



Architecture Modelling of Parametric Component-Based Systems

Maria Pittou and George Rahonis^(✉)



Department of Mathematics, Aristotle University of Thessaloniki,
54124 Thessaloniki, Greece
{mpittou,grahonis}@math.auth.gr

Abstract. We study formal modelling of architectures applied on parametric component-based systems consisting of an unknown number of instances of each component. Architecture modelling is achieved by means of logics. We introduce an extended propositional interaction logic and investigate its first-order level which serves as a formal language for the interactions of parametric systems. Our logic effectively describes the execution order of interactions which is a main feature in several important architectures. We state the decidability of equivalence, satisfiability, and validity of first-order extended interaction logic formulas, and provide several examples of formulas describing well-known architectures.

Keywords: Architecture modelling · Parametric component-based systems · First-order extended interaction logic

1 Introduction

Developing well-founded modelling techniques is a challenging task for large and complex systems. Rigorous formalisms in systems engineering are mainly component-based that allow reconfigurability and validation [8]. Component-based design lies in constructing multiple components which coordinate in order to generate the global model for a system [8, 20]. Therefore, defining the communication patterns of systems is one of the key aspects in modelling process. Coordination principles can be specified by means of architectures that characterize the permissible interactions and their implementation order as well as the topology, of the system's components [28, 34]. Architectures have been proved important in systems modelling since they enforce design rules on the components, and hence ensure correctness by construction with respect to basic properties such as deadlock freedom and mutual exclusion [7, 10, 28].

M. Pittou—  The research work was supported by the Hellenic Foundation for Research and Innovation (HFRI) under the HFRI PhD Fellowship grant (Fellowship Number: 1471).

© IFIP International Federation for Information Processing 2020

Published by Springer Nature Switzerland AG 2020

S. Bliudze and L. Bocchi (Eds.): COORDINATION 2020, LNCS 12134, pp. 281–300, 2020.

https://doi.org/10.1007/978-3-030-50029-0_18

In this paper we provide a formal framework for the architecture modelling of parametric component-based systems using a first-order logic. Parametric systems represent a wide class of component-based systems including communication protocols and concurrent and distributed algorithms [1, 9, 17]. Parametric systems are constructed by a finite number of component types each consisting of an unknown number of instances [4, 9]. We address the problem of modelling the ordering restrictions for the components' connections, an important aspect of several parametric architectures, including Publish/Subscribe and Request/Response [15, 38]. For instance, in a Request/Response architecture a service needs firstly to enroll in the service registry and then receives requests from the interested clients. On the other hand, several services fulfilling the same task, maybe be enrolled in the registry in any order. We model components with the standard formalism of labelled transitions systems (cf. [2, 3, 8, 22]) where communication is performed by their set of labels, called ports, and is defined by interactions, i.e., sets of ports. Then, architectures are modelled by logic formulas encoding allowed interactions and their execution order. Briefly, the contributions of our work are the following:

- (1) We introduce *Extended Propositional Interaction Logic* (EPIL for short) over a finite set of ports, which augments PIL from [28] with two operators namely the concatenation $*$ and the shuffle operator \sqcup . In contrast to PIL, where the satisfaction relation is checked against interactions, EPIL formulas are interpreted over finite words whose letters are interactions. Intuitively, the semantics of concatenation and shuffle operator specifies the execution of consecutive and interleaving interactions, respectively. We apply EPIL formulas for formalizing *Blackboard* [14], *Request/Response* [15], and *Publish/Subscribe* [18] architectures.
- (2) We introduce *First-Order Extended Interaction Logic* (FOEIL for short), as a modelling language for the architectures of parametric systems. The syntax of FOEIL is over typed variables and is equipped with the syntax of EPIL, the common existential and universal quantifiers, and four new quantifiers, namely existential and universal concatenation and shuffle quantifiers. The new quantifiers achieve to encode the partial and whole participation of component instances in sequential and interleaved interactions of parametric architectures.
- (3) We show the expressiveness of FOEIL by examples for architectures of parametric component-based systems. Specifically, we consider the architectures *Blackboard*, *Request/Response*, and *Publish/Subscribe*, that impose orders on the implementation of their interactions.
- (4) We state an effective translation of FOEIL formulas to finite automata and prove the decidability of equivalence, validity and satisfiability for FOEIL sentences.

The structure of the paper is as follows. In Sect. 2 we discuss related work and in Sect. 3 we recall the basic notions for component-based systems and interactions. Then, in Sect. 4 we introduce the syntax and semantics of EPIL and present

three examples of architectures defined by EPIL formulas. In Sect. 5 we introduce the syntax and semantics of our FOEIL and provide examples of FOEIL sentences describing concrete parametric architectures. Section 6 deals with the decidability results for FOEIL sentences. Finally, in Conclusion, we present open problems and future work.

2 Related Work

In [28] the authors introduced a Propositional Configuration Logic (PCL) as a modelling language for the description of architectures. First- and second-order configuration logic was considered for parametric architectures (called styles of architectures in that paper). PCL which was interpreted over sets of interactions has a nice property: for every PCL formula an equivalent one in full normal form can be constructed. This implied the decidability of equivalence of PCL formulas in an automated way. Though PCL does not describe the order of interactions required by architectures as it is done by our logics, EPIL and FOEIL.

In [26] the first-order level of PIL, namely First-Order Interaction Logic (FOIL) was introduced to describe finitely many interactions, for parametric systems in BIP (cf. [8]). FOIL applied for modelling classical architectures (Star, Ring etc.) and contributed to model checking of parametric systems. Monadic Interaction Logic (MIL) was introduced in [10] as an alternative logic for the interactions of parametric systems. MIL was used for the description of parametric rendezvous and broadcast communication and applied for developing an automated method for detecting deadlocks. In the same line, in [11], an Interaction Logic with One Successor (IL1S) was developed for describing rendezvous and broadcast communications, and the architectures of parametric systems. IL1S was proved to be decidable and used for checking correctness of safety properties of parametric systems. FOIL, MIL, and IL1S, have been proved satisfactory for formalizing communication and architectures in parametric systems, though without capturing any order restrictions, as required by each architecture.

One of the main features in BIP framework is “priorities among interactions” (cf. [8]). A priority system is determined by a strict partial order \prec among the set of permitted interactions. If $a \prec a'$ for two interactions a and a' , then a' must be implemented before a since it has bigger priority. Clearly the set of strings of interactions satisfying an EPIL sentence containing a shuffle operator, cannot be obtained by any strict partial order among the set of interactions.

In [5] the authors established a strict framework for architectures composability. There, architectures were considered as operators enforcing properties to semantics of systems’ components. Preservation of safety and liveness properties was also studied for composed architectures. The subsequent work in [7] investigated architectures of composed-based systems with data and conditions under which safety properties are preserved. In both works the required order of the interactions’ execution in architectures has not been considered.

Hennessy and Milner introduced in 1985 (cf. [23]) a logic, called HML, as a calculus for the specification of concurrent programs and their properties.

In [19] the authors studied μHML , i.e., HML with least and greatest fixpoints and focused on a fragment of that logic that is monitored for runtime verification of programs' execution. μHML succeeded to describe simple client/server processes but it is far from describing complex architectures. Specifically our shuffle operator cannot be described in μHML .

In [20] the authors introduced the Components and Behaviors (CAB) process calculus which extended BIP with dynamic capabilities and showed the expressiveness of its priorities. The paper studied dynamic composition of sub-components based on the calculus language which though does not cover the architecture of the compound system.

Distributed systems were investigated in the setup of pomsets in [21] (cf. also [36]) where the execution order of interactions was considered. Though, due to the imposed orders of pomsets, our shuffle operation cannot be sufficiently described in this framework. For instance, the subformula $\varphi_1 \sqcup \varphi_2$ of the EPIL formula φ describing the Publish/Subscribe architecture (cf. Example 3) cannot be described by means of pomsets.

Multiparty session types described efficiently communication protocols and their interactions patterns (cf. for instance [24, 25]). The relation among multiparty session types and communicating automata was studied in [16]. Parameterized multiparty session types were investigated in [13, 17]. Nevertheless, the work of [17] did not study the implementation order of the parameterized interactions and the models of [13, 16, 24, 25] did not consider the architectures of the systems.

Finally, an architectural design rewriting model for the development and reconfiguration of software architectures was introduced in [12]. Though, no order of interactions' execution was considered.

3 Preliminaries

For every natural number $n \geq 1$ we denote by $[n]$ the set $\{1, \dots, n\}$. For every set S we denote by $\mathcal{P}(S)$ the powerset of S . Let A be an alphabet, i.e., a finite nonempty set. We denote by A^* the set of all finite words over A and we let $A^+ = A^* \setminus \{\varepsilon\}$ where ε denotes the empty word. Given $w, u \in A^*$, the shuffle product $w \sqcup u$ of w and u is a language over A defined by $w \sqcup u = \{w_1 u_1 \dots w_m u_m \mid w_1, \dots, u_m \in A^* \text{ and } w = w_1 \dots w_m, u = u_1 \dots u_m\}$.

A component-based system consists of a finite number of components of the same or different type. We define components by labelled transition systems (LTS for short) like in well-known component-based modelling frameworks including BIP [8], REO [3], X-MAN [22], and B [2].

Formally, an *atomic component* is an LTS $B = (Q, P, q_0, R)$ where Q is a finite set of *states*, P is a finite set of *ports*, q_0 is the *initial state* and $R \subseteq Q \times P \times Q$ is the set of *transitions*. We call an atomic component B a *component*, when we deal with several atomic components. For every set $\mathcal{B} = \{B(i) \mid i \in [n]\}$ with $B(i) = (Q(i), P(i), q_0(i), R(i))$, $i \in [n]$, we assume that $(Q(i) \cup P(i)) \cap (Q(i') \cup P(i')) = \emptyset$ for every $1 \leq i \neq i' \leq n$.

Here we focus only on the communication patterns of systems' components, using the terminology of BIP for the basic notions. Communication is achieved through components' interfaces. The interface of an LTS corresponds to its set of labels, called ports. Then, communications of components are defined by interactions, i.e., sets of ports, that can be represented by formulas of *propositional interaction logic* (PIL for short) [10, 11, 28]. Hence, firstly we need to recall PIL.

Let P be a nonempty finite set of *ports*. Then $I(P) = \mathcal{P}(P) \setminus \{\emptyset\}$ is the set of interactions over P . The syntax of PIL formulas ϕ over P is given by the grammar $\phi ::= \text{true} \mid p \mid \neg\phi \mid \phi \vee \phi$ where $p \in P$. We set $\text{false} = \neg\text{true}$ and $\neg(\neg\phi) = \phi$, $\phi \wedge \phi' := \neg(\neg\phi \vee \neg\phi')$, $\phi \rightarrow \phi' := \neg\phi \vee \phi'$ for PIL formulas ϕ, ϕ' over P . PIL formulas are interpreted over interactions in $I(P)$. For every PIL formula ϕ and $a \in I(P)$ we define the satisfaction relation $a \models_{\text{PIL}} \phi$ by induction on the structure of ϕ as follows:

$$\begin{array}{ll} a \models_{\text{PIL}} \text{true}, & a \models_{\text{PIL}} \neg\phi \text{ iff } a \not\models_{\text{PIL}} \phi, \\ a \models_{\text{PIL}} p \text{ iff } p \in a, & a \models_{\text{PIL}} \phi_1 \vee \phi_2 \text{ iff } a \models_{\text{PIL}} \phi_1 \text{ or } a \models_{\text{PIL}} \phi_2. \end{array}$$

Note that PIL differs from propositional logic, since it is interpreted over interactions, and thus the name “interaction” is assigned to it.

Two PIL formulas ϕ, ϕ' are called equivalent, denoted by $\phi \equiv \phi'$, when $a \models \phi$ iff $a \models \phi'$ for every $a \in I(P)$. For every $a = \{p_1, \dots, p_l\} \in I(P)$ we consider the PIL formula $\phi_a = p_1 \wedge \dots \wedge p_l$. Then, $a \models_{\text{PIL}} \phi_a$, and for every $a, a' \in I(P)$ we get $a = a'$ iff $\phi_a \equiv \phi_{a'}$. We can describe a set of interactions as a PIL formula. Specifically for $\gamma = \{a_1, \dots, a_m\}$, the PIL formula ϕ_γ of γ is $\phi_\gamma = \phi_{a_1} \vee \dots \vee \phi_{a_m}$.

Let $\mathcal{B} = \{B(i) \mid i \in [n]\}$ and set $P_{\mathcal{B}} = \bigcup_{i \in [n]} P(i)$. An *interaction of \mathcal{B}* is an interaction $a \in I(P_{\mathcal{B}})$ such that $|a \cap P(i)| \leq 1$, for every $i \in [n]$. We denote by $I_{\mathcal{B}}$ the set of all interactions of \mathcal{B} , i.e.,

$$I_{\mathcal{B}} = \{a \in I(P_{\mathcal{B}}) \mid |a \cap P(i)| \leq 1 \text{ for every } i \in [n]\}.$$

Definition 1. A component-based system is a pair (\mathcal{B}, γ) where $\mathcal{B} = \{B(i) \mid i \in [n]\}$ is a set of components, with $B(i) = (Q(i), P(i), q_0(i), R(i))$ for every $i \in [n]$, and γ is a set of interactions in $I_{\mathcal{B}}$.

The set γ of interactions of (\mathcal{B}, γ) specifies the architecture of the system. Obviously, we can replace γ by its corresponding PIL formula ϕ_γ , i.e., in a logical directed notation. Expression of software architectures by logics has been used in several works and gave nice results (cf. [10, 11, 28]).

4 Extended Propositional Interaction Logic

PIL describes nicely several architectures but its semantics does not capture the execution order of the interactions imposed by each architecture. Ordered interactions occur in common architectures, including Request/Response and Publish/Subscribe. In this section, we introduce a propositional logic that extends PIL with two operators, the concatenation $*$ and the shuffle operator \sqcup , and we model architectures of component-based systems with order restrictions.

Definition 2. Let P be a finite set of ports. The syntax of extended propositional interaction logic (EPIL for short) formulas φ over P is given by the grammar

$$\begin{aligned}\zeta &::= \phi \mid \zeta * \zeta \\ \varphi &::= \zeta \mid \neg \zeta \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \varphi * \varphi \mid \varphi \sqcup \varphi\end{aligned}$$

where ϕ is a PIL formula over P .

The binding strength, in decreasing order, of the EPIL operators is: negation, shuffle, concatenation, conjunction, and disjunction. Negation is applied only in PIL formulas and EPIL formulas of type ζ . The latter ensures exclusion of erroneous interactions in architectures. The restricted use of negation allows a reasonable complexity of translation of FOEIL formulas to finite automata. This in turn implies the decidability of equivalence, satisfiability, and validity of FOEIL sentences (cf. Sect. 6). Our assumption has no impact in expressiveness of EPIL formulas, since they can efficiently model most known architectures.

EPIL formulas are interpreted over finite words $w \in I(P)^*$. Intuitively, a word w encodes each of the distinct interactions within a system as a letter. Moreover, the position of each letter in w depicts the order in which the corresponding interaction is executed in the system, in case there is an order restriction.

Definition 3. Let φ be an EPIL formula over P and $w \in I(P)^*$. If $w = \varepsilon$ and $\varphi = \text{true}$, then we set $w \models \text{true}$. If $w \in I(P)^+$, then we define the satisfaction relation $w \models \varphi$ by induction on the structure of φ as follows:

- $w \models \phi$ iff $w \models_{\text{PIL}} \phi$,
- $w \models \zeta_1 * \zeta_2$ iff there exist $w_1, w_2 \in I(P)^*$ such that $w = w_1 w_2$ and $w_i \models \zeta_i$ for $i = 1, 2$,
- $w \models \neg \zeta$ iff $w \not\models \zeta$,
- $w \models \varphi_1 \vee \varphi_2$ iff $w \models \varphi_1$ or $w \models \varphi_2$,
- $w \models \varphi_1 \wedge \varphi_2$ iff $w \models \varphi_1$ and $w \models \varphi_2$,
- $w \models \varphi_1 * \varphi_2$ iff there exist $w_1, w_2 \in I(P)^*$ such that $w = w_1 w_2$ and $w_i \models \varphi_i$ for $i = 1, 2$,
- $w \models \varphi_1 \sqcup \varphi_2$ iff there exist $w_1, w_2 \in I(P)^*$ such that $w \in w_1 \sqcup w_2$ and $w_i \models \varphi_i$ for $i = 1, 2$.

If $\varphi = \phi$ is a PIL formula, then $w \models \phi$ implies that w is a letter in $I(P)$. Two EPIL formulas φ, φ' are called *equivalent*, denoted by $\varphi \equiv \varphi'$, when $w \models \varphi$ iff $w \models \varphi'$ for every $w \in I(P)^*$. Now, we define an updated version of component-based systems by replacing the PIL formula by an EPIL formula. Specifically, a *component-based system* is a pair (\mathcal{B}, φ) where $\mathcal{B} = \{B(i) \mid i \in [n]\}$ is a set of components and φ is an EPIL formula over $P_{\mathcal{B}}$. The investigation of semantics and verification of component-based systems is a part of future work.

Next we present three examples of component-based models (\mathcal{B}, φ) whose architectures have ordered interactions encoded by EPIL formulas satisfied by words over $I_{\mathcal{B}}$. Clearly, there exist several variations of the following architectures

and their order restrictions, that EPIL formulas could also model sufficiently by applying relevant modifications. We need to define the following macro EPIL formula. Let $P = \{p_1, \dots, p_n\}$ be a set of ports. Then, for $p_{i_1}, \dots, p_{i_m} \in P$ with $m < n$ we let

$$\#(p_{i_1} \wedge \dots \wedge p_{i_m}) ::= p_{i_1} \wedge \dots \wedge p_{i_m} \wedge \bigwedge_{p \in P \setminus \{p_{i_1}, \dots, p_{i_m}\}} \neg p.$$

Example 1 (Blackboard). We consider a component-based system (\mathcal{B}, φ) with the Blackboard architecture. The latter is applied in planning and scheduling [35] as well as in artificial intelligence [6]. Blackboard architecture involves a blackboard, a controller and the (knowledge) sources components [14, 29]. Blackboard presents the state of the problem to be solved and sources provide partial solutions without knowing about the existence of other sources. When there is enough information for a source to provide its partial solution, the source is triggered, i.e., is keen to write on the blackboard. Since multiple sources may be triggered, a controller component is used to resolve any conflicts.

We consider three knowledge sources components. Hence, $\mathcal{B} = \{B(i) \mid i \in [5]\}$ where $B(1), \dots, B(5)$ refer to blackboard, controller and the sources components, respectively (Fig. 1). Blackboard has two ports p_d, p_a to declare the state of the problem and add the new data as obtained by a source, respectively. Sources have three ports $p_{n_k}, p_{t_k}, p_{w_k}$, for $k = 1, 2, 3$, for being notified about the existing data on the blackboard, the trigger of the source, and for writing on the blackboard, respectively. Controller has three ports, p_r used to record blackboard data, p_l for logging the triggered sources, and p_e for their execution to blackboard. Here we assume that all sources are triggered, i.e., that they participate in the architecture. The EPIL formula φ for Blackboard architecture is

$$\begin{aligned} \varphi = & \#(p_d \wedge p_r) * \left(\#(p_d \wedge p_{n_1}) \sqcup \#(p_d \wedge p_{n_2}) \sqcup \#(p_d \wedge p_{n_3}) \right) * \\ & \left(\varphi_1 \vee \varphi_2 \vee \varphi_3 \vee (\varphi_1 \sqcup \varphi_2) \vee (\varphi_1 \sqcup \varphi_3) \vee (\varphi_2 \sqcup \varphi_3) \vee (\varphi_1 \sqcup \varphi_2 \sqcup \varphi_3) \right) \end{aligned}$$

where $\varphi_i = \#(p_l \wedge p_{t_i}) * \#(p_e \wedge p_{w_i} \wedge p_a)$ for $i = 1, 2, 3$.

The first PIL subformula encodes the connection among blackboard and controller. The EPIL subformula between the two $*$ operators represents the connections of knowledge sources to blackboard. The last part of φ captures the connection of some of knowledge sources with controller and blackboard. The use of $*$ operator in φ ensures that the controller is informed before the sources, and that sources are triggered before writing on blackboard. The shuffle operator in φ captures any possible order, among the sources, for connecting with controller and blackboard.

Before our second example we show the expressive difference among EPIL and PCL formulas of [28].

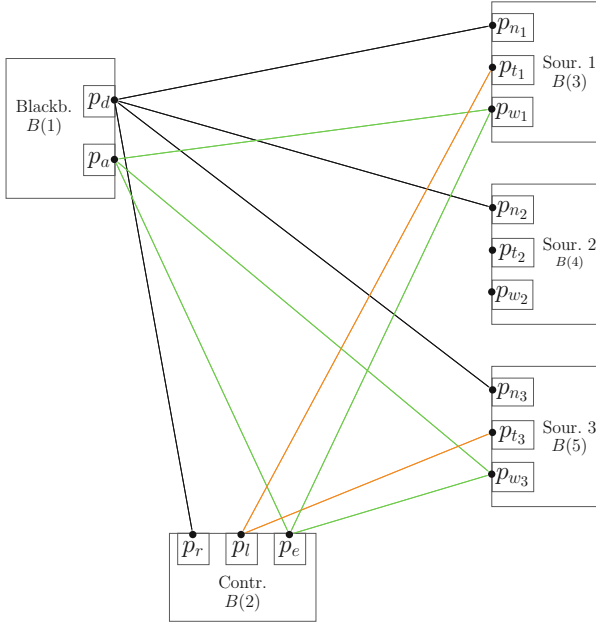


Fig. 1. Blackboard architecture. A possible execution for the interactions.

Remark 1. Consider the Blackboard architecture presented in the previous example. Then the corresponding PCL formula ρ (cf. [28]) describing that architecture is

$$\begin{aligned} \rho = & \#(p_d \wedge p_r) + \#(p_d \wedge p_{n_1}) + \#(p_d \wedge p_{n_2}) + \#(p_d \wedge p_{n_3}) + \\ & (\phi_1 \sqcup \phi_2 \sqcup \phi_3 \sqcup (\phi_1 + \phi_2) \sqcup (\phi_1 + \phi_3) \sqcup (\phi_2 + \phi_3) \sqcup (\phi_1 + \phi_2 + \phi_3)) \end{aligned}$$

where $+$ denotes the coalescing operator, \sqcup denotes the union operator, and

$$\phi_i = \#(p_l \wedge p_{t_i}) + \#(p_e \wedge p_{w_i} \wedge p_a)$$

for $i = 1, 2, 3$.

Then, the PCL formula ρ is interpreted over sets of interactions in $\mathcal{P}(I(P)) \setminus \{\emptyset\}$ which trivially cannot express the required order of the execution of the interactions. For instance the set of interactions $\{\{p_d, p_r\}, \{p_d, p_{n_1}\}, \{p_d, p_{n_2}\}, \{p_d, p_{n_3}\}, \{p_l, p_{t_2}\}, \{p_e, p_{w_2}, p_a\}\}$ satisfies ρ but represents no order of the interactions' execution.

Example 2 (Request/Response). Request/Response architectures refer to services and clients, and are classical interaction patterns widely used for web services [15]. Services become available to clients by enrolling in the so-called service registry. Then clients scan the registry and choose a service. Each client that is interested in a service sends a request and waits until the service's respond. Meanwhile no other client is connected to the service. To achieve this, in [28] a third component called coordinator was added for each service.

For our example we consider seven components, namely the service registry, two services with their coordinators, and two clients. (Fig. 2). Service registry has the ports p_e, p_u , and p_t , for the services' enrollment and for allowing a client to search and take the service's address, respectively. Services have the ports $p_{r_k}, p_{g_k}, p_{s_k}$, for $k = 1, 2$, for enrolling to service registry, and connecting to a client (via coordinator) for receiving a request and responding, respectively. Clients have the ports p_{l_k}, p_{o_k} for connecting with service registry to look up and obtain a service's address, while the ports p_{n_k}, p_{q_k} and p_{c_k} express the connection of the client to coordinator, to service (via coordinator) for sending the request and for collecting its response, respectively, for $k = 1, 2$. Coordinators have three ports, p_{m_k} for controlling that only one client is connected to a service, p_{a_k} for acknowledging that the connected client sends a request, and p_{d_k} that disconnects the client when the service responds to the request, for $k = 1, 2$. The EPIL formula φ for the Request/Response architecture equals to

$$\begin{aligned} & (\#(p_e \wedge p_{r_1}) \sqcup \#(p_e \wedge p_{r_2})) * (\xi_1 \sqcup \xi_2) * \left(\left((\varphi_{11} \vee \varphi_{21} \vee (\varphi_{11} * \varphi_{21}) \vee (\varphi_{21} * \varphi_{11})) \right) \vee \right. \\ & \left(\varphi_{12} \vee \varphi_{22} \vee (\varphi_{12} * \varphi_{22}) \vee (\varphi_{22} * \varphi_{12}) \right) \vee \left((\varphi_{11} \vee \varphi_{21} \vee (\varphi_{11} * \varphi_{21}) \vee (\varphi_{21} * \varphi_{11})) \sqcup \right. \\ & \left. \left. (\varphi_{12} \vee \varphi_{22} \vee (\varphi_{12} * \varphi_{22}) \vee (\varphi_{22} * \varphi_{12})) \right) \right) \end{aligned}$$

where $\xi_i = \#(p_{l_i} \wedge p_u) * \#(p_{o_i} \wedge p_t)$ for $i = 1, 2$, and $\varphi_{ij} = \#(p_{n_i} \wedge p_{m_j}) * \#(p_{q_i} \wedge p_{a_j}) * \#(p_{c_i} \wedge p_{d_j} \wedge p_{s_j})$ for $i = 1, 2$ (clients) and $j = 1, 2$ (services).

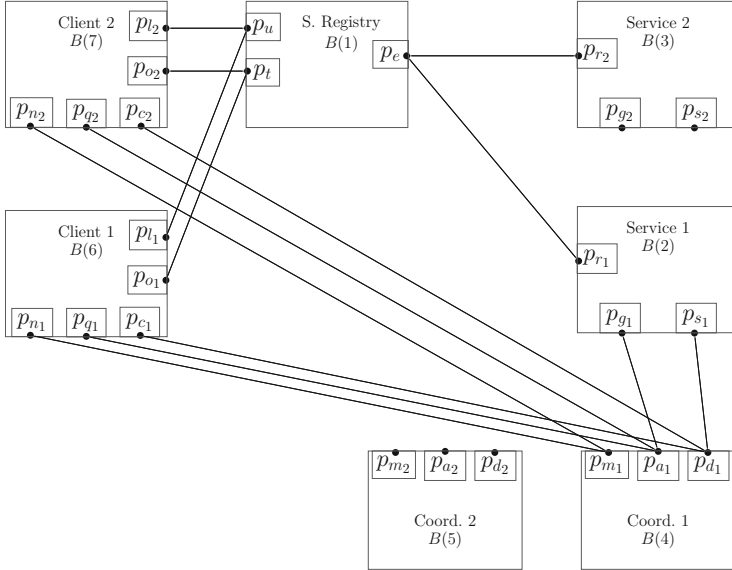


Fig. 2. Request/Response architecture. The omitted interactions are derived similarly.

The subformulas at the left of the first two concatenation operators encode the connections of the two services and the two clients with registry. Then, each of the three subformulas connected with the big disjunctions expresses that either one of the two clients or both of them (one at each time) are connected with the first service only, the second service only, or both of the services, respectively.

Example 3 (Publish/Subscribe). Publish/Subscribe architecture is used in IoT applications (cf. [30,31]), and recently in cloud systems [37] and robotics [27]. It involves publishers, subscribers, and topics components. Publishers advertise and transmit to topics the type of messages they produce. Then, subscribers are connected with topics they are interested in, and topics in turn transfer the messages from publishers to corresponding subscribers. Once a subscriber receives the requested message, it is disconnected from the relevant topic. Publishers cannot check the existence of subscribers and vice-versa [18].

We consider two publisher, two topic and three subscriber components (Fig. 3). Publishers have two ports, p_{a_k} and p_{t_k} , for $k = 1, 2$, for advertising and transferring their messages to topics, respectively. Topics are notified from the publishers and receive their messages through ports p_{n_k} and p_{r_k} , for $k = 1, 2$, respectively. Ports p_{c_k}, p_{s_k} and p_{f_k} , for $k = 1, 2$, are used from topics for the connection, the sending of a message, and disconnection with a subscriber, respectively. Subscribers use the ports $p_{e_m}, p_{g_m}, p_{d_m}$, for $m = 1, 2, 3$, for connecting with the topic (express interest), getting a message from the topic, and disconnecting from the topic, respectively. The EPIL formula for the Publish/Subscribe architecture is $\varphi = \varphi_1 \vee \varphi_2 \vee (\varphi_1 \sqcup \varphi_2)$ with

$$\varphi_i = \left((\xi_i * \varphi_{i1}) \vee (\xi_i * \varphi_{i2}) \vee (\xi_i * \varphi_{i3}) \vee (\xi_i * (\varphi_{i1} \sqcup \varphi_{i2})) \vee (\xi_i * (\varphi_{i1} \sqcup \varphi_{i3})) \vee \right. \\ \left. (\xi_i * (\varphi_{i2} \sqcup \varphi_{i3})) \vee (\xi_i * (\varphi_{i1} \sqcup \varphi_{i2} \sqcup \varphi_{i3})) \right)$$

for $i \in \{1, 2\}$ (topics) and $\xi_1 = \xi_{11} \vee \xi_{12} \vee (\xi_{11} \sqcup \xi_{12})$, $\xi_2 = \xi_{21} \vee \xi_{22} \vee (\xi_{21} \sqcup \xi_{22})$ encode that each of the two topics connects with the first publisher, or the second one, or with both of them, where $\xi_{ij} = \#(p_{n_i} \wedge p_{a_j}) * \#(p_{r_i} \wedge p_{t_j})$ for $i, j \in \{1, 2\}$, and $\varphi_{ij} = \#(p_{c_i} \wedge p_{e_j}) * \#(p_{s_i} \wedge p_{g_j}) * \#(p_{f_i} \wedge p_{d_j})$ for $i \in \{1, 2\}$ and $j \in \{1, 2, 3\}$, describes the connections of the two topics with each of the three subscribers.

The presented examples demonstrate that EPIL formulas can encode the order restrictions within architectures and also specify all the different instantiations for the connections among the coordinated components in the system.

5 Parametric Component-Based Systems

In this section we deal with the parametric extension of component-based systems defined by a finite number of distinct *component types* whose number of *instances* is a parameter for the system. In real world applications we do not need an unbounded number of components. Though, the number of instances

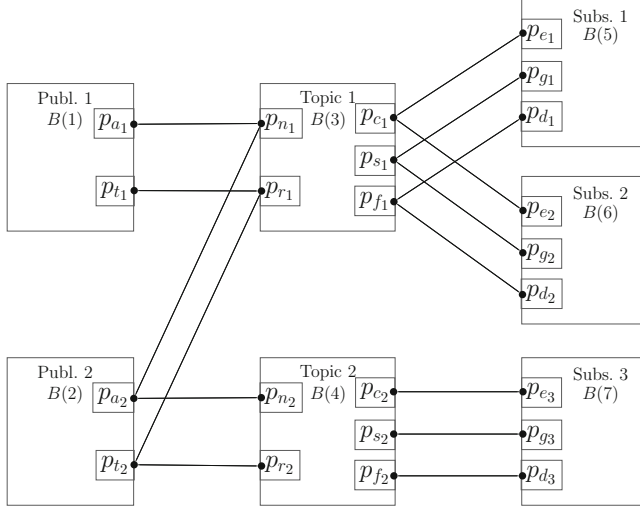


Fig. 3. Publish/Subscribe architecture. A possible execution for the interactions.

of every component type is unknown or it can be modified during a process. Next we consider parametric component-based systems, i.e., component-based systems with infinitely many instances of every component type.

Let $\mathcal{B} = \{B(i) \mid i \in [n]\}$ be a set of component types. For every $i \in [n]$ and $j \geq 1$ we consider a copy $B(i, j) = (Q(i, j), P(i, j), q_0(i, j), R(i, j))$ of $B(i)$, namely the j -th instance of $B(i)$. Hence, for every $i \in [n]$ and $j \geq 1$, the instance $B(i, j)$ is also a component and we call it a *parametric component* or a *component instance*. We assume that $(Q(i, j) \cup P(i, j)) \cap (Q(i', j') \cup P(i', j')) = \emptyset$ whenever $i \neq i'$ or $j \neq j'$ for every $i, i' \in [n]$ and $j, j' \geq 1$. This restriction permits us to use, without any confusion, the notation $P(i, j) = \{p(j) \mid p \in P(i)\}$ for every $i \in [n]$ and $j \geq 1$. We set $p\mathcal{B} = \{B(i, j) \mid i \in [n], j \geq 1\}$ and call it a set of *parametric components*, with $P_{p\mathcal{B}} = \bigcup_{i \in [n], j \geq 1} P(i, j)$.

Since parametric systems consist of an unknown number of component instances, we need a symbolic representation to describe their architectures. For this, we introduce the first-order extended interaction logic whose semantics describes the order of interactions implemented in a parametric architecture. Our logic is proved sufficient to model several complex architectures. This is important because parametric systems based on well-defined architectures satisfy most of their requirements [4, 9].

5.1 First-Order Extended Interaction Logic

We introduce the first-order extended interaction logic as a modelling language for describing the interactions of parametric component-based systems. For this, we equip EPIL formulas with variables. Due to the nature of parametric systems we need to distinguish variables referring to different component types.

Let $p\mathcal{B} = \{B(i, j) \mid i \in [n], j \geq 1\}$ and $\mathcal{X}^{(1)}, \dots, \mathcal{X}^{(n)}$ be pairwise disjoint countable sets of first-order variables referring to instances of component types $B(1), \dots, B(n)$, respectively. Variables in $\mathcal{X}^{(i)}$, for every $i \in [n]$, are denoted by small letters with the corresponding superscript, i.e., $x^{(i)} \in \mathcal{X}^{(i)}$, $i \in [n]$, is a first-order variable referring to an instance of component type $B(i)$. Let $\mathcal{X} = \mathcal{X}^{(1)} \cup \dots \cup \mathcal{X}^{(n)}$ and set $P_{p\mathcal{B}(\mathcal{X})} = \{p(x^{(i)}) \mid i \in [n], x^{(i)} \in \mathcal{X}^{(i)}, \text{ and } p \in P(i)\}$.

Definition 4. Let $p\mathcal{B} = \{B(i, j) \mid i \in [n], j \geq 1\}$ be a set of parametric components. Then the syntax of first-order extended interaction logic (FOEIL for short) formulas ψ over $p\mathcal{B}^1$ is given by the grammar

$$\begin{aligned} \psi ::= & \varphi \mid x^{(i)} = y^{(i)} \mid \neg(x^{(i)} = y^{(i)}) \mid \psi \vee \psi \mid \psi \wedge \psi \mid \psi * \psi \mid \psi \sqcup \psi \mid \\ & \exists x^{(i)}. \psi \mid \forall x^{(i)}. \psi \mid \exists^* x^{(i)}. \psi \mid \forall^* x^{(i)}. \psi \mid \exists^\sqcup x^{(i)}. \psi \mid \forall^\sqcup x^{(i)}. \psi \end{aligned}$$

where φ is an EPIL formula over $P_{p\mathcal{B}(\mathcal{X})}$, $i \in [n]$, $x^{(i)}, y^{(i)}$ are first-order variables in $\mathcal{X}^{(i)}$, \exists^* denotes the existential concatenation quantifier, \forall^* the universal concatenation quantifier, \exists^\sqcup is the existential shuffle quantifier, and \forall^\sqcup the universal shuffle quantifier. Furthermore, we assume that whenever ψ contains a subformula of the form $\exists^* x^{(i)}. \psi'$ or $\exists^\sqcup x^{(i)}. \psi'$, then the application of negation in ψ' is permitted only in PIL formulas and formulas of the form $x^{(j)} = y^{(j)}$.

Let ψ be a FOEIL formula over $p\mathcal{B}$. We denote by $\text{free}(\psi)$ the set of free variables of ψ . If ψ has no free variables, then it is a *sentence*. We consider a mapping $r : [n] \rightarrow \mathbb{N}$. The value $r(i)$, for every $i \in [n]$, represents the finite number of instances of the component type $B(i)$ in the parametric system. We let $p\mathcal{B}(r) = \{B(i, j) \mid i \in [n], j \in [r(i)]\}$ and call it the *instantiation* of $p\mathcal{B}$ w.r.t. r . Also $P_{p\mathcal{B}(r)} = \bigcup_{i \in [n], j \in [r(i)]} P(i, j)$ and $I_{p\mathcal{B}(r)} = \{a \in I(P_{p\mathcal{B}(r)}) \mid |a \cap P(i, j)| \leq 1 \text{ for every } i \in [n] \text{ and } j \in [r(i)]\}$.

Let $\mathcal{V} \subseteq \mathcal{X}$ be a finite set of first-order variables and set $P_{p\mathcal{B}(\mathcal{V})} = \{p(x^{(i)}) \in P_{p\mathcal{B}(\mathcal{X})} \mid x^{(i)} \in \mathcal{V}\}$. A (\mathcal{V}, r) -assignment is a mapping $\sigma : \mathcal{V} \rightarrow \mathbb{N}$ such that $\sigma(\mathcal{V} \cap \mathcal{X}^{(i)}) \subseteq [r(i)]$ for every $i \in [n]$, and $\sigma[x^{(i)} \rightarrow j]$ is the $(\mathcal{V} \cup \{x^{(i)}\}, r)$ -assignment which acts as σ on $\mathcal{V} \setminus \{x^{(i)}\}$ and assigns j to $x^{(i)}$. If φ is an EPIL formula over $P_{p\mathcal{B}(\mathcal{V})}$, then $\sigma(\varphi)$ is an EPIL formula over $P_{p\mathcal{B}(r)}$ which is obtained by φ by replacing every port $p(x^{(i)})$ in φ by $p(\sigma(x^{(i)}))$.

We interpret FOEIL formulas ψ over triples consisting of a mapping $r : [n] \rightarrow \mathbb{N}$, a (\mathcal{V}, r) -assignment σ , and a word $w \in I_{p\mathcal{B}(r)}^*$. The semantics of formulas of the form $\exists^* x^{(i)}. \psi$ and $\forall^* x^{(i)}. \psi$ (resp. $\exists^\sqcup x^{(i)}. \psi$ and $\forall^\sqcup x^{(i)}. \psi$) refer to satisfaction of ψ by subwords of w . The subwords correspond to component instances which are determined by the application of the assignment σ to $x^{(i)}$, and w results by the $*$ (resp. \sqcup) operator among the subwords.

Definition 5. Let ψ be a FOEIL formula over a set $p\mathcal{B} = \{B(i, j) \mid i \in [n], j \geq 1\}$ of parametric components and $\mathcal{V} \subseteq \mathcal{X}$ a finite set containing $\text{free}(\psi)$. Then for

¹ According to our terminology for EPIL formulas, a FOEIL formula should be defined over the set of ports of $p\mathcal{B}$. Nevertheless, we prefer for simplicity to refer to the set $p\mathcal{B}$ of parametric components.

every $r : [n] \rightarrow \mathbb{N}$, (\mathcal{V}, r) -assignment σ , and $w \in I_{p\mathcal{B}(r)}^*$ we define the satisfaction relation $(r, \sigma, w) \models \psi$, inductively on the structure of ψ as follows:

- $(r, \sigma, w) \models \varphi$ iff $w \models \sigma(\varphi)$,
- $(r, \sigma, w) \models x^{(i)} = y^{(i)}$ iff $\sigma(x^{(i)}) = \sigma(y^{(i)})$,
- $(r, \sigma, w) \models \neg(x^{(i)} = y^{(i)})$ iff $(r, \sigma, w) \not\models x^{(i)} = y^{(i)}$,
- $(r, \sigma, w) \models \psi_1 \vee \psi_2$ iff $(r, \sigma, w) \models \psi_1$ or $(r, \sigma, w) \models \psi_2$,
- $(r, \sigma, w) \models \psi_1 \wedge \psi_2$ iff $(r, \sigma, w) \models \psi_1$ and $(r, \sigma, w) \models \psi_2$,
- $(r, \sigma, w) \models \psi_1 * \psi_2$ iff there exist $w_1, w_2 \in I_{p\mathcal{B}(r)}^*$ such that $w = w_1 w_2$ and $(r, \sigma, w_i) \models \psi_i$ for $i = 1, 2$,
- $(r, \sigma, w) \models \psi_1 \sqcup \psi_2$ iff there exist $w_1, w_2 \in I_{p\mathcal{B}(r)}^*$ such that $w \in w_1 \sqcup w_2$ and $(r, \sigma, w_i) \models \psi_i$ for $i = 1, 2$,
- $(r, \sigma, w) \models \exists x^{(i)}. \psi$ iff there exists $j \in [r(i)]$ such that $(r, \sigma[x^{(i)} \rightarrow j], w) \models \psi$,
- $(r, \sigma, w) \models \forall x^{(i)}. \psi$ iff $(r, \sigma[x^{(i)} \rightarrow j], w) \models \psi$ for every $j \in [r(i)]$,
- $(r, \sigma, w) \models \exists^* x^{(i)}. \psi$ iff there exist $w_{l_1}, \dots, w_{l_k} \in I_{p\mathcal{B}(r)}^*$ with $1 \leq l_1 < \dots < l_k \leq r(i)$ such that $w = w_{l_1} \dots w_{l_k}$ and $(r, \sigma[x^{(i)} \rightarrow j], w_j) \models \psi$ for every $j = l_1, \dots, l_k$,
- $(r, \sigma, w) \models \forall^* x^{(i)}. \psi$ iff there exist $w_1, \dots, w_{r(i)} \in I_{p\mathcal{B}(r)}^*$ such that $w = w_1 \dots w_{r(i)}$ and $(r, \sigma[x^{(i)} \rightarrow j], w_j) \models \psi$ for every $j \in [r(i)]$,
- $(r, \sigma, w) \models \exists^{\sqcup} x^{(i)}. \psi$ iff there exist $w_{l_1}, \dots, w_{l_k} \in I_{p\mathcal{B}(r)}^*$ with $1 \leq l_1 < \dots < l_k \leq r(i)$ such that $w \in w_{l_1} \sqcup \dots \sqcup w_{l_k}$ and $(r, \sigma[x^{(i)} \rightarrow j], w_j) \models \psi$ for every $j = l_1, \dots, l_k$,
- $(r, \sigma, w) \models \forall^{\sqcup} x^{(i)}. \psi$ iff there exist $w_1, \dots, w_{r(i)} \in I_{p\mathcal{B}(r)}^*$ such that $w \in w_1 \sqcup \dots \sqcup w_{r(i)}$ and $(r, \sigma[x^{(i)} \rightarrow j], w_j) \models \psi$ for every $j \in [r(i)]$.

By definition of parametric systems, all instances of each component type are identical, hence the order specified above in the semantics of \exists^* , \forall^* , \exists^{\sqcup} , \forall^{\sqcup} quantifiers causes no restriction in the derived architecture.

If ψ is a FOEIL sentence over $p\mathcal{B}$, then we write $(r, w) \models \psi$. Let also ψ' be a FOEIL sentence over $p\mathcal{B}$. Then, ψ and ψ' are called *equivalent w.r.t. r* when $(r, w) \models \psi$ iff $(r, w) \models \psi'$, for every $w \in I_{p\mathcal{B}(r)}^*$.

In the sequel, we shall write also $x^{(i)} \neq y^{(i)}$ for $\neg(x^{(i)} = y^{(i)})$. Let β be a boolean combination of atomic formulas of the form $x^{(i)} = y^{(i)}$ and ψ a FOEIL formula over $p\mathcal{B}$. Then, we define $\beta \rightarrow \psi ::= \neg\beta \vee \psi$.

For simplicity we denote boolean combinations of formulas of the form $x^{(i)} = y^{(i)}$ as constraints. For instance we write $\exists x^{(i)} \forall y^{(i)} \exists x^{(j)} \forall y^{(j)} ((x^{(i)} \neq y^{(i)}) \wedge (x^{(j)} \neq y^{(j)})) . \psi$ for $\exists x^{(i)} \forall y^{(i)} \exists x^{(j)} \forall y^{(j)} . (((x^{(i)} \neq y^{(i)}) \wedge (x^{(j)} \neq y^{(j)})) \rightarrow \psi)$.

Note that in [28] the authors considered a universe of component types and hence, excluded in their logic formulas the erroneous types for each architecture. Such a restriction is not needed in our setting since we consider a well-defined set $[n]$ of component types for each architecture.

Definition 6. A parametric component-based system is a pair $(p\mathcal{B}, \psi)$ where $p\mathcal{B} = \{B(i, j) \mid i \in [n], j \geq 1\}$ is a set of parametric components and ψ is a FOEIL sentence over $p\mathcal{B}$.

In the sequel, we refer to parametric component-based systems simply as parametric systems. We remind that in this paper we focus on the architectures of parametric systems. The study of parametric systems' behavior is left for investigation in subsequent work as a part of parametric verification.

For our examples in the next subsection, we shall need the following macro FOEIL formula. Let $p\mathcal{B} = \{B(i, j) \mid i \in [n], j \geq 1\}$ and $1 \leq i_1, \dots, i_m \leq n$ be pairwise different indices. Then

$$\begin{aligned} \#(p_{i_1}(x^{(i_1)}) \wedge \dots \wedge p_{i_m}(x^{(i_m)})) &::= (p_{i_1}(x^{(i_1)}) \wedge \dots \wedge p_{i_m}(x^{(i_m)})) \wedge \\ \left(\bigwedge_{j=i_1, \dots, i_m} \bigwedge_{p \in P(j) \setminus \{p_j\}} \neg p(x^{(j)}) \right) &\wedge \left(\bigwedge_{j=i_1, \dots, i_m} \forall y^{(j)} (y^{(j)} \neq x^{(j)}) \cdot \bigwedge_{p \in P(j)} \neg p(y^{(j)}) \right) \wedge \\ &\left(\bigwedge_{k \in [n] \setminus \{i_1, \dots, i_m\}} \bigwedge_{p \in P(k)} \forall x^{(k)} \cdot \neg p(x^{(k)}) \right). \end{aligned}$$

The first $m - 1$ conjunctions express that the ports in the argument of $\#$ participate in the interaction. The double indexed conjunctions in the first pair of big parentheses disable all the other ports of the participating instances of components of type i_1, \dots, i_m described by $x^{(i_1)}, \dots, x^{(i_m)}$, respectively; conjunctions in the second pair of parentheses disable all ports of remaining instances of component types i_1, \dots, i_m . The last conjunct in the third line ensures that no ports in instances of remaining component types participate in the interaction.

5.2 Examples of FOEIL Sentences for Parametric Architectures

We present examples of FOEIL sentences describing parametric architectures, where the order of interactions is a main feature. We should note that FOEIL describes effectively as well, architectures with no order restrictions. Due to space limitations, we refer the reader to [32] for such examples.

Example 4 (Blackboard). The subsequent FOEIL sentence ψ encodes the interactions of Blackboard architecture, described in Example 1, in the parametric setting. We let $\mathcal{X}^{(1)}, \mathcal{X}^{(2)}, \mathcal{X}^{(3)}$ to be set of variables for blackboard, controller, and knowledge sources component instances, respectively.

$$\begin{aligned} \psi = \exists x^{(1)} \exists x^{(2)} \cdot &\left(\#(p_d(x^{(1)}) \wedge p_r(x^{(2)})) * \left(\forall^{\sqcup} x^{(3)} \cdot \#(p_d(x^{(1)}) \wedge p_n(x^{(3)})) \right) * \right. \\ &\left. \left(\exists^{\sqcup} y^{(3)} \cdot (\#(p_l(x^{(2)}) \wedge p_t(y^{(3)})) * \#(p_e(x^{(2)}) \wedge p_w(y^{(3)}) \wedge p_a(x^{(1)}))) \right) \right). \end{aligned}$$

Example 5 (Request/Response). We present a FOEIL sentence ψ for Request/Response architecture, described in Example 2, in the parametric setting. Let $\mathcal{X}^{(1)}, \mathcal{X}^{(2)}, \mathcal{X}^{(3)}$, and $\mathcal{X}^{(4)}$ refer to instances of service registry, service, client, and coordinator component, respectively. Then,

$$\begin{aligned} \psi = & \left(\exists x^{(1)}. \left((\forall^{\sqcup} x^{(2)}. \#(p_e(x^{(1)}) \wedge p_r(x^{(2)}))) * \right. \right. \\ & \left. \left. (\forall^{\sqcup} x^{(3)}. (\#(p_l(x^{(3)}) \wedge p_u(x^{(1)})) * \#(p_o(x^{(3)}) \wedge p_t(x^{(1)})))) \right) \right) * \\ & \left(\exists^{\sqcup} y^{(2)} \exists x^{(4)} \exists^* y^{(3)}. \xi \wedge \left(\forall y^{(4)} \forall z^{(3)} \forall z^{(2)}. (\theta \vee (\forall t^{(3)} \forall t^{(2)} (z^{(2)} \neq t^{(2)}). \theta')) \right) \right) \end{aligned}$$

where the EPIL formulas ξ , θ , and θ' are given respectively, by:

$$\xi = \#(p_n(y^{(3)}) \wedge p_m(x^{(4)})) * \#(p_q(y^{(3)}) \wedge p_a(x^{(4)}) \wedge p_g(y^{(2)})) * \#(p_c(y^{(3)}) \wedge p_d(x^{(4)}) \wedge p_s(y^{(2)})),$$

$$\theta = \neg(\text{true} * \#(p_q(z^{(3)}) \wedge p_a(y^{(4)}) \wedge p_g(z^{(2)})) * \text{true}),$$

and

$$\begin{aligned} \theta' = & (\text{true} * \#(p_q(z^{(3)}) \wedge p_a(y^{(4)}) \wedge p_g(z^{(2)})) * \text{true}) \wedge \\ & \neg(\text{true} * \#(p_q(t^{(3)}) \wedge p_a(y^{(4)}) \wedge p_g(t^{(2)})) * \text{true}). \end{aligned}$$

The subformula $\forall y^{(4)} \forall z^{(3)} \forall z^{(2)}. (\theta \vee (\forall t^{(3)} \forall t^{(2)} (z^{(2)} \neq t^{(2)}). \theta'))$ in ψ serves as a constraint to ensure that a unique coordinator is assigned to each service.

Example 6 (Publish/Subscribe). We consider Publish/Subscribe architecture, described in Example 3, in the parametric setting. In the subsequent FOEIL sentence ψ , we let variable sets $\mathcal{X}^{(1)}$, $\mathcal{X}^{(2)}$, $\mathcal{X}^{(3)}$ correspond to publisher, topic, and subscriber component instances, respectively.

$$\begin{aligned} \psi = & \exists^{\sqcup} x^{(2)}. \left(\left(\exists^{\sqcup} x^{(1)}. (\#(p_a(x^{(1)}) \wedge p_n(x^{(2)})) * \#(p_t(x^{(1)}) \wedge p_r(x^{(2)}))) \right) * \right. \\ & \left. \left(\exists^{\sqcup} x^{(3)}. (\#(p_e(x^{(3)}) \wedge p_c(x^{(2)})) * \#(p_g(x^{(3)}) \wedge p_s(x^{(2)})) * \#(p_d(x^{(3)}) \wedge p_f(x^{(2)}))) \right) \right). \end{aligned}$$

In [28] a simpler version of Request/Response and Blackboard architectures is described where the resulting sets of interactions do not depict any order. Publish/Subscribe architecture has not been studied in [10, 26, 28].

Observe that in the presented examples, whenever is defined a unique instance for a component type we may also consider the corresponding set of variables as a singleton.

6 Decidability Results for FOEIL

In this section, we prove that the equivalence and validity problems for FOEIL sentences are decidable in doubly exponential time, whereas the satisfiability problem is decidable in exponential time. For this, we establish an effective translation of every FOEIL formula to an expressive equivalent finite automaton, and hence we take advantage of well-known computational results for finite automata. We refer the reader to [32] for detailed proofs of our results.

Theorem 1. *Let ψ be a FOEIL sentence over a set $p\mathcal{B} = \{B(i, j) \mid i \in [n], j \geq 1\}$ of parametric components and $r : [n] \rightarrow \mathbb{N}$. Then, we can effectively construct a finite automaton $\mathcal{A}_{\psi, r}$ over $I_{p\mathcal{B}(r)}$ such that $(r, w) \models \psi$ iff $w \in L(\mathcal{A}_{\psi, r})$ for every $w \in I_{p\mathcal{B}(r)}^*$. The worst case run time for the translation algorithm is exponential and the best case is polynomial.*

We prove Theorem 1 using the subsequent proposition. Let $\mathcal{V} \subseteq \mathcal{X}$ be a finite set of variables. For every $i \in [n]$ and $x^{(i)} \in \mathcal{V}$, we define the set $P(i)(x^{(i)}) = \{p(x^{(i)}) \mid p \in P(i) \text{ and } x^{(i)} \in \mathcal{V}\}$ and let $I_{p\mathcal{B}(\mathcal{V})} = \{a \in I(P_{p\mathcal{B}(\mathcal{V})}) \mid |a \cap P(i)(x^{(i)})| \leq 1 \text{ for every } i \in [n] \text{ and } x^{(i)} \in \mathcal{V}\}$. Next let σ be a (\mathcal{V}, r) -assignment and L a language over $I_{p\mathcal{B}(\mathcal{V})}$. We denote by $\sigma(L)$ the language over $I(P_{p\mathcal{B}(r)})$ ² which is obtained by L by replacing every variable $x \in \mathcal{V}$ by $\sigma(x)$.

Proposition 1. *Let ψ be a FOEIL formula over a set $p\mathcal{B} = \{B(i, j) \mid i \in [n], j \geq 1\}$ of parametric components. Let also $\mathcal{V} \subseteq \mathcal{X}$ be a finite set of variables containing $\text{free}(\psi)$ and $r : [n] \rightarrow \mathbb{N}$. Then, we can effectively construct a finite automaton $\mathcal{A}_{\psi, r}$ over $I_{p\mathcal{B}(\mathcal{V})}$ such that for every (\mathcal{V}, r) -assignment σ and $w \in I_{p\mathcal{B}(r)}^*$ we have $(r, \sigma, w) \models \psi$ iff $w \in \sigma(L(\mathcal{A}_{\psi, r})) \cap I_{p\mathcal{B}(r)}^*$. The worst case run time for the translation algorithm is exponential and the best case is polynomial.*

Proof. We prove our claim by induction on the structure of the FOEIL formula ψ . The input of the translation algorithm is the FOEIL formula ψ and the complexity measure refers to the set of states of the derived finite automaton $\mathcal{A}_{\psi, r}$.

Proof (of Theorem 1). We apply Proposition 1. Since ψ is a sentence it contains no free variables. Hence, we get a finite automaton $\mathcal{A}_{\psi, r}$ over $I_{p\mathcal{B}(r)}$ such that $(r, w) \models \psi$ iff $w \in L(\mathcal{A}_{\psi, r})$ for every $w \in I_{p\mathcal{B}(r)}^*$, and this concludes our proof.

Theorem 2. *Let $p\mathcal{B} = \{B(i, j) \mid i \in [n], j \geq 1\}$ be a set of parametric components and $r : [n] \rightarrow \mathbb{N}$ a mapping. Then, the equivalence problem for FOEIL sentences over $p\mathcal{B}$ w.r.t. r is decidable in doubly exponential time.*

Next, we deal with the decidability of satisfiability and validity results for FOEIL sentences. For this, we recall firstly these notions. A FOEIL sentence ψ over $p\mathcal{B}$ is called *satisfiable w.r.t. r* whenever there exists a $w \in I_{p\mathcal{B}(r)}^*$ such that $(r, w) \models \psi$, and *valid w.r.t. r* whenever $(r, w) \models \psi$ for every $w \in I_{p\mathcal{B}(r)}^*$.

Theorem 3. *Let $p\mathcal{B} = \{B(i, j) \mid i \in [n], j \geq 1\}$ be a set of parametric components and $r : [n] \rightarrow \mathbb{N}$ a mapping. Then, the satisfiability problem for FOEIL sentences over $p\mathcal{B}$ w.r.t. r is decidable in exponential time.*

² $\sigma(L)$ is not always over $I_{p\mathcal{B}(r)}$. For instance, assume that $a \in L$ with $a \in I_{p\mathcal{B}(\mathcal{V})}$, $p(x^{(i)}), p'(y^{(i)}) \in a$ for some $i \in [n]$, $p, p' \in P(i)$, and $\sigma(x^{(i)}) = \sigma(y^{(i)})$. Then $\sigma(a) \notin I_{p\mathcal{B}(r)}$.

Proof. Let ψ be a FOEIL sentence over $p\mathcal{B}$. By Theorem 1 we construct, in exponential time, a finite automaton $\mathcal{A}_{\psi,r}$ such that $(r, w) \models \psi$ iff $w \in L(\mathcal{A}_{\psi,r})$ for every $w \in I_{p\mathcal{B}(r)}^*$. Then, ψ is satisfiable iff $L(\mathcal{A}_{\psi,r}) \neq \emptyset$ which is decidable in linear time [33], and thus satisfiability of FOEIL sentences over $p\mathcal{B}$ is decidable in exponential time.

Theorem 4. *Let $p\mathcal{B} = \{B(i, j) \mid i \in [n], j \geq 1\}$ be a set of parametric components and $r : [n] \rightarrow \mathbb{N}$ a mapping. Then, the validity problem for FOEIL sentences over $p\mathcal{B}$ w.r.t. r is decidable in doubly exponential time.*

Proof. Let ψ be a FOEIL sentence over $p\mathcal{B}$. By Theorem 1 we construct, in exponential time, a finite automaton $\mathcal{A}_{\psi,r}$ such that $(r, w) \models \psi$ iff $w \in L(\mathcal{A}_{\psi,r})$ for every $w \in I_{p\mathcal{B}(r)}^*$. Then, ψ is valid iff $L(\mathcal{A}_{\psi,r}) = I_{p\mathcal{B}(r)}^*$ which is decidable in exponential time [33]. Hence, we can decide whether ψ is valid or not in doubly exponential time.

7 Conclusion

In this paper we deal with the formal study of architectures for parametric component-based systems. We introduce a propositional logic, EPIL, which augments PIL from [28] with a concatenation and a shuffle operator, and interpret EPIL formulas over finite words of interactions. We also study FOEIL, the first-order level of EPIL, as a modelling language for the architectures of parametric systems. EPIL and FOEIL encode the permissible interactions and the order restrictions of complex architectures. Several examples are presented and we show the decidability of equivalence, satisfiability and validity of FOEIL sentences.

Ongoing work involves the verification of parametric systems against formal properties, and specifically the application of architectures modelled by FOEIL, for studying the behavior and proving properties (such as deadlock-freedom) of parametric systems. Several architectures, like Ring and Linear [28] cannot be formalized by FOEIL sentences. For this, the study of second-order level of EPIL is needed which is left as future work. Another direction is the extension of our framework for modelling architectures with data applied on parametric systems. Also, it would be interesting to investigate in our setting the architecture composition problem (cf. [5]). Finally, in a forthcoming paper we study parametric component-based systems and FOEIL in the weighted setup.

Acknowledgement. We are deeply grateful to Simon Blidze for discussions on a previous version of the paper.

References

1. Abdulla, P.A., Delzanno, G.: Parameterized verification. *Int. J. Softw. Tools Technol. Transf.* **18**(5), 469–473 (2016). <https://doi.org/10.1007/s10009-016-0424-3>
2. Alagar, V.S., Periyasamy, K.: The B-Method. In: *Specification of Software Systems. Texts in Computer Science*. Springer, London (2011). https://doi.org/10.1007/978-0-85729-277-3_19
3. Amaro, S., Pimentel, E., Roldan, A.M.: REO based interaction model. *Electron. Notes Theor. Comput. Sci.* **160**, 3–14 (2006). <https://doi.org/10.1016/j.entcs.2006.05.012>
4. Aminof, B., Kotek, T., Rubin, S., Spegni, F., Veith, H.: Parameterized model checking of rendezvous systems. *Distrib. Comput.* **31**(3), 187–222 (2017). <https://doi.org/10.1007/s00446-017-0302-6>
5. Attie, P., Baranov, E., Bliudze, S., Jaber, M., Sifakis, J.: A general framework for architecture composability. *Formal Aspects Comput.* **28**(2), 207–231 (2016). <https://doi.org/10.1007/s00165-015-0349-8>
6. Barr, A., Cohen, P., Feigebaum, E.A. (eds.): *Handbook of Artificial Intelligence*. Addison-Wesley, Boston (1989)
7. Bliudze, S., Henrio, L., Madelaine, E.: Verification of concurrent design patterns with data. In: Riis Nielson, H., Tuosto, E. (eds.) *COORDINATION 2019. LNCS*, vol. 11533, pp. 161–181. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-22397-7_10
8. Bliudze, S., Sifakis, J.: The algebra of connectors - structuring interaction in BIP. *IEEE Trans. Comput.* **57**(10), 1315–1330 (2008). <https://doi.org/10.1109/TC.2008.2>
9. Bloem, R., et al.: Decidability in parameterized verification. *SIGACT News* **47**(2), 53–64 (2016). <https://doi.org/10.2200/S00658ED1V01Y201508DCT013>
10. Bozga, M., Iosif, R., Sifakis, J.: Checking deadlock-freedom of parametric component-based systems. In: Vojnar, T., Zhang, L. (eds.) *TACAS 2019. LNCS*, vol. 11428, pp. 3–20. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17465-1_1
11. Bozga, M., Iosif, R., Sifakis, J.: Structural invariants for parametric verification of systems with almost linear architectures (2019). <https://arxiv.org/pdf/1902.02696.pdf>
12. Bruni, R., Lluch Lafuente, A., Montanari, U., Tuosto, E.: Service oriented architectural design. In: Barthe, G., Fournet, C. (eds.) *TGC 2007. LNCS*, vol. 4912, pp. 186–203. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78663-4_14
13. Charalambides, M., Dinges, P., Agha, G.: Parameterized, concurrent session types for asynchronous multi-actor interactions. *Sci. Comput. Program.* **115–116**, 100–126 (2016). <https://doi.org/10.1016/j.scico.2015.10.006>
14. Corkill, D.D.: Blackboard systems. *AI Expert* **6**(9), 40–47 (1991)
15. Daigneau, R. (ed.): *Service Design Patterns: Fundamental Design Solutions for SOAP/WSDL and RESTful web services*. Addison-Wesley, Boston (2012). <https://doi.org/10.1145/2237796.2237821>
16. Deniélou, P.-M., Yoshida, N.: Multiparty session types meet communicating automata. In: Seidl, H. (ed.) *ESOP 2012. LNCS*, vol. 7211, pp. 194–213. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28869-2_10
17. Deniélou, P.-M., Yoshida, N., Bejleri, A., Hu, R.: Parameterised multiparty session types. *Log. Methods Comput. Sci.* **8**(4:6), 1–46 (2012). [https://doi.org/10.2168/LMCS-8\(4:6\)2012](https://doi.org/10.2168/LMCS-8(4:6)2012)

18. Eugster, P., Felber, P., Guerraoui, R., Kermarrec, M.A.: The many faces of Publish/Subscribe. *ACM Comput. Surv.* **35**(2), 114–131 (2003). <https://doi.org/10.1145/857076.857078>
19. Francalanza, A., Aceto, L., Ingolfssdottir, A.: Monitorability for the Hennessy–Milner logic with recursion. *Formal Methods Syst. Des.* **51**(1), 87–116 (2017). <https://doi.org/10.1007/s10703-017-0273-z>
20. Giusto Di, C., Stefani, B.J.: Revising glue expressiveness in component-based systems. In: Meuter, W.D., Ronan, G.C. (eds.) *COORDINATION 2011. LNCS*, vol. 6721, pp. 16–30 (2011). https://doi.org/10.1007/978-3-642-21464-6_2
21. Guanciale, R., Tuosto, E.: Realisability of pomsets. *J. Log. Algebr. Methods Program.* **108**, 69–89 (2019). <https://doi.org/10.1016/j.jlamp.2019.06.003>
22. He, N., et al.: Component-based design and verification in X-MAN. In: *ERTS²* (2012). <https://web1.see.asso.fr/erts2012/Site/0P2RUC89/1D-2.pdf>
23. Hennessy, M., Milner, R.: Algebraic laws for nondeterminism and concurrency. *J. ACM* **32**(1), 137–161 (1985). <https://doi.org/10.1145/2455.2460>
24. Honda, K., Yoshida, N., Carbone, M.: Multiparty asynchronous session types. *J. ACM* **63**(1), 9:1–9:67 (2016). <https://doi.org/10.1145/2827695>
25. Hüttel, H., et al.: Foundations of session types and behavioural contracts. *ACM Comput. Surv.* **49**(1), 3:1–3:36 (2016). <https://doi.org/10.1145/2873052>
26. Konnov, I., Kotek, T., Wang, Q., Veith, H., Bliudze, S., Sifakis, J.: Parameterized systems in BIP: design and model checking. In: Desharnais, J., Jagadeesan, R. (eds.) *CONCUR 2016. LIPIcs*, vol. 59, pp. 30:1–30:16. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2016). <https://doi.org/10.4230/LIPIcs.CONCUR.2016.30>
27. Malavolta, I., Lewis, G., Schmerl, B., Lago, P., Garlan, D.: How do you architect your robots? State of the practice and guidelines for ROS-based systems. In: *ICSE-CEIP 2020. ACM* (2020). <https://doi.org/10.1145/3377813.3381358>
28. Mavridou, A., Baranov, E., Bliudze, S., Sifakis, J.: Configuration logics: modelling architecture styles. *J. Log. Algebr. Methods Program.* **86**, 2–29 (2016). <https://doi.org/10.1016/j.jlamp.2016.05.002>
29. Nii, H.: Blackboard Systems, chap. in [6]
30. Olivieri, A., Rizzo, G., Morand, F.: A publish-subscribe approach to IoT integration: the smart office use case. In: Baroli, L., Takizawa, M., Xhafa, F., Enokido, T., Park, J. (eds.) *29th International Conference on Advanced Information Networking and Applications Workshops*, pp. 644–651. IEEE (2015). <https://doi.org/10.1109/WAINA.2015.28>
31. Patel, S., Jardosh, S., Makwana, A., Thakkar, A.: Publish/Subscribe mechanism for IoT: a survey of event matching algorithms and open research challenges. In: Modi, N., Verma, P., Trivedi, B. (eds.) *Proceedings of International Conference on Communication and Networks. AISC*, vol. 508, pp. 287–294. Springer, Singapore (2017). https://doi.org/10.1007/978-981-10-2750-5_30
32. Pittou, M., Rahonis, G.: Architecture modelling of parametric component-based systems (2020). <http://arxiv.org/abs/1904.02222>
33. Sakarovitch, J.: *Elements of Automata Theory*. Cambridge University Press, Cambridge (2009)
34. Sharma, A., Kumar, M., Agarwal, S.: A complete survey on software architectural styles and patterns. *Procedia Comput. Sci.* **70**, 16–28 (2015). <https://doi.org/10.1016/j.procs.2015.10.019>
35. Straub, J., Reza, H.: The use of the blackboard architecture for a decision making system for the control of craft with various actuator and movement capabilities. In: Latifi, S. (ed.) *ITNG 2014*. pp. 514–519. IEEE (2014)

36. Tuosto, E., Guanciale, R.: Semantics of global view of choreographies. *J. Log. Algebr. Methods Program.* **95**, 17–40 (2018). <https://doi.org/10.1016/j.jlamp.2017.11.002>
37. Yang, K., Zhang, K., Jia, X., Hasan, M.A., Shen, X.: Privacy-preserving attribute-keyword based data publish-subscribe service on cloud platforms. *Inform. Sci.* **387**, 116–131 (2017). <https://doi.org/10.1016/j.ins.2016.09.020>
38. Zhang, K., Muthusamy, V., Jacobsen, A., H.: Total order in content-based Publish/Subscribe systems. In: 2012 32nd IEEE International Conference on Distributed Computing Systems, pp. 335–344. IEEE (2012). <https://doi.org/10.1109/ICDCS.2012.17>