Configuration Logic—Modelling Architecture Styles

Anastasia Mavridou Eduard Baranov Simon Bliudze Joseph Sifakis
École polytechnique fédérale de Lausanne, Station 14, 1015 Lausanne, Switzerland
firstname.lastname@epfl.ch

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 logic for the specification of architectures as generic coordination schemes involving a configuration of interactions between typed components; 2) configuration logic for the specification of architecture styles as sets of interaction configurations. Our results build on previous work on architecture modelling in BIP. We show how the propositional interaction logic can be extended to 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 procedure for deriving a normal form for any formula. To allow genericity of specifications, we study first-order and second-order extensions of the propositional configuration logic.

1 Introduction

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 such architectures for arbitrary numbers of masters and slaves.

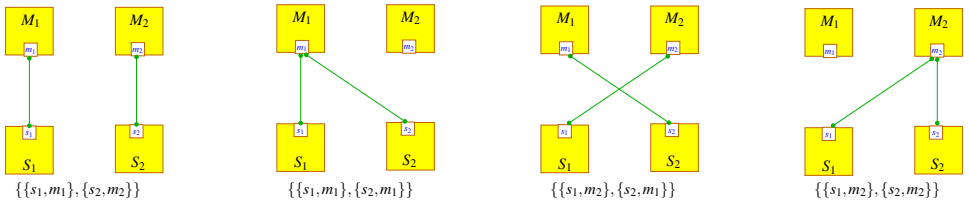
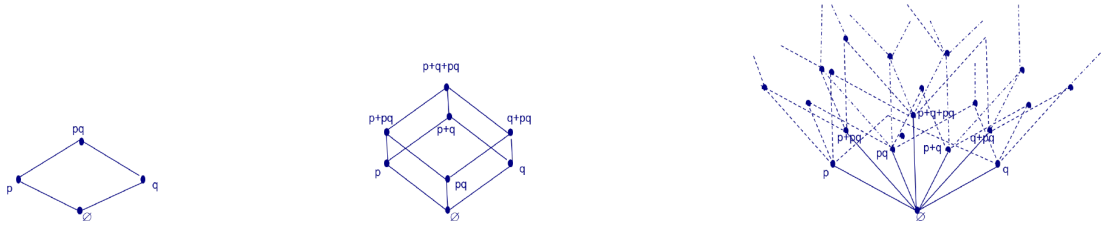


Figure 1: Master/Slave architectures

The relation between architectures and architecture styles 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.

We define the propositional configuration logic (PCL) whose formulas represent, for a given set of components, the allowed configuration sets. A configuration on a set of components represents a particular architecture. Defining configuration logics requires considering three hierarchically structured semantic domains: the lattices of interactions, configurations and configuration sets. An interaction is an

Submitted to:
ICE 2015



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

Figure 2: Lattices of interactions (a), configurations (b) and configuration sets (c) for $P = \{p,q\}$.

n-ary connector between ports. Configurations are sets of interactions characterizing architectures. Sets of configurations are properties described by the configuration logic. Figure 2 shows the three lattices for $P = \{p,q\}$. We only show the generators of the lattice of configuration sets.

This work consistently extends results on modelling architectures by using propositional interaction logics [4, 5, 6], 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 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 interaction logic. Its formulas are generated from the formulas of the propositional interaction logic by using the operators *union* $\oplus$, *complementation* $\neg$ and *coalescing* $+$. Its semantics is defined via a satisfaction relation $\models$ between configurations $\gamma = \{a_1,\ldots,a_n\}$ and formulas. For a PIL formula f, we define $\gamma \models f$ iff $a \models_i f$, for all $a \in \gamma$. For set-theoretic operators, we take the standard meaning. The formula $f_1 + f_2$ represents configurations γ obtained as the union of configurations of f_1 with configurations of f_2, i.e. $\gamma \models f_1 + f_2$ iff γ can be decomposed as $\gamma = \gamma_1 \cup \gamma_2$, such that $\gamma_1 \models f_1$ and $\gamma_2 \models f_2$. Formulas of the form $f + true$ present a particular interest for writing specifications. Their characteristic configuration set is the set of all configurations, each containing a subset of interactions satisfying f. Despite its apparent complexity, configuration logic is easy to use because of its stratified construction.

We provide a full axiomatization 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 illustrating the results discussed here can be found in the technical report [10].

2 Propositional configuration logic

The propositional configuration logic (PCL), is an extension of the propositional interaction logic (PIL), which was studied in [4, 5]. PIL is a Boolean logic used to characterize the interactions between components on a global set of ports P. Interactions are non-empty sets of ports $\emptyset \neq a \subseteq P$. PIL is defined by the following grammar:

$$\phi ::= true \mid p \in P \mid \overline{\phi} \mid \phi \vee \phi \mid \phi \wedge \phi\,.$$

To simplify the notation, we omit conjunction 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 $\emptyset \neq a \subseteq P$ be an interaction. We define: $a \models_i \phi$ iff ϕ evaluates to true for the valuation $p = true$, for all $p \in a$, and $p = false$, for all $p \notin a$.*

PCL is an extension of PIL defined by the grammar

$$f ::= true \mid m \mid \neg f \mid f + f \mid f \vee f \mid f \oplus f,$$

where m is a PIL monomial, $\oplus$ is the *union* operator, $\vee$ is the *disjunction* operator, $+$ is the *coalescing* operator and $\neg$ is the *complementation* operator. As usual, we write $f_1 \Rightarrow f_2 \stackrel{def}{=} \neg f_1 \oplus f_2$.

The language of PCL formulas is generated from PIL formulas—in the form of disjunctions of monomials. In contrast to PIL formulas which represent interaction sets, PCL formulas represent configuration sets, where a *configuration* is a non-empty set of interactions. Let P be a set of ports. The semantic domain of PCL is the lattice of configuration sets $CS(P) = 2^{C(P) \setminus \{\emptyset\}}$ (Figure 2(c)).

Semantics. *The meaning of a PCL formula f is defined by the following satisfaction relation. Let $\emptyset \neq \gamma \in C(P)$ be a configuration. We define:*

$\gamma \models true,$	*for all γ,*
$\gamma \models m,$	*if $\forall a \in \gamma, a \models_i m$ (where $\models_i$ 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 $\gamma_2 \models f_2$,*
$\gamma \models f_1 \oplus f_2,$	*if $\gamma \models f_1$ or $\gamma \models f_2$,*
$\gamma \models f_1 \vee f_2,$	*if $\gamma \models f_1 + f_2$ or $\gamma \models f_1 \oplus f_2$,*
$\gamma \models \neg f,$	*if $\gamma \not\models f$ (i.e. $\gamma \models f$ does not hold).*

Two formulas are equal $f_1 = f_2$ iff, for all $\emptyset \neq \gamma \in C(P)$, $\gamma \models f_1 \Leftrightarrow \gamma \models f_2$.

Proposition 2.1. *Equality is a congruence w.r.t. to all PCL operations.*

PCL syntax does not allow direct representation of all PIL formulas. Nonetheless, any PIL formula can be put in the disjunctive normal form—a syntactically correct PCL formula. We call *interaction formulas* those PCL formulas that are also syntactically PIL formulas. We have shown that PCL is a conservative extension of PIL, i.e. the meaning of an interaction formula in PIL is the same as in PCL.

Coalescing $+$ is the key operator in PCL. It combines configurations, as opposed to the union operator $\oplus$, which combines configuration sets. Coalescing is associative, commutative, idempotent and has an absorbing element $false \stackrel{def}{=} \neg true$. Coalescing with *true* presents a particular interest for writing specifications, since they allow adding any set of interactions to the configurations satisfying f. For any formula f, the *closure operator* $\sim$ is defined by putting $\sim f \stackrel{def}{=} f + true$.

Proposition 2.2. *For any formula f, holds the equality $\sim\sim f = \sim f$.*

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 f, holds the equality $\sim \neg f = \neg f$, which implies $\neg \sim \neg f = \neg\neg f = f$. However, this is not true in general.

The following proposition allows us to address the relation between complementation and negation. Indeed, complementation is not an extension of PIL negation. Furthermore, PIL negation cannot be applied directly to an arbitrary PCL formula. However, for any interaction formula $f = \bigvee_{i \in I} m_i$, its negation $\overline{f}$ can be put in the form of disjunction of monomials and considered as a formula of PCL.

Proposition 2.3. *For any interaction formula f, holds $f \oplus \overline{f} \oplus (\overline{f} + f) = true$.*

The three terms on the left are mutually disjoint. Thus, for any interaction formula f, we have $\neg f = \overline{f} \oplus (f + \overline{f}) = \overline{f} + true = {\sim}\overline{f}$ and, consequently, $\neg\overline{f} = {\sim}f$ and ${\sim}\neg f = \neg f$.

In [10], we present a sound and complete axiomatization of the PCL equality $=$, which allows us to define a normal form for PCL formulas. The existence of such a normal form immediately implies the decidability of 1) the equality of two PCL formulas and 2) the satisfaction of a formula by a configuration.

Definition 2.4. A formula is in *normal form* iff it has the form $\bigoplus_{i\in I}\sum_{j\in J_i} m_{i,j}$, with all $m_{i,j}$ monomials.

In order to specify rich and flexible architecture styles, we extend PCL to first and second order, introducing quantification over component and component-set variables, respectively. We assume that all components in the system have types representing their interfaces. Notation $c:T$ means that component c has type T; $C:T$ means that all components belonging to C are of type T. Let $\mathscr{T}$ be the set of all types. We denote $c.P$ the set of ports of the component c. Similarly, we write $c.p$ to denote the port p of component c. The following notation expresses an exact interaction, i.e. all ports in the arguments must participate in the interaction and all other ports of the system cannot participate in the interaction:

$$\sharp(c_1.p_1,\ldots,c_n.p_n) \stackrel{def}{=} \bigwedge_{i\in[1,n]} c_i.p_i \;\wedge\; \bigwedge_{i\in[1,n]}\ \bigwedge_{p\in c_i.P\setminus\{p_i\}} \overline{c_i.p} \;\wedge\; \bigwedge_{T\in\mathscr{T}} \Big(\forall c:T(c\notin\{c_1,\ldots,c_n\}).\ \bigwedge_{p\in c.P}\overline{c.p}\Big).$$

Example 2.5 (Star)**.** One central component s is connected to every other component through a binary interaction and there are no other interactions:

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

The three conjuncts of this formula express respectively the properties: 1) any component is connected to s; 2) components other than s are not connected; and 3) unary interactions are forbidden.

Example 2.6 (Pipes and Filters)**.** Consider 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:

$$\begin{aligned}
&\forall f:F.\ \exists p:P.\ {\sim}(f.in\ p.out)\wedge\forall p':P(p\neq p').\ \big(\overline{f.in}\vee\overline{p'.out}\big)\ \wedge\\
&\forall f:F.\ \exists p:P.\ {\sim}(f.out\ p.in)\wedge\forall p':P(p\neq p').\ \big(\overline{f.out}\vee\overline{p'.in}\big)\ \wedge\\
&\forall p:P.\ \exists f:F.\ \forall f':F(f\neq f').\ (\overline{p.out}\vee\overline{f'.in}).
\end{aligned}$$

Example 2.7 (Ring)**.** All components form a single ring by connecting their *in* and *out* ports:

$$\begin{aligned}
&\Big(\Sigma c:T.\ \exists c':T(c\neq c').\ \sharp(c.in\ c'.out)+\Sigma c:T.\ \exists c':T(c\neq c').\ \sharp(c.out\ c'.in)\Big)\\
&\qquad\wedge\ \forall C:T.\ \forall c:T(c\notin C).\Big(\exists c':T(c'\in C).\ \exists c'':T(c''\notin C).\ {\sim}(c'.in\ c''.out)\Big).
\end{aligned}$$

3 Discussion and related work

A plethora of approaches exist for characterizing architecture styles. Among the formal approaches for representing and analysing architecture descriptions, we distinguish two main categories:

Extensional approaches. Every object needed for the specification, i.e. the connections inducing interactions among the components, is explicitly defined. All connections, other than the ones specified, are excluded. Most ADLs, for instance SOFA [9], Wright [2], XCD [11], adopt this approach.

Intentional approaches: Connections among the components are not explicitly specified, but derived from a set of logical constraints, formulating the intentions of the designer.

The proposed framework encompasses both approaches. It allows individual interactions, e.g. by using interaction formulas. It also allows specification of configuration sets, e.g. by using formulas of the form $\sim f$. The proposed framework has similarities but also significant differences with work on Darwin [8] and ACME [7], where Alloy[1] is used to specify and check architecture topologies. It differs in that it achieves a strong semantic integration between architectures and architecture styles.

The presented work is a contribution to a long-term research program that aims at developing the BIP component framework. 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 [3]. The presented work nicely complements the existing component framework with logics for the specification of architecture styles. Quantification over components and sets of components allows the genericity needed for architecture styles. In future work, we will extend the theoretical results in two directions. First, to allow data transfer specification associated with interactions. Second, to allow description of hierarchically structured interactions.

References

[1] *Alloy*. `http://alloy.mit.edu/alloy/`.

[2] R. Allen & D. Garlan (1994): *Formalizing architectural connection*. In: *Proceedings of the 16th international conference on Software engineering*, IEEE Computer Society Press, pp. 71–80.

[3] P. Attie, E. Baranov, S. Bliudze, M. Jaber & J. Sifakis (2014): *A General Framework for Architecture Composability*. In: *SEFM 2014*, *LNCS* 8702, Springer, pp. 128–143.

[4] S. Bliudze & J. Sifakis (2008): *The Algebra of Connectors—Structuring Interaction in BIP*. *IEEE Transactions on Computers* 57(10), pp. 1315–1330, doi:10.1109/TC.2008.26.

[5] S. Bliudze & J. Sifakis (2010): *Causal semantics for the algebra of connectors*. *FMSD* 36(2), pp. 167–194.

[6] S. Bliudze & J. Sifakis (2011): *Synthesizing Glue Operators from Glue Constraints for the Construction of Component-Based Systems*. In: *SC'11*, *LNCS* 6708, Springer, pp. 51–67, doi:10.1007/978-3-642-22045-6_4.

[7] D. Garlan, R. Monroe & D. Wile (1997): *Acme: An Architecture Description Interchange Language*. In: *CASCON '97*, IBM Press, pp. 159–173.

[8] I. Georgiadis, J. Magee & J. Kramer (2002): *Self-organising software architectures for distributed systems*. In: *Proc of the 1st workshop on Self-healing systems*, ACM, pp. 33–38.

[9] T. Kalibera & P. Tuma (2002): *Distributed component system based on architecture description: The Sofa experience*. In: *On the Move to Meaningful Internet Systems 2002*, Springer, pp. 981–994.

[10] A. Mavridou, E. Baranov, S. Bliudze & J. Sifakis (2015): *Configuration Logics - Modelling Architecture Styles*. Technical Report EPFL-REPORT-206825, EPFL IC IIF RiSD. Available at: `http://infoscience.epfl.ch/record/206825`; paper submitted to SEFM 2015.

[11] M. Ozkaya & Ch. Kloukinas (2014): *Design-by-contract for reusable components and realizable architectures*. In: *CBSE'14*, ACM, pp. 129–138.