviour of such entities is not part of the architecture style, it is impossible to distinguish, e.g., between an n-ary synchronisation and a sequence of n binary ones.

Both Alloy and OCL rely on first-order logics extended with some form of the Kleene closure operator that allows to iterate over a transitive relationship. In particular, this operator allows defining reachability among components. It is known that the addition of the Kleene closure increases the expressive power w.r.t. a first-order logic [18]. To the best of our knowledge, the expressiveness relation between a first-order logic extended with Kleene closure and a corresponding second-order logic remains to be established.

8 Conclusion

The presented work is a contribution to a long-term research program that we have been pursuing for more than 15 years. The program aims at developing the BIP component framework for rigorous systems design [25]. BIP is a language and a set of supporting tools including code generators, verification and simulation tools. So far the theoretical work has focused on the study of expressive composition frameworks and their algebraic and logical formalization. This led in particular, to the formalization of architectures as a generic coordination schemes applied to sets of components in order to enforce a given global property [1].

The presented work nicely complements the existing component framework with logics for the specification of architecture styles. Configuration

logic formulas characterize interaction configurations between instances of typed components. Quantification over components and sets of components allows the genericity needed for architecture styles. We have shown through examples that configuration logic allows full expressiveness combined with ease of use.

Configuration logic is a powerset extension of interaction logic used to describe architectures. It is integrated in a unified semantic framework which is equipped with a decision procedure for checking that a given architecture model meets given style requirements.

As part of the future work, we will extend our results in several directions. From the specification perspective, we are planning to incorporate hierarchically structured interactions and data transfer among the participating ports. From the analysis perspective, we will study techniques for deciding satisfiability of higher-order extensions of PCL. Finally, from the practical perspective, we also plan to extend to the higher-order logics the Maude implementation of the decision procedures. We will also study sublogics that are practically relevant and for which more efficient decision procedures can be applied.

In parallel, we are currently using configuration logic to formally specify reference architectures for avionics systems, in a project with ESA.

References

1. Paul Attie, Eduard Baranov, Simon Bliudze, Mohamad Jaber, and Joseph Sifakis. A general framework for architecture composability. In *SEFM'14*, number 8702 in LNCS, pages 128–143. Springer, 2014.
2. Simon Bliudze and Joseph Sifakis. The algebra of connectors—structuring interaction in BIP. *IEEE Transactions on Computers*, 57(10):1315–1330, 2008.
3. Simon Bliudze and Joseph Sifakis. Causal semantics for the algebra of connectors. *FMSD*, 36(2):167–194, 2010.
4. Simon Bliudze and Joseph Sifakis. Synthesizing glue operators from glue constraints for the construction of component-based systems. In *SC '11*, volume 6708 of *LNCS*, pages 51–67. Springer, 2011.
5. Grady Booch, James Rumbaugh, and Ivar Jacobson. The unified modeling language user guide. *Addison-Welsley Longman Inc*, 1999.
6. Roberto Bruni, Alberto Lluch-Lafuente, Ugo Montanari, and Emilio Tuosto. Style-based architectural reconfigurations. *Bulletin of the EATCS*, 94:161–180, 2008.
7. Paul Clements, David Garlan, Len Bass, Judith Stafford, Robert Nord, James Ivers, and Reed Little. *Documenting software architectures: views and beyond*. Pearson Education, 2002.
8. Daniel D. Corkill. Blackboard systems. *AI expert*, 6(9):40–47, 1991.
9. Robert Daigneau. *Service Design Patterns: fundamental design solutions for SOAP/WSDL and restful Web Services*. Addison-Wesley, 2011.
10. Hartmut Ehrig and Barbara Konig. Deriving bisimulation congruences in the DPO approach to graph rewriting. In *FoSSaCS*, volume 2987 of *LNCS*, pages 151–166. Springer, 2004.
11. Gian Luigi Ferrari, Dan Hirsch, Ivan Lanese, Ugo Montanari, and Emilio Tuosto. Synchronised hyperedge replacement as a model for

service oriented computing. In *Formal Methods for Components and Objects*, pages 22–43. Springer, 2006.
12. David Garlan, Robert Monroe, and David Wile. Acme: An architecture description interchange language. In *CASCON '97*, pages 159–173. IBM Press, 1997.
13. David Garlan and Mary Shaw. An introduction to software architecture. In *Advances in Software Engineering and Knowledge Engineering*, pages 1–39. World Scientific Publishing Company, 1993.
14. Ioannis Georgiadis, Jeff Magee, and Jeff Kramer. Self-organising software architectures for distributed systems. In *Proceedings of the first workshop on Self-healing systems*, pages 33–38. ACM, 2002.
15. Dan Hirsch, Paola Inverardi, and Ugo Montanari. Modeling software architectures and styles with graph grammars and constraint solving. In Patrick Donohoe, editor, *Software Architecture*, volume 12 of *IFIP*, pages 127–143. Springer, 1999.
16. Gregor Hohpe and Bobby Woolf. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.
17. Daniel Jackson. Alloy: A lightweight object modelling notation. *ACM Trans. Softw. Eng. Methodol.*, 11(2):256–290, April 2002.
18. Uwe Keller. Some remarks on the definability of transitive closure in first-order logic and Datalog. Internal report, Digital Enterprise Research Institute (DERI), University of Innsbruck, 2004.
19. Christian Koehler, Alexander Lazovik, and Farhad Arbab. Connector rewriting with high-level replacement systems. *Electronic Notes in Theoretical Computer Science*, 194(4):77–92, 2008.
20. Christian Krause, Ziyan Maraikar, Alexander Lazovik, and Farhad Arbab. Modeling dynamic reconfigurations in Reo using high-level replacement systems. *Sci. of Comp. Prog.*, 76(1):23–36, 2011.
21. Daniel Le Métayer. Describing software architecture styles using graph grammars. *IEEE Transactions on Software Engineering*, 24(7):521–533, July 1998.
22. Anastasia Mavridou, Eduard Baranov, Simon Bliudze, and Joseph Sifakis. Configuration logics - modelling architecture styles. Technical Report EPFL-REPORT-206825, EPFL IC IIF RiSD, March 2015. Available at: `http://infoscience.epfl.ch/record/206825`.
23. Dewayne E. Perry and Alexander L. Wolf. Foundations for the study of software architecture. *ACM SIGSOFT Software Engineering Notes*, 17(4):40–52, 1992.
24. Grzegorz Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformation*. World Scientific, 1997.
25. Joseph Sifakis. Rigorous system design. *Foundations and Trends in Electronic Design Automation*, 6:293–362, 2012.
26. Jos B. Warmer and Anneke G. Kleppe. *The Object Constraint Language: Precise Modeling With UML*. Addison-Wesley, 1998.
27. Da-Qian Zhang, Kang Zhang, and Jiannong Cao. A context-sensitive graph grammar formalism for the specification of visual languages. *The Computer Journal*, 44(3):186–200, 2001.