Architectural Types Revisited: Extensible And/Or Connections

Marco Bernardo and Francesco Franzè

Università di Urbino - Italy
Centro per l'Applicazione delle Scienze e Tecnologie dell'Informazione

Abstract. The problem of formalizing architectural styles has been recently tackled with the introduction of the concept of architectural type. The internal behavior of the system components can vary from instance to instance of an architectural type in a controlled way, which preserves the absence of deadlock related architectural mismatches proved via the architectural compatibility and interoperability checks. In this paper we extend the notion of architectural type by permitting a controlled variability of the component topology as well. This is achieved by declaring some component connections to be extensible, in the sense that the number of connected components can vary from instance to instance of an architectural type. We show that such a controlled variability of the topology is still manageable from the analysis viewpoint, as the architectural compatibility and interoperability checks scale with respect to the number of components attached to the extensible connections.

1 Introduction

An important goal of the software architecture discipline [10, 11] is the creation of an established and shared understanding of the common forms of software design. Starting from the user requirements, the designer should be able to identify a suitable organizational style, in order to capitalize on codified principles and experience to specify, analyze, plan, and monitor the construction of a software system with high levels of efficiency and confidence. An architectural style defines a family of software systems having a common vocabulary of components as well as a common topology and set of contraints on the interactions among the components. Since an architectural style encompasses an entire family of software systems, it is desirable to formalize the concept of architectural style both to have a precise definition of the system family and to study the architectural properties common to all the systems of the family. This is not a trivial task because there are at least two degrees of freedom: variability of the component topology and variability of the component internal behavior.

Some papers have appeared in the literature that address the formalization of the architectural styles. In [1] a formal framework based on Z has been provided for precisely defining architectural styles and analyzing within and between different architectural styles. This is accomplished by means of a small set of mappings from the syntactic domain of architectural descriptions to the

semantic domain of architectural meaning, following the standard denotational approach developed for programming languages. In [7] a syntactic theory of software architecture has been presented that is based on set theory, regular expressions, and context free grammars. Architectural styles have been categorized through the typing of the nodes and the connections in the diagrammatic syntax as well as a pattern matching mechanism. In [9] architectural styles have been represented as logical theories and a method has been introduced for the stepwise refinement of an abstract architecture into a relatively correct lower level one. In [4] a process algebraic approach is adopted. In such an approach the description of an architectural style via WRIGHT [3] comprises the definition of component and connector types with a fixed internal behavior as well as topological constraints, whereas the component and connector instances and the related attachments are separately specified in the configurations of the style, so that the set of component and connector instances and the related attachments can vary from configuration to configuration. Also in [5] a process algebraic approach is adopted. An intermediate abstraction called architectural type is introduced, which denotes a set of software architectures with the same topology that differ only for the internal behavior of their architectural elements and satisfy the same architectural compatibility and interoperability properties [6].

The purpose of this paper is to encompass the two complementary, process algebra based approaches of [5] and [3]. This is accomplished by enriching the expressivity of an architectural type by adding the capability of modeling extensible and/or connections, i.e. connections to which a variable number of coordinated/independent architectural elements can be attached. From the analysis viewpoint, the main contribution of the paper is to show that, under certain constraints, the architectural compatibility and interoperability checks of [6] are still effective as they scale w.r.t. the number of software components attached to the extensible and/or connections.

This paper is organized as follows. In Sect. 2 we recall syntax, semantics, and architectural checks for PADL, a process algebra based ADL for the description of architectural types. In Sect. 4 we enrich PADL with extensible and/or connections and we prove the scalability of the architectural checks. Finally, in Sect. 4 we discuss some future work.

2 PADL: A Process Algebra Based ADL

In this section we recall the syntax, the semantics, and the architectural checks for PADL, a process algebra based ADL for the compositional, graphical, and hierarchical modeling of architectural types. For a complete presentation and comparisons with related work, the reader is referred to [5, 6].

The set of process terms of the process algebra PA on which PADL is based is generated by the following syntax

$$E ::= \underline{0} \mid a.E \mid E/L \mid E[\varphi] \mid E + E \mid E \parallel_S E \mid A$$

where a belongs to a set Act of actions including a distinguished action τ for unobservable activities, $L, S \subseteq Act - \{\tau\}$, φ belongs to a set $ARFun$ of action

relabeling functions preserving observability (i.e., $\varphi^{-1}(\tau) = \{\tau\}$), and A belongs to a set *Const* of constants each possessing a (possibly recursive) defining equation of the form $A \triangleq E$. In the syntax above, "$\underline{0}$" is the term that cannot execute any action. Term $a.E$ can execute action a and then behaves as term E. Term E/L behaves as term E with each executed action a turned into τ whenever $a \in L$. Term $E[\varphi]$ behaves as term E with each executed action a turned into $\varphi(a)$. Term $E_1 + E_2$ behaves as either term E_1 or term E_2 depending on whether an action of E_1 or an action of E_2 is executed. Term $E_1 \parallel_S E_2$ asynchronously executes actions of E_1 or E_2 not belonging to S and synchronously executes equal actions of E_1 and E_2 belonging to S. The action prefix operator and the alternative composition operator are called dynamic operators, whereas the hiding operator, the relabeling operator, and the parallel composition operator are called static operators. A term is called sequential if it is composed of dynamic operators only. The notion of equivalence that we consider for PA is the weak bisimulation equivalence [8], denoted $\approx_{\mathrm{B}}$, which captures the ability of two terms to simulate each other behaviors up to τ actions.

A description in PADL represents an architectural type (AT). Each AT is defined as a function of its architectural element types (AETs) and its architectural topology. An AET is defined as a function of its behavior, specified either as a family of PA sequential terms or through an invocation of a previously defined AT, and its interactions, specified as a set of PA actions. The architectural topology is specified through the declaration of a fixed set of architectural element instances (AEIs), a fixed set of architectural interactions (AIs) for the whole AT when viewed as a single component, and a fixed set of directed architectural attachments (DAAs) among the AEIs. We show in Table 1 a PADL textual description for a client-server system. The same system is depicted in Fig. 1 through the PADL graphical notation, which is based on flow graphs [8].

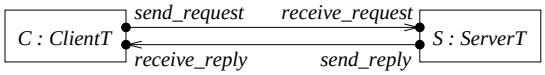

Fig. 1. Flow graph of *ClientServer*

The semantics of a PADL specification is given by translation into PA in two steps. In the first step, the semantics of all the instances of each AET is defined to be the behavior of the AET projected onto its interactions.

Definition 1. *Given a PADL specification, let $\mathcal{C}$ be an AET with behavior E and interaction set $\mathcal{I}$. The semantics of $\mathcal{C}$ and its instances is defined by*

$$[\![\mathcal{C}]\!] = E/(Act - \{\tau\} - \mathcal{I})$$ ■

In our client-server example we have $[\![ClientT]\!] = [\![C]\!] = Client$ and $[\![ServerT]\!] = [\![S]\!] = Server/\{process\_request\}$.

archi_type	*ClientServer*
archi_elem_types	
elem_type	*ClientT*
behavior	$Client \triangleq send\_request.receive\_reply.Client$
interactions	**output** *send_request*
	input *receive_reply*
elem_type	*ServerT*
behavior	$Server \triangleq receive\_request.process\_request.send\_reply.Server$
interactions	**input** *receive_request*
	output *send_reply*
archi_topology	
archi_elem_instances	C : *ClientT*
	S : *ServerT*
archi_interactions	
archi_attachments	**from** $C.send\_request$ **to** $S.receive\_request$
	from $S.send\_reply$ **to** $C.receive\_reply$
end	

Table 1. Textual description of *ClientServer*

In the second step, the semantics of an AT is obtained by composing in parallel the semantics of its AEIs according to the specified DAAs. In our client-server example we have $[\![ClientServer]\!] = [\![C]\!][send\_request \mapsto a, receive\_reply \mapsto b] \parallel_{\{a,b\}} [\![S]\!][receive\_request \mapsto a, send\_reply \mapsto b]$, where the use of the relabeling operator is necessary to make the AEIs interact. In general, let $C_1, \ldots, C_n$ be AEIs of an AT, with interaction sets $\mathcal{I}_{C_1}, \ldots, \mathcal{I}_{C_n}$ containing the AI sets $\mathcal{AI}_{C_1}, \ldots, \mathcal{AI}_{C_n}$, respectively. Let i, j, k range over $\{1, \ldots, n\}$. We say that $C_i.a_1$ is connected to $C_j.a_2$ iff either there is a DAA between them, or there exists an interaction a_3 of C_k such that $C_i.a_1$ is connected to $C_k.a_3$ and there is a DAA between $C_k.a_3$ and $C_j.a_2$. We say that a subset of interactions of $C_1, \ldots, C_n$ is connected iff they are pairwise connected via DAAs involving interactions of $C_1, \ldots, C_n$ only and the subset is maximal. Since the actions of a connected subset of interactions must be identically relabeled in order to result in a synchronization at the semantic level, denoted by $\mathcal{I}_{C_i;C_1,\ldots,C_n} \subseteq \mathcal{I}_{C_i}$ the subset of interactions of C_i attached to $C_1, \ldots, C_n$, let $\mathcal{S}(C_1, \ldots, C_n)$ be a set of as many fresh actions as there are connected subsets of interactions among the considered AEIs, let $\varphi_{C_i;C_1,\ldots,C_n} : \mathcal{I}_{C_i;C_1,\ldots,C_n} \longrightarrow \mathcal{S}(C_1, \ldots, C_n)$ be injective relabeling functions such that $\varphi_{C_i;C_1,\ldots,C_n}(a_1) = \varphi_{C_j;C_1,\ldots,C_n}(a_2)$ iff $C_i.a_1$ is connected to $C_j.a_2$, and let $\mathcal{S}(C_i; C_1, \ldots, C_n) = \varphi_{C_i;C_1,\ldots,C_n}(\mathcal{I}_{C_i;C_1,\ldots,C_n})$ and $\mathcal{S}(C_i, C_j; C_1, \ldots, C_n) = \mathcal{S}(C_i; C_1, \ldots, C_n) \cap \mathcal{S}(C_j; C_1, \ldots, C_n)$.

Definition 2. *Let $C_1, \ldots, C_n$ be AEIs of an AT. The closed and the open interacting semantics of C_i restricted to $C_1, \ldots, C_n$ are defined by*

$$[\![C_i]\!]^{\mathrm{c}}_{C_1,\ldots,C_n} = [\![C_i]\!] \,/\, (Act - \{\tau\} - \mathcal{I}_{C_i;C_1,\ldots,C_n}) \quad [\varphi_{C_i;C_1,\ldots,C_n}]$$
$$[\![C_i]\!]^{\mathrm{o}}_{C_1,\ldots,C_n} = [\![C_i]\!] \,/\, (Act - \{\tau\} - (\mathcal{I}_{C_i;C_1,\ldots,C_n} \cup \mathcal{AI}_{C_i})) \, [\varphi_{C_i;C_1,\ldots,C_n}] \quad \blacksquare$$

Definition 3. *Let $C_1, \ldots, C_n$ be AEIs of an AT. The closed and the open interacting semantics of the set of AEIs are defined by*

$$\begin{aligned}[\![C_1, \ldots, C_n]\!]^{\mathrm{c}} = {} & [\![C_1]\!]^{\mathrm{c}}_{C_1,\ldots,C_n} \,\|_{\mathcal{S}(C_1,C_2;C_1,\ldots,C_n)} \\ & [\![C_2]\!]^{\mathrm{c}}_{C_1,\ldots,C_n} \,\|_{\mathcal{S}(C_1,C_3;C_1,\ldots,C_n)\cup\mathcal{S}(C_2,C_3;C_1,\ldots,C_n)} \cdots \\ & \cdots \|_{\cup_{i=1}^{n-1}\mathcal{S}(C_i,C_n;C_1,\ldots,C_n)} [\![C_n]\!]^{\mathrm{c}}_{C_1,\ldots,C_n} \\ [\![C_1, \ldots, C_n]\!]^{\mathrm{o}} = {} & [\![C_1]\!]^{\mathrm{o}}_{C_1,\ldots,C_n} \,\|_{\mathcal{S}(C_1,C_2;C_1,\ldots,C_n)} \\ & [\![C_2]\!]^{\mathrm{o}}_{C_1,\ldots,C_n} \,\|_{\mathcal{S}(C_1,C_3;C_1,\ldots,C_n)\cup\mathcal{S}(C_2,C_3;C_1,\ldots,C_n)} \cdots \\ & \cdots \|_{\cup_{i=1}^{n-1}\mathcal{S}(C_i,C_n;C_1,\ldots,C_n)} [\![C_n]\!]^{\mathrm{o}}_{C_1,\ldots,C_n} \end{aligned} \quad \blacksquare$$

Definition 4. *The semantics of an AT $\mathcal{A}$ with AEIs $C_1, \ldots, C_n$ is defined by*

$$[\![\mathcal{A}]\!] = [\![C_1, \ldots, C_n]\!]^{\mathrm{o}} \quad \blacksquare$$

A PADL description represents a family of software architectures called an AT. An instance of an AT can be obtained by invoking the AT and passing actual behavior preserving AETs and actual names for the AIs, whereas it is not possible to pass an actual topology. This restriction allows us to efficiently check whether an AT invocation conforms to an AT definition.

Definition 5. *Let $\mathcal{A}(\mathcal{C}'_1, \ldots, \mathcal{C}'_m; a'_1, \ldots, a'_l)$ be an invocation of the AT $\mathcal{A}$ defined with formal AETs $\mathcal{C}_1, \ldots, \mathcal{C}_m$ and AIs $a_1, \ldots, a_l$. $\mathcal{C}'_i$ is said to conform to $\mathcal{C}_i$ iff there exist an injective relabeling function φ'_i for the interactions of $\mathcal{C}'_i$ and an injective relabeling function φ_i for the interactions of $\mathcal{C}_i$ such that*

$$[\![\mathcal{C}'_i]\!][\varphi'_i] \approx_{\mathrm{B}} [\![\mathcal{C}_i]\!][\varphi_i] \quad \blacksquare$$

Definition 6. *Let $\mathcal{A}(\mathcal{C}'_1, \ldots, \mathcal{C}'_m; a'_1, \ldots, a'_l)$ be an invocation of the AT $\mathcal{A}$ defined with formal AETs $\mathcal{C}_1, \ldots, \mathcal{C}_m$ and AIs $a_1, \ldots, a_l$. If $\mathcal{C}'_i$ conforms to $\mathcal{C}_i$ for all $i = 1, \ldots, m$, then the semantics of the AT invocation is defined by*

$$[\![\mathcal{A}(\mathcal{C}'_1, \ldots, \mathcal{C}'_m; a'_1, \ldots, a'_l)]\!] = [\![\mathcal{A}]\!][a_1 \mapsto a'_1, \ldots, a_l \mapsto a'_l] \quad \blacksquare$$

Theorem 1. *Let $\mathcal{A}(\mathcal{C}'_1, \ldots, \mathcal{C}'_m; a'_1, \ldots, a'_l)$ be an invocation of the AT $\mathcal{A}$ defined with formal AETs $\mathcal{C}_1, \ldots, \mathcal{C}_m$ and AIs $a_1, \ldots, a_l$ and let $C'_1, \ldots, C'_n$ be the AEIs of the AT invocation. If $\mathcal{C}'_i$ conforms to $\mathcal{C}_i$ for all $i = 1, \ldots, m$, then there exist an injective relabeling function φ' for the interactions of the AT invocation and an injective relabeling function φ for the interactions of the AT definition such that $[\![C'_1, \ldots, C'_n]\!]^{\mathrm{o}}[\varphi'] \approx_{\mathrm{B}} [\![\mathcal{A}]\!][\varphi]$.*

PADL is equipped with two architectural checks to detect system blocks. The first check (compatibility) is concerned with the well formedness of acyclic ATs, while the second check (interoperability) is concerned with the well formedness of sets of AEIs forming a cycle. Both checks are preserved by conformity.

Definition 7. *Given an acyclic AT, let $C_1, \ldots, C_n$ be the AEIs attached to AEI K. C_i is said to be compatible with K iff*

$$[\![K]\!]^{\mathrm{c}}_{K,C_1,\ldots,C_n} \,\|_{\mathcal{S}(K;K,C_1,\ldots,C_n)}\, [\![C_i]\!]^{\mathrm{c}}_{K,C_1,\ldots,C_n} \approx_{\mathrm{B}} [\![K]\!]^{\mathrm{c}}_{K,C_1,\ldots,C_n} \quad \blacksquare$$

Theorem 2. *Given an acyclic AT, let $C_1, \ldots, C_n$ be the AEIs attached to AEI K. If $[\![K]\!]^{\mathrm{c}}_{K,C_1,\ldots,C_n}$ is deadlock free and C_i is compatible with K for all $i = 1, \ldots, n$, then*

$$[\![K; C_1, \ldots, C_n]\!] = [\![K]\!]^{\mathrm{c}}_{K,C_1,\ldots,C_n} \parallel_{\mathcal{S}(K;K,C_1,\ldots,C_n)} [\![C_1]\!]^{\mathrm{c}}_{K,C_1,\ldots,C_n} \parallel_{\mathcal{S}(K;K,C_1,\ldots,C_n)} \cdots \cdots \parallel_{\mathcal{S}(K;K,C_1,\ldots,C_n)} [\![C_n]\!]^{\mathrm{c}}_{K,C_1,\ldots,C_n}$$

is deadlock free. ■

Corollary 1. *Given an acyclic AT, if every restricted closed interacting semantics of each AEI is deadlock free and every AEI is compatible with each AEI attached to it, then the AT is deadlock free.* ■

Definition 8. *Given an AT, let $C_1, \ldots, C_n$ be AEIs forming a cycle. C_i is said to interoperate with $C_1, \ldots, C_{i-1}, C_{i+1}, \ldots, C_n$ iff*

$$[\![C_1, \ldots, C_n]\!]^{\mathrm{c}} / (Act - \{\tau\} - \mathcal{S}(C_i; C_1, \ldots, C_n)) \approx_{\mathrm{B}} [\![C_i]\!]^{\mathrm{c}}_{C_1,\ldots,C_n}$$ ■

Theorem 3. *Given an AT, let $C_1, \ldots, C_n$ be AEIs forming a cycle. If there exists C_i such that $[\![C_i]\!]^{\mathrm{c}}_{C_1,\ldots,C_n}$ is deadlock free and C_i interoperates with $C_1, \ldots, C_{i-1}, C_{i+1}, \ldots, C_n$, then $[\![C_1, \ldots, C_n]\!]^{\mathrm{c}}$ is deadlock free.* ■

Theorem 4. *Let $\mathcal{A}(\mathcal{C}'_1, \ldots, \mathcal{C}'_m; a'_1, \ldots, a'_l)$ be an invocation of the AT $\mathcal{A}$ defined with formal AETs $\mathcal{C}_1, \ldots, \mathcal{C}_m$ and AIs $a_1, \ldots, a_l$. If $\mathcal{C}'_i$ conforms to $\mathcal{C}_i$ for all $i = 1, \ldots, m$, then the AT invocation and the AT definition have the same compatibility and interoperability properties.* ■

3 Adding Extensible And/Or Connections to ATs

The instances of an AT can differ for the internal behavior of their AETs. However, it is desirable to have some form of variability in the topology as well. As an example, consider the client-server system of Table 1. Every instance of such an AT can admit a single client and a single server, whereas it would be useful to allow for an arbitrary number of clients (to be instantiated when invoking the AT) that can connect to the server. In this section we enrich the notion of AT by introducing extensible and/or connections and we investigate to which extent they preserve the effectiveness of the architectural checks.

3.1 Syntax and Semantics for Extensible And/Or Connections

From the syntactical viewpoint, the extensible and/or connections are introduced in PADL by further typing the interactions of the AETs. Besides the input/output qualification, the interactions are classified as uniconn, andconn, and orconn, with only the three types of DAA shown in Fig. 2 considered legal. A uniconn interaction is an interaction to which a single AEI can be attached; e.g., all the interactions of *ClientServer* are of this type. An andconn interaction is an interaction to which a variable number of AEIs can be attached, such that all the attached AEIs must synchronize when that interaction takes place; e.g.,

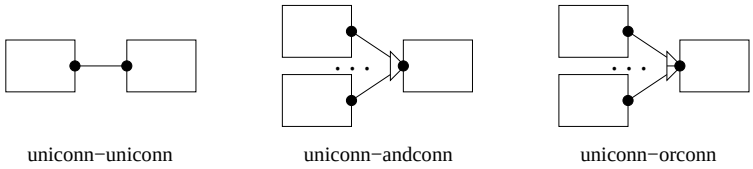

Fig. 2. Legal DAAs in case of extensible and/or connections

a broadcast transmission. An orconn interaction is an interaction to which a variable number of AEIs can be attached, such that only one of the attached AEIs must synchronize when that interaction takes place; e.g., a client-server system with several clients. We observe that, whenever an AEI is attached (with a uniconn interaction) to an andconn/orconn interaction of another AEI, then the former AEI cannot be attached (with uniconn/andconn/orconn interactions) to uniconn interactions of the latter AEI. If this were not the case, in case of extension some interactions of the former AEI should be attached to interactions of the latter AEI that are not extensible (see Fig. 3(a)) or should be made extensible even if they are not (see Fig. 3(b)).

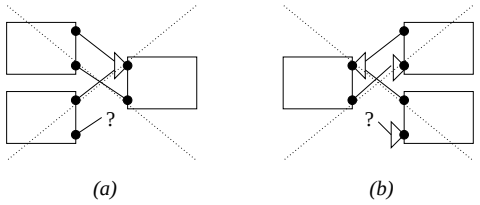

Fig. 3. Forbidden DAAs

As an example, let us define an AT *XClientServer* obtained from *ClientServer* by letting the number of clients connected to the server vary. To achieve that, we redefine the interactions of *ServerT* as follows:

input orconn *receive_request*
output orconn *send_reply* **dep** *receive_request*

and the interactions of *ClientT* as follows:

output uniconn *send_request*
input uniconn *receive_reply*

This means that in an invocation of *XClientServer* arbitrarily many instances of *ClientT* can be declared and attached to the single instance of *ServerT*, with every occurrence of *receive_request* in *Server* replaced by a choice among as many indexed occurrences of *receive_request* as there are instances of *ClientT* attached to the only instance of *ServerT*. In order to make the server send a reply to the client (among the connected ones) that issued the processed request, *send_reply* is declared to depend on *receive_request*. This establishes that every

occurrence of *send_reply* must be given the same index as the indexed occurrence of *receive_request* that precedes it.

In order to make the indexing mechanism work in case of dependencies, we impose the following constraints. In every AET, for each output orconn interaction there must be exactly one input orconn interaction on which it depends, with all pairs of related input output orconn interactions attached to the same AEIs. The behavior of each AET possessing orconn interactions must be i/o alternating: along every maximal simple path of the state transition graph starting from the initial state, it must be the case that every occurrence of an output orconn interaction is preceded by an occurrence of the related input orconn interaction, and that no two occurrences of the same input (output) orconn interaction appear without an intervening occurrence of each of the related output orconn interactions (of the related input orconn interaction). We denote by $\mathcal{G}_{\mathrm{seq,i/o}}$ the set of sequential, i/o alternating terms.

If an AET contains only uniconn/andconn interactions, then its semantics and the semantics of its instances are defined exactly as in Def. 1. The reason is that in the case of a uniconn interaction only one AEI is attached to the interaction, while in the case of an andconn interaction all the AEIs attached to the interaction must synchronize when the interaction takes places. As a consequence, if all the AETs of an AT contain only uniconn/andconn interactions, then the semantics of the AT is defined exactly as in Def. 4.

In the case of an AT with orconn interactions, Def. 1 and Def. 4 still apply provided that they are preceded by a suitable modification of the terms representing the behavior of the AETs containing those interactions and a suitable modification of those interactions themselves, respectively. Such modifications, which are completely transparent to the architect, are necessary to reflect the fact that an orconn interaction expresses a choice among different attached AEIs whenever the interaction takes place. For the sake of simplicity, we formalize below the case in which every input orconn interaction has exactly one output orconn interaction depending on it.

The first modification, which is concerned with the applicability of Def. 1, works as follows on the behavior section of the definition of the AETs. From a conceptual viewpoint, we view every term $E \in \mathcal{G}_{\mathrm{seq,i/o}}$ representing the behavior of an AET with orconn interactions as expressing the behavior of an instance of the AET when cooperating with a selection of the AEIs attached to its orconn interactions. Therefore, in order to take into account all the possible selections, in principle term E must be rewritten into

$$\sum_{i_1=1}^{m_1} \cdots \sum_{i_n=1}^{m_n} E_{i_1,\ldots,i_n}$$

where n is the number of pairs of related input output orconn interactions, m_j is the number of AEIs attached to pair $j = 1, \ldots, n$, and $E_{i_1,\ldots,i_n}$ is the term obtained from E by attaching index $i_j = 1, \ldots, m_j$ to all the occurrences of the interactions of pair $j = 1, \ldots, n$. In practice, this is realized in a different way, because the choice of the right summand cannot be made at the beginning of the computation but only during the computation. More precisely, denoted by (a, b)

the j-th pair of related input output orconn interactions and by F an arbitrary term in $\mathcal{G}_{\text{seq,i/o}}$, we rewrite each subterm $a.F$ of E into

$$\sum_{i_j=1}^{m_j} a_{i_j}.F$$

and each occurrence of b in E into b_k if k if the index of the last encountered occurrence of a. [1]

The second modification, which is concerned with the applicability of Def. 4, works as follows on the interaction section of the definition of the AETs and on the attachment section of the definition of the architectural topology. For all $j = 1, \ldots, n$, the j-th pair (a, b) of related input output orconn interactions is transformed into $2 \cdot m_j$ indexed, uniconn interactions $a_1, \ldots, a_{m_j}, b_1, \ldots, b_{m_j}$ where a_{i_j} and b_{i_j} are attached to the i_j-th AEI originally attached to a and b. Such a transformation is consistent with the rewriting carried out by the first modification and must be applied before computing the connected subsets of interactions.

In order to make Def. 1 applicable to both the case in which there are no orconn interactions and the case in which there are orconn interactions, we assume that in Def. 1 term E is replaced by term $orbeh(E, \emptyset)$. The second argument of function *orbeh* represents a set of pairs concerned with occurrences of input orconn interactions encountered along a computation of E, where the first element of a pair is the name of the input orconn interaction, while the second element is the index associated with the encountered occurrence of the first element.

Definition 9. *Function* $orbeh : \mathcal{G}_{\text{seq,i/o}} \times 2^{Act \times \mathbb{N}} \longrightarrow \mathcal{G}_{\text{seq,i/o}}$ *determines the smallest term satisfying the following equalities:*

$$orbeh(\underline{0}, I) = \underline{0}$$

$$orbeh(a.E, I) = \begin{cases} a.orbeh(E, I) \\ \qquad \textit{if } a \textit{ not orconn interaction} \\ \sum_{i=1}^{n} a_i.orbeh(E, I \cup (a, i)) \\ \qquad \textit{if } a \textit{ input orconn interaction with } n \textit{ attached AEIs} \\ a_{index(b,I)}.orbeh(E, I - \{(b, index(b, I))\}) \\ \qquad \textit{if } a \textit{ output orconn interaction referring to } b \end{cases}$$

$$orbeh(E_1 + E_2, I) = orbeh(E_1, I) + orbeh(E_2, I)$$

$$orbeh(A, I) = A^I \text{ with } A^I \stackrel{\Delta}{=} orbeh(E, I) \qquad \textit{if } A \stackrel{\Delta}{=} E$$

where $index(b, I)$ *is the index with which* b *occurs in* I. ■

Definition 10. *Given a PADL specification, let* $\mathcal{C}$ *be an AET with behavior* E *and interaction set* $\mathcal{I}$*. The semantics of* $\mathcal{C}$ *and its instances is defined by*

$$[\![\mathcal{C}]\!] = orbeh(E, \emptyset)/(Act - \{\tau\} - \mathcal{I})$$ ■

As an example, consider an instance of *XClientServer* with two instances C_1, C_2 of *ClientT* attached to the only instance S of *ServerT*. Then $[\![C_1]\!] = [\![C_2]\!] = orbeh(Client, \emptyset) = Client$ while $[\![S]\!] = orbeh(Server, \emptyset) = Server^{\emptyset}$ where:

[1] This technique is similar in spirit to that of [2] for dealing with dynamic software architectures through a mapping on static descriptions.

$$Server^{\emptyset} \triangleq receive\_request_1.process\_request.send\_reply_1.Server^{\emptyset} + receive\_request_2.process\_request.send\_reply_2.Server^{\emptyset}$$

After transforming the two orconn interactions *receive_request* and *send_reply* into the following four interactions

input uniconn $receive\_request_1, receive\_request_2$
output uniconn $send\_reply_1, send\_reply_2$

and modifying the related attachments as follows

from $C_1.send\_request$ **to** $S.receive\_request_1$
from $C_2.send\_request$ **to** $S.receive\_request_2$
from $S.send\_reply_1$ **to** $C_1.receive\_reply$
from $S.send\_reply_2$ **to** $C_2.receive\_reply$

we have that the semantics for the whole AT is given by

$$Client[send\_request \mapsto a_1, receive\_reply \mapsto b_1] \parallel_{\emptyset} Client[send\_request \mapsto a_2, receive\_reply \mapsto b_2] \parallel_{\{a_1,a_2,b_1,b_2\}} Server^{\emptyset}[receive\_request_1 \mapsto a_1, receive\_request_2 \mapsto a_2, send\_reply_1 \mapsto b_1, send\_reply_2 \mapsto b_2]$$

Theorem 5. *Let $E \in \mathcal{G}_{\mathrm{seq,i/o}}$. Then there exists a relabeling function φ such that $orbeh(E, \emptyset)[\varphi] \approx_{\mathrm{B}} E$.*

Proof. It suffices to take φ such that every action that is not an orconn interaction is mapped to itself, while every indexed occurrence of an action that is an orconn interaction is mapped to the action. For all constant A occurring in E, the weak bisimulation demonstrating the equivalence of $orbeh(E, \emptyset)[\varphi]$ and E will in particular contain a pair (A, A^I) for every I built during the application of orbeh. ■

Corollary 2. *Let $E \in \mathcal{G}_{\mathrm{seq,i/o}}$. (i) If E has no orconn interactions, then $orbeh(E, \emptyset)$ is isomorphic to E. (ii) If E has no orconn interactions, then $orbeh(E, \emptyset)$ does not depend on the number of AEIs attached to the andconn interactions of the AEI whose behavior is E.* ■

3.2 AT Invocations: Topology Conformity

When invoking an AT, we can pass actual behavior preserving AETs and actual names for the AIs. Now that we can declare extensible andconn/orconn interactions in the definition of an AT, in case of invocation we can also pass an actual topology. Such an actual topology is expressed in an AT invocation between the two previous arguments by means of four additional arguments that declare the actual AEIs, the actual AIs, the actual DAAs, and the list of interaction extensions, respectively. As an example, we provide below the invocation of the AT *XClientServer* with two instances of *ClientT*:

$XClientServer($*ClientT*, *ServerT*;
C_1 : *ClientT*, S : *ServerT*;
;
from C_1.*send_request* **to** S.*receive_request*,
from S.*send_reply* **to** C_1.*receive_reply*;
extend(S.*receive_request*, S.*send_reply*;
C_2 : *ClientT*;
;
from C_2.*send_request* **to** S.*receive_request*,
from S.*send_reply* **to** C_2.*receive_reply*;
);
)

Every interaction extension starts with the keyword extend and comprises six arguments: the andconn/orconn interactions that are extended, the additional AEIs, the additional AIs, the additional DAAs, and the list of further extensions concerning andconn/orconn interactions of the additional AEIs only.

When invoking an AT, as before we have to check the conformity of the actual AETs to the formal AETs according to Def. 5. Additionally, we now have to check the conformity of the actual topology to the formal topology. Besides verifying that the number and type of actual AEIs, AIs, and DAAs coincide with the number and type of formal AEIs, AIs, and DAAs, respecitvely, the check amounts to investigate whether every interaction extension preserves the formal DAAs and is maximal. This means that the specification of an interaction extension must include not only the additional DAAs between the specified andconn/orconn interactions to be extended and the interactions of additional AEIs according to the formal DAAs, but also the additional DAAs between the interactions of such AEIs and the uniconn interactions of further additional AEIs that preserve the formal DAAs, and so on (additional DAAs between the interactions of such additional AEIs and andconn/orconn interactions cannot be encountered if the set of specified andconn/orconn interactions to be extended is maximal). As an example, if in the definition of *XClientServer* we include a further AET called *BrowserT* an we declare an instance B of *BrowserT* whose only uniconn interaction is attached to a new uniconn interaction of C, then in every invocation of the new version of *XClientServer* there must be as many instances of *BrowserT* as there are instances of *ClientT* in order to conform to the formal topology (see Fig. 4).

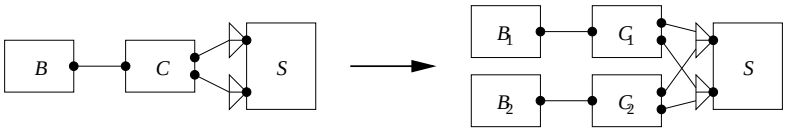

Fig. 4. An extension of the new version of *XClientServer*

3.3 Architectural Check Scalability

In this section we investigate whether the compatibility/interoperability results proved on the instance of an AT with the smallest number of AEIs attached to the andconn/orconn interactions scale to all of its extensions. Here we assume that an AEI attached to an orconn interaction of another AEI cannot be attached to other andconn and unrelated orconn interactions of that AEI.

Definition 11. *Given an andconn/orconn interaction to which instances of certain AETs must be attached, we say that the interaction is minimally (subminimally) attached if one instance (two instances) of each of the AETs above is (are) attached to the interaction. We say that an AT is minimal (subminimal) if each of its andconn/orconn interactions is minimally (subminimally) attached. We say that a (sub)minimal AT is acyclic if so is each of its extensions.* ■

In the case of the architectural compatibility check, which is concerned with acyclic ATs, we always get the desired scalability from the minimal AT to all of its extensions, i.e. when attaching additional AEIs of certain types to the andconn/orconn interactions without introducing cycles.

Theorem 6. *Given an acyclic minimal AT, let $C_1, \ldots, C_n$ be the AEIs attached to AEI K. If $[\![K]\!]^{\mathrm{c}}_{K,C_1,\ldots,C_n}$ is deadlock free and C_i is compatible with K for all $i = 1, \ldots, n$, then $[\![K; C_1, \ldots, C_n]\!]$ and all of its extensions are deadlock free.*

Proof. The first part of the result stems directly from Thm. 2. The second part of the result is trivial if $[\![K; C_1, \ldots, C_n]\!]$ cannot be extended, i.e. K has no andconn/orconn interactions to which additional AEIs of the same type as $C_1, \ldots, C_n$ can be attached, and none among $C_1, \ldots, C_n$ has andconn/orconn interactions. We now examine the nontrivial case concerned with the second part of the result.

Suppose that K has only andconn interactions and that we attach additional AEIs to them. Let us denote by $beh(K)$ the term representing the behavior of K. Since $orbeh(beh(K), \emptyset) = beh(K)$ by virtue of Cor. 2(ii) and each additional AEI has the same type as one among $C_1, \ldots, C_n$, each additional AEI passes the compatibility check. Thus the second part of the result follows.

Suppose now that K has orconn interactions and that we attach additional AEIs to them. In this case $orbeh(beh(K), \emptyset) \neq beh(K)$ because of the choices that function orbeh introduces. Let C be an arbitrary AEI attached to K in the considered extension of $[\![K; C_1, \ldots, C_n]\!]$ and let us investigate the compatibility of C with the extension of K. Since C has the same type as one (say C') among $C_1, \ldots, C_n$, we have that C is compatible with K. Now, if we compute the state transition graph of the parallel composition of the extension of K and C, we observe differences w.r.t. the state transition graph of the parallel composition of K and C' only when we encounter an occurrence of an input orconn interaction of K to which C' is attached (if C' is not attached to it, then from the point of view of C the choice of the summand of the extension of K is irrelevant as the summands differ only for the index given to the occurrence of the input orconn interaction and the occurrences of its related output orconn interactions). At that

point, there are two options. The first option is that C synchronizes with the extension of K on the properly indexed occurrence of the input orconn interaction similarly to what C' did with K, so from that point to the next occurrence of an input orconn interaction of K to which C' is attached there is compatibility. The second option is that C does not synchronize with the extension of K, hence the extension of K drives the computation of the parallel composition by performing one of the other indexed occurrences of the input orconn interaction. The computation is driven by the extension of K until we encounter an occurrence of an interaction of K to which C' is attached. Due to the assumption made at the beginning of this section, we know that this must be an occurrence of the same input orconn interaction considered earlier, hence deadlock cannot arise while the computation is driven by the extension of K. Thus the second part of the result follows.

Finally, let us consider the case in which at least one (say D) among $C_1, \ldots, C_n$ has andconn/orconn interactions. This can give rise to a replication of the structure under consideration (if D has an andconn/orconn interaction to which K is attached) or to an extension of a structure interacting with the one under consideration via D (if D has no andconn/orconn interaction to which K is attached). In both cases, by applying to D an argument similar to that applied above to K, we can derive the second part of the result. ■

Corollary 3. *Given an acyclic minimal AT, if every restricted closed interacting semantics of each AEI is deadlock free and every AEI is compatible with each AEI attached to it, then the AT and all of its extensions are deadlock free.* ■

We observe that verifying whether a minimal AT is cyclic does not require the construction of all the extensions of the AT. It simply requires to build the corresponding subminimal AT and to check that it is acyclic. The reason is that every cycle in an extension of the subminimal AT that is not in the subminimal AT must be a replica of a cycle in the subminimal AT, possibly sharing some AEIs with the original cycle.

In the case of the architectural interoperability check, which is concerned with cyclic ATs, we obtain the desired scalability (for individual cycles) only from the subminimal (instead of minimal) AT to all of its extensions. The reason is that new cycles can be generated when building the subminimal AT corresponding to a minimal AT, whereas every cycle in an extension of the subminimal AT that is not in the subminimal AT must be a replica of a cycle in the subminimal AT.

Theorem 7. *Given a subminimal AT, let $C_1, \ldots, C_n$ be AEIs forming a cycle. If there exists C_i such that $[\![C_i]\!]^{\mathrm{c}}_{C_1,\ldots,C_n}$ is deadlock free and C_i interoperates with $C_1, \ldots, C_{i-1}, C_{i+1}, \ldots, C_n$, then $[\![C_1, \ldots, C_n]\!]^{\mathrm{c}}$ and all of its extensions are deadlock free.*

Proof. The first part of the result stems directly from Thm. 3. The second part of the result is trivial if $[\![C_1, \ldots, C_n]\!]^{\mathrm{c}}$ cannot be replicated, i.e. none among $C_1, \ldots, C_n$ has andconn/orconn interactions to which additional AEIs among which one of the same type as $C_1, \ldots, C_n$ can be attached, and none among

$C_1, \ldots, C_n$ has andconn/orconn interactions to which additional AEIs having types different from those of $C_1, \ldots, C_n$ can be attached. We now examine the nontrivial case concerned with the second part of the result.

Suppose that C_j has only andconn interactions and that we attach additional AEIs to them, among which one of the same type as $C_1, \ldots, C_{j-1}, C_{j+1}, \ldots, C_n$. Let us denote by $beh(C_j)$ the term representing the behavior of C_j. Since $orbeh(beh(C_j), \emptyset) = beh(C_j)$ by virtue of Cor. 2(ii) and the AEIs in the replica of the cycle have the same type as the corresponding AEIs in the cycle under examination, the replica of C_i interoperates with each AEI in the replica of the cycle. Thus the second part of the result follows.

Suppose now that C_j has orconn interactions and that we attach additional AEIs to them, among which one of the same type as $C_1, \ldots, C_{j-1}, C_{j+1}, \ldots, C_n$. In this case $orbeh(beh(C_j), \emptyset) \neq beh(C_j)$ because of the choices that function orbeh introduces, and a replica of the cycle is generated with the cycle and its replica sharing the extended version of C_j. Let C be an AEI in the cycle or in the cycle replica having the same type as C_i (the AEI passing the interoperability check by the initial hypothesis) and let us investigate the interoperability of C in the cycle or in the cycle replica. If we compute the state transition graph expressing the semantics of the cycle or its replica (depending on whether C is), we observe differences w.r.t. the state transition graph expressing the semantics of the original version of the cycle (in which C_j is not extended) only when we encounter an occurrence of an input orconn interaction of C_j. If C is not attached to such an interaction, then interoperability is preserved because, from the point of view of C, the choice of the summand of the extension of C_j is irrelevant as the summands differ only for the index given to the occurrence of the input orconn interaction and the occurrences of its related output orconn interactions. If instead C is attached to such an interaction, there are two options. The first option is that C synchronizes with the extension of C_j on the properly indexed occurrence of the input orconn interaction. In this case, from that point to the next occurrence of an input orconn interaction of C_j to which C is attached there is interoperability. The second option is that C does not synchronize with the extension of C_j, hence the extension of C_j drives the computation by performing one of the other indexed occurrences of the input orconn interaction. From the point of view of C, the computation is driven by the extension of C_j until we encounter an occurrence of an interaction of C_j to which C is attached. Due to the assumption made at the beginning of this section, we know that this must be an occurrence of the same input orconn interaction considered earlier, hence deadlock cannot arise while the computation is driven by the extension of C_j. Thus the second part of the result follows.

Finally, let us consider the case in which at least one (say D) among $C_1, \ldots, C_n$ has andconn/orconn interactions to which additional AEIs having types different from those of $C_1, \ldots, C_n$ can be attached. This can give rise to an extension of a structure interacting with the cycle under consideration via D. By applying to D an argument similar to that applied above to C_j, we can derive the second part of the result. ■

4 Conclusion

In this paper we have enriched the notion of AT of [5] by introducing the capability of expressing extensible and/or connections between software components, in such a way that the architectural checks of [6] scale w.r.t. the number of software components attached to the extensible and/or connections.

As far as future work is concerned, first of all we would like to investigate whether information can be gained about the interoperability of cycles that are generated when building the subminimal AT corresponding to a minimal AT, starting from the compatibility of the involved AEIs of the minimal AT. Second, we would like to investigate the scalability of the architectural checks when relaxing the assumption at the beginning of Sect. 3.3. Finally, we would like to investigate the scalability of the architectural checks when taking a different view in the definition of function *orbeh*. As said in Sect. 3.1, we have taken the view that the term representing the behavior of an AET with orconn interactions expresses the behavior of an instance of the AET when cooperating with a selection of the AEIs attached to its orconn interactions. In the future, we would like to examine the case in which an instance of the AET can simultaneously cooperate with many or all of the AEIs attached to its orconn interactions. An example of this case would be $ServerT$ of Table 1 if it had a queue where requests coming from different clients could be buffered.

References

1. G.D. Abowd, R. Allen, D. Garlan, *"Formalizing Style to Understand Descriptions of Software Architecture"*, in ACM Trans. on Software Engineering and Methodology 4:319-364, 1995
2. R. Allen, R. Douence, D. Garlan, *"Specifying and Analyzing Dynamic Software Architectures"*, in Proc. of FASE 1998, 1998
3. R. Allen, D. Garlan, *"A Formal Basis for Architectural Connection"*, in ACM Trans. on Software Engineering and Methodology 6:213-249, 1997
4. R. Allen, D. Garlan, *"A Case Study in Architectural Modelling: The AEGIS System"*, in Proc. of IWSSD-8, 1998
5. M. Bernardo, P. Ciancarini, L. Donatiello, *"On the Formalization of Architectural Types with Process Algebras"*, in Proc. of FSE-8, ACM Press, pp. 140-148, 2000
6. M. Bernardo, P. Ciancarini, L. Donatiello, *"Detecting Architectural Mismatches in Process Algebraic Descriptions of Software Systems"*, in Proc. of WICSA 2001, IEEE-CS Press, pp. 77-86, 2001
7. T.R. Dean, J.R. Cordy, *"A Syntactic Theory of Software Architecture"*, in IEEE Trans. on Software Engineering 21:302-313, 1995
8. R. Milner, *"Communication and Concurrency"*, Prentice Hall, 1989
9. M. Moriconi, X. Qian, R.A. Riemenschneider, *"Correct Architecture Refinement"*, in IEEE Trans. on Software Engineering 21:356-372, 1995
10. D.E. Perry, A.L. Wolf, *"Foundations for the Study of Software Architecture"*, in ACM SIGSOFT Software Engineering Notes 17:40-52, 1992
11. M. Shaw, D. Garlan, *"Software Architecture: Perspectives on an Emerging Discipline"*, Prentice Hall, 1996