Exogenous and Endogenous Extensions of Architectural Types

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 means of two kinds of topological extensions: exogenous and endogenous. An exogenous extension consists of attaching a set of new topology compliant components to a set of already existing components. An endogenous extension consists of replacing a set of already existing components with a set of new topology compliant components. We show that such a variability of the topology is still manageable from the analysis viewpoint.

1 Introduction

An important goal of the software architecture discipline [9, 10] 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 [6] 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 [8] 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 [3] a process algebraic approach is adopted. In such an approach the description of an architectural style via WRIGHT [2] 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 [4] 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 for the internal behavior of their architectural elements and satisfy the same architectural compatibility and interoperability properties [5].

The purpose of this paper is to make the notion of architectural type of [4] closer to the notion of architectural style through a controlled variability of the topology that preserves the properties of [5]. We propose two kinds of topological extensions: exogenous and endogenous. An exogenous extension consists of attaching a set of new topology compliant components to a set of already existing components. An endogenous extension instead consists of replacing a set of already existing components with a set of new topology compliant components. Besides giving rise to scalable architectural specifications, we show that such a controlled variability of the topology is still manageable from the analysis viewpoint, as the absence of deadlock related architectural mismatches proved via the architectural compatibility and interoperability checks scales w.r.t. the number of new components for all the exogenous extensions as well as for all the endogenous extensions satisfying a certain contraint. Finally, we prove that the endogenous extensions are more expressive than the exogenous ones.

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. 3 and 4 we enrich PADL with exogenous and endogenous extensions, respectively, and we investigate the scalability of the architectural checks. Finally, in Sect. 5 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 [4, 5].

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 $A \stackrel{\Delta}{=} E$. 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 [7], 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) given by some interactions of the AEIs that act as interfaces for the whole AT, and a fixed set of directed architectural attachments (DAAs) among the interactions of the AEIs. Every interaction is declared to be an input interaction or an output interaction and the DAAs must respect such a classification: every DAA must involve an output interaction and an input interaction of two different AEIs. Every interaction that is not an AI must be involved in at least one DAA. In order to allow for multi AEI synchronizations, every interaction can be involved in several DAAs, provided that no autosynchronization arises, i.e. no chain of DAAs is created that starts from an interaction of an AEI and terminates on an interaction of the same AEI.

We show in Table 1 a PADL textual description for a pipe-filter system. The system is composed of three identical filters and one pipe. Each filter acts as a service center of capacity two that is subject to failures and subsequent repairs. For each item processed by the upstream filter, the pipe forwards it to one of the two downstream filters according to the availability of free positions in their buffers. If both have free positions, the choice is resolved nondeterministically.

The same system is depicted in Fig. 1 through the PADL graphical notation, which is based on flow graphs [7].

archi_type	*PipeFilter*
archi_elem_types	
elem_type	*FilterT*
behavior	$Filter \triangleq accept\_item.Filter' + fail.repair.Filter$ $Filter' \triangleq accept\_item.Filter'' + serve\_item.Filter + fail.repair.Filter'$ $Filter'' \triangleq serve\_item.Filter' + fail.repair.Filter''$
interactions	**input** *accept_item* **output** *serve_item*
elem_type	*PipeT*
behavior	$Pipe \triangleq accept\_item.(forward\_item_1.Pipe + forward\_item_2.Pipe)$
interactions	**input** *accept_item* **output** $forward\_item_1$ **output** $forward\_item_2$
archi_topology	
archi_elem_instances	$F_0, F_1, F_2 : FilterT$ $P : PipeT$
archi_interactions	**input** $F_0.accept\_item$ **output** $F_1.serve\_item$, $F_2.serve\_item$

archi_attachments	**from** $F_0.serve\_item$ **to** $P.accept\_item$
	from $P.forward\_item_1$ **to** $F_1.accept\_item$
	from $P.forward\_item_2$ **to** $F_2.accept\_item$

end	

Table 1. Textual description of *PipeFilter*

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 pipe-filter example we have

$$\begin{array}{rcl} [\![FilterT]\!] = [\![F_0]\!] = [\![F_1]\!] = [\![F_2]\!] &=& Filter/\{fail, repair\} \\ [\![PipeT]\!] = [\![P]\!] &=& Pipe \end{array}$$

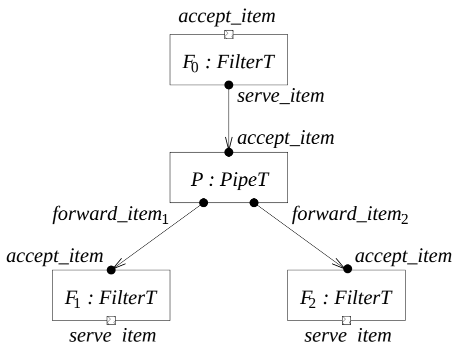

Fig. 1. Flow graph of *PipeFilter*

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 pipe-filter example we have

$$\begin{aligned}[\![PipeFilter]\!] = {} & [\![F_0]\!] [serve\_item \mapsto a] \parallel_{\emptyset} \\ & [\![F_1]\!] [accept\_item \mapsto a_1] \parallel_{\emptyset} \\ & [\![F_2]\!] [accept\_item \mapsto a_2] \parallel_{\{a, a_1, a_2\}} \\ & [\![P]\!] [accept\_item \mapsto a, \\ & \quad forward\_item_1 \mapsto a_1, forward\_item_2 \mapsto a_2]\end{aligned}$$

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*

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

■

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

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

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

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

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 $\mathcal{A}(\mathcal{C}'_1, \dots, \mathcal{C}'_m; a'_1, \dots, a'_l)$ be an invocation of the AT $\mathcal{A}$ defined with formal AETs $\mathcal{C}_1, \dots, \mathcal{C}_m$ and AIs $a_1, \dots, a_l$. $\mathcal{C}'_i$ 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*

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

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

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

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

PADL is equipped with two checks for the detection of architectural mismatches resulting in deadlocks when combining deadlock free AEIs. 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 $C_1, \dots, C_n$ be the AEIs attached to AEI K. C_i is said to be compatible with K iff*

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

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

$$\begin{aligned}[\![K; C_1, \dots, C_n]\!] = [\![K]\!]^{\mathrm{c}}_{K,C_1,\dots,C_n} \; \|_{\mathcal{S}(K;K,C_1,\dots,C_n)} \\ [\![C_1]\!]^{\mathrm{c}}_{K,C_1,\dots,C_n} \; \|_{\mathcal{S}(K;K,C_1,\dots,C_n)} \cdots \\ \cdots \|_{\mathcal{S}(K;K,C_1,\dots,C_n)} [\![C_n]\!]^{\mathrm{c}}_{K,C_1,\dots,C_n}\end{aligned}$$

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 $C_1, \dots, C_n$ be AEIs forming a cycle. C_i is said to interoperate with $C_1, \dots, C_{i-1}, C_{i+1}, \dots, C_n$ iff*

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

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

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

3 Exogenous Extensions

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 pipe-filter system of Sect. 2. Every instance of such an AT can admit a single pipe connected to one upstream filter and two downstream filters, whereas it would be desirable to be able to express by means of that AT any pipe-filter system with an arbitrary number of filters and pipes, such that every pipe is connected to one upstream filter and two downstream filters. E.g., the flow graph in Fig. 2 should be considered as a legal extension of the flow graph in Fig. 1. The idea is that, since the AIs of an AT are the frontier of the whole AT, it is reasonable to extend the AT at some of its AIs in a way that follows the prescribed topology. This cannot be done at a simple interaction because every simple interaction must occur in at least one DAA, hence it cannot be free.

An exogenous extension of an AT can take place only at a set $K_1, \dots, K_n$ of AEIs having one or more AIs and consists of attaching a set of new AEIs to one or more AIs of each of $K_1, \dots, K_n$ in a controlled way. By controlled way we mean that the addendum topologically conforms to the AT, i.e.:

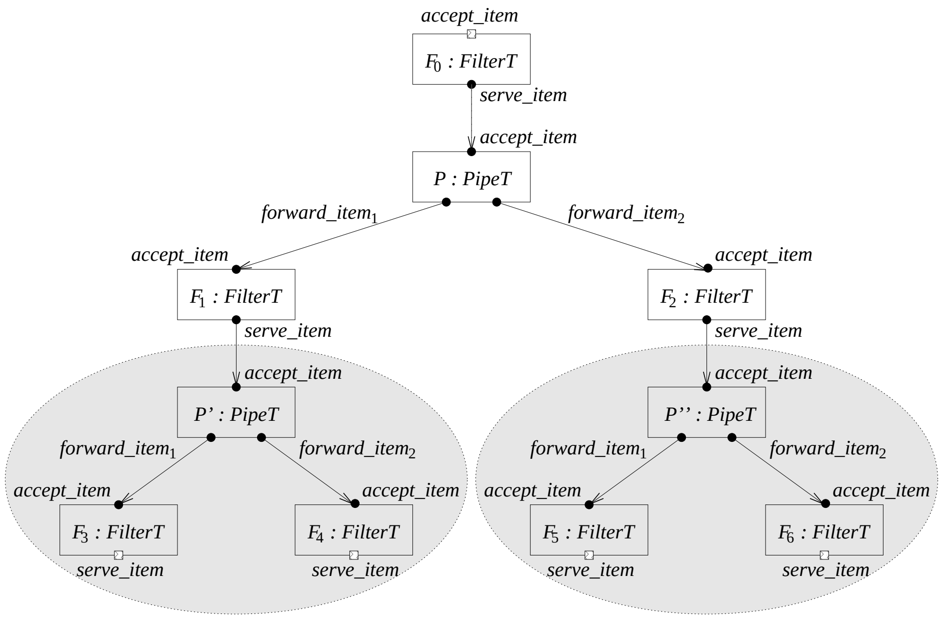

Fig. 2. Flow graph of an exogenous extension of *PipeFilter*

1. For each AEI C in the addendum, there is a corresponding AEI $\mathit{corr}(C)$ in the AT such that C has the same type as $\mathit{corr}(C)$ and an interaction a of C is simple/architectural iff the corresponding interaction a of $\mathit{corr}(C)$ is simple/architectural. Every AI in the addendum must be equal to one of the AIs of $K_1, \ldots, K_n$ involved in the extension.
2. For each AEI C and for each simple interaction a of C in the addendum, there are an AEI C' and a DAA from $C.a$ $(C'.a')$ to $C'.a'$ $(C.a)$ in the addendum iff there is a DAA from $\mathit{corr}(C).a$ $(\mathit{corr}(C').a')$ to $\mathit{corr}(C').a'$ $(\mathit{corr}(C).a)$ in the AT.
3. For each AEI K_i, $1 \leq i \leq n$, there is an AEI K'_i with the same type as K_i in the AT such that, for each AI a of K_i, there are an AEI C' in the addendum and a DAA from $K_i.a$ $(C'.a')$ to $C'.a'$ $(K_i.a)$ iff there is a DAA from $K'_i.a$ $(\mathit{corr}(C').a')$ to $\mathit{corr}(C').a'$ $(K'_i.a)$ in the AT, in which case the AI a of K_i is made simple.

The first constraint makes sure that no instances of new AETs are present in the addendum and that only the AIs of $K_1, \ldots, K_n$ involved in the extension propagate to the addendum. The second constraint guarantees that the addendum follows the same topological pattern prescribed by the AT. The third constraint ensures that the DAAs between the involved AEIs on the frontier of the AT and the addendum is consistent with the topological pattern prescribed by the AT. As can be noted, an exogenous extension never introduces a DAA between two interactions a_1, a_2 of two different AEIs C_1, C_2 of the addendum if no two interactions a_1, a_2 of two different AEIs of the same type as C_1, C_2 are attached in the AT. In other words, an exogenous extension fully preserves the type of the DAAs. We finally observe that an exogenous extension does not necessarily take place at a single AEI having AIs, but can involve several AEIs having AIs. As an example, consider a variant *PipeFilter′* of the AT *PipeFilter* in which the pipe has two upstream filters instead of a single one. In that case, an exogenous extension must involve both upstream or downstream filters.

Exogenous extensions are syntactically expressed in an AT invocation by passing an actual topology between the actual AETs and the actual names for the AIs. Such an actual topology is given by four arguments that declare the actual AEIs, the actual AIs, the actual DAAs, and the exogenous extensions, respectively. As an example, we provide below the invocation of the AT *PipeFilter* that gives rise to the flow graph in Fig. 2:

```
PipeFilter(FilterT, PipeT;
           $F_0, F_1, F_2 : FilterT, P : PipeT$;
           $F_0.accept\_item, F_1.serve\_item, F_2.serve\_item$;
           from $F_0.serve\_item$ to $P.accept\_item$,
           from $P.forward\_item_1$ to $F_1.accept\_item$,
           from $P.forward\_item_2$ to $F_2.accept\_item$;
           exo($F_3, F_4 : FilterT, P' : PipeT$;
               subst $F_3.serve\_item, F_4.serve\_item$ for $F_1.serve\_item$;
               from $F_1.serve\_item$ to $P'.accept\_item$,
               from $P'.forward\_item_1$ to $F_3.accept\_item$,
               from $P'.forward\_item_2$ to $F_4.accept\_item$;
               ),
           exo($F_5, F_6 : FilterT, P'' : PipeT$;
               subst $F_5.serve\_item, F_6.serve\_item$ for $F_2.serve\_item$;
               from $F_2.serve\_item$ to $P''.accept\_item$,
               from $P''.forward\_item_1$ to $F_5.accept\_item$,
               from $P''.forward\_item_2$ to $F_6.accept\_item$;
               );
           accept, serve, serve, serve, serve)
```

An AT invocation has the following six semicolon separated arguments:

1. The first argument is the list of actual AETs, which must conform to the corresponding formal AETs as established in Def. 5. In the invocation above, the actual AETs coincide with the formal AETs.
2. The second argument is the list of actual AEIs, whose number and types must match the number and types of the corresponding formal AEIs. In the invocation above, the actual AEIs coincide with the formal AETs.
3. The third argument is the list of actual prefixed AIs, whose number and prefixes must match the number and prefixes of the corresponding formal AIs. In the invocation above, the actual AIs coincide with the formal AIs.
4. The fourth argument is the list of actual DAAs, whose number and directions must match the number and directions of the corresponding formal DAAs. In the invocation above, the actual DAAs coincide with the formal DAAs.

5. The fifth argument is the list of exogenous extensions. Each exogenous extension has the following four semicolon separated arguments:
   (a) The first argument is the list of additional AEIs, whose types must occur in the list of actual AETs of the AT invocation. The number and types of such additional AEIs must allow the topological pattern prescribed by the AT to be preserved. In the invocation above, both exogenous extensions declare two additional instances of *FilterT* and one additional instance of *PipeT*.
   (b) The second argument is the list of substitutions of prefixed additional AIs for previously declared prefixed AIs, where all the prefixes/interactions in a substitution must be of the same type/equal and an AI can be replaced only once in an AT invocation thus becoming a simple interaction. Such substitutions must follow the topological pattern prescribed by the AT. In the invocation above, both exogenous extensions substitute the two *serve_item* interactions of the two additional instances of *FilterT* for the *serve_item* interaction of one of the two original downstream instances of *FilterT*.
   (c) The third argument is the list of additional DAAs, which must connect the replaced AIs with the interactions of the additional AEIs as well as the additional AEIs with themselves. Such DAAs must follow the topological pattern prescribed by the AT. In the invocation above, both exogenous extensions declare three DAAs: one from one of the two original downstream instances of *FilterT* to the additional instance of *PipeT*, one from the additional instance of *PipeT* to one of the two additional instances of *FilterT*, and one from the additional instance of *PipeT* to the other additional instance of *FilterT*.
   (d) The fourth argument is the list of exogenous extensions to the current exogenous extension, i.e. those exogenous extensions that can take place at the AIs declared in the substitutions of the current exogenous extension. In the invocation above, both exogenous extensions declare no nested exogenous extension. A nested exogenous extension would be declared if e.g. F_3 in Fig. 2 were attached to an instance of *PipeT* attached in turn to two downstream instances of *FilterT*.
6. The sixth argument is the list of actual names for the AIs, whose number must match the number of AIs declared in the third argument according to their possibly nested substitutions. In the invocation above, we have five AIs with the last four equally renamed, which means that the software component whose behavior is given by the AT invocation has just two interactions.

We finally investigate whether the compatibility and interoperability results proved on an AT scale to all of its exogenous extensions. In the case of the architectural compatibility check, which is concerned with acyclic ATs, we always get the desired scalability from an AT to all of its exogenous extensions provided that no cycles are introduced. Note that cycles can be introduced in an acyclic AT when performing an exogenous extension, as is the case with *PipeFilter*$'$.

Theorem 5. *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, C_i is compatible with K for all $i = 1, \ldots, n$, and every further AEI that can be attached to K through an acyclic exogenous extension is compatible with K, then $[\![K; C_1, \ldots, C_n]\!]$ and all of its acyclic exogenous extensions involving K 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 has no AIs. If K has AIs and an arbitrary acyclic exogenous extension involves some of them, then the corresponding acyclic exogenous extension of $[\![K; C_1, \ldots, C_n]\!]$ is deadlock free by virtue of Thm. 2, because C_i is compatible with K for all $i = 1, \ldots, n$ and every AEI attached to K in the acyclic exogenous extension is compatible with K by hypothesis.* ■

Corollary 2. *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 and all of its acyclic exogenous extensions are deadlock free.*

Proof. *It follows from the theorem above and the three constraints that establish that any addendum must topologically conform to the AT. In particular, we observe that the third constraint provides the additional information required by the theorem above about the satisfaction of the compatibility condition on the frontier of K.* ■

In the case of the architectural interoperability check, which is concerned with cyclic ATs, we obtain the desired scalability (for individual cycles) from an AT to all of its exogenous extensions, because attaching an addendum to one or more AEIs in the cycle does not alter the interoperability of the AEIs within the cycle.

Theorem 6. *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 even in the presence of an exogenous extension involving some of $C_1, \ldots, C_n$.* ■

As an example of application of the architectural checks above, let us consider the AT *PipeFilter*. It is easy to see that F_0, F_1, F_2 are all compatible with P, hence we can conclude by Thm. 2 that $[\![\mathit{PipeFilter}]\!]$ is deadlock free. Let us now consider the exogenous extension of *PipeFilter* depicted in Fig. 2. By applying Thm. 5 to F_1 and F_2, we obtain that $[\![F_1; P, P']\!]$ and $[\![F_2; P, P'']\!]$ are deadlock free. By subsequently applying Cor. 2, we further obtain that every exogenous extension of *PipeFilter* is deadlock free, hence so is in particular the one in Fig. 2.

4 Endogenous Extensions

Besides exogenous extensions, there are other desirable forms of variability in the topology of an AT. As an example, let us consider Table 2, which provides a

archi_type	*Ring*
archi_elem_types	
elem_type	*InitStationT*
behavior	$InitStation \triangleq start.InitStation''$
	$InitStation' \triangleq receive.process.InitStation''$
	$InitStation'' \triangleq send.InitStation'$
interactions	**input** *receive*
	output *send*
elem_type	*StationT*
behavior	$Station \triangleq receive.process.send.Station$
interactions	**input** *receive*
	output *send*
archi_topology	
archi_elem_instances	IS : *InitStationT*
	S_1, S_2, S_3 : *StationT*
archi_interactions	
archi_attachments	**from** $IS.send$ **to** $S_1.receive$
	from $S_1.send$ **to** $S_2.receive$
	from $S_2.send$ **to** $S_3.receive$
	from $S_3.send$ **to** $IS.receive$
end	

Table 2. Textual description of *Ring*

PADL description of a ring of stations each following the same protocol: wait for a message from the previous station in the ring, process the received message, and send the processed message to the next station in the ring. Since such a protocol guarantees that only one station can transmit at a given instant, the protocol can be considered as an abstraction of the IEEE 802.5 standard medium access control protocol for local area networks known as token ring. One of the stations is designated to be the initial one, in the sense that it is the first station allowed to send a message. The PADL description in Table 2 declares one instance of the initial station and three instances of the normal station. Every instance of the AT *Ring* can thus admit a single initial station and three normal stations connected to form a ring, whereas it would be desirable to be able to express by means of that AT any ring system with an arbitrary number of normal stations. E.g., the flow graph in Fig. 3 should be considered as a legal extension of the AT *Ring*. The idea is that of replacing a set of AEIs with a set of new AEIs following the topology prescribed by the AT. In this case, we consider the frontier of the AT w.r.t. one of the replaced AEIs to be the set of interactions previously attached to the simple interactions of the replaced AEI. On the other hand, all the replacing AEIs that will be attached to the frontier of the AT w.r.t. one of the replaced AEIs must be of the same type as the replaced AEI.

An endogenous extension of an AT can take place at a set $K_1, \dots, K_n$ of AEIs, with K_i attached to $K_{i,1}, \dots, K_{i,n_i}$ for all $i = 1, \dots, n$, and consists of substituting a set $\mathcal{S}$ of AEIs for $K_1, \dots, K_n$ in a controlled way. By controlled way we mean that $\mathcal{S}$ topologically conforms to the AT, i.e.:

1. For each AEI C in $\mathcal{S}$, there is a corresponding AEI $corr(C)$ in the AT such that C has the same type as $corr(C)$ and an interaction a of C is simple/architectural iff the corresponding interaction a of $corr(C)$ is simple/architectural. Every AI in $\mathcal{S}$ must be equal to one of the AIs of $K_1, \dots, K_n$.
2. For each AEI C and for each simple interaction a of C in $\mathcal{S}$, there are an AEI C' and a DAA from $C.a$ ($C'.a'$) to $C'.a'$ ($C.a$) in $\mathcal{S}$ iff there is a DAA from $corr(C).a$ ($corr(C').a'$) to $corr(C').a'$ ($corr(C).a$) in the AT.
3. For each AEI $K_{i,j}$, $1 \leq i \leq n$, $1 \leq j \leq n_i$, and for each interaction a of $K_{i,j}$ attached to an interaction a' of K_i in the AT, there are an AEI C' with the same type as K_i in $\mathcal{S}$ and a DAA from $K_{i,j}.a$ ($C'.a'$) to $C'.a'$ ($K_{i,j}.a$). If $C'.a'$ is an AI, it is made simple.

The three constraints above are similar to those for the exogenous extensions. The only difference is in the third constraint, which is now simpler because of the requirement that C' has the same type as K_i.

Similarly to exogenous extensions, when invoking an AT the endogenous extensions are syntactically expressed through four additional arguments for the actual topology. As an example, we provide below the invocation of the AT *Ring* that gives rise to the flow graph in Fig. 3:

```
Ring(InitStationT, StationT;
     IS : InitStationT, S_1, S_2, S_3 : StationT;
     ;
     from IS.send to S_1.receive,
     from S_1.send to S_2.receive,
     from S_2.send to S_3.receive,
     from S_3.send to IS.receive;
     endo(subst S_2', S_2'' : StationT for S_2 : StationT;
          ;
          from S_1.send to S_2'.receive,
          from S_2'.send to S_2''.receive,
          from S_2''.send to S_3.receive;
          );
     )
```

The fifth argument above is the list of endogenous extensions, which can be interleaved with the exogenous extensions. Each endogenous extension has the following four semicolon separated arguments:

1. The first argument is the substitution of new AEIs for previously declared AEIs, where the types of the replacing AEIs must occur in the list of actual AETs of the AT invocation and an AEI can be replaced only once in an AT

invocation. The number and types of such replacing AEIs must allow the topological pattern prescribed by the AT to be preserved. In the invocation above, S_2' and S_2'' substitute for S_2.

2. The second argument is the list of substitutions of prefixed additional AIs for previously declared AIs of the replaced AEIs that consequently become simple, where all the prefixes/interactions in a substitution must be of the same type/equal. Such substitutions must follow the topological pattern prescribed by the AT. In the invocation above, there are no AI substitutions as there are no AIs.
3. The third argument is the list of DAAs. Some of them replaces the DAAs involving the replaced AEIs (see those from S_1 to S_2' and from S_2'' to S_3), while the others are new DAAs connecting the replacing AEIs (see the one from S_2' to S_2''). Such DAAs must follow the topological pattern prescribed by the AT.
4. The fourth argument is the list of endogenous/exogenous extensions to the current exogenous extension, i.e. those extensions that can take place at the AEIs/AIs declared in the substitutions of the current endogenous extension. In the invocation above, there are no nested extensions.

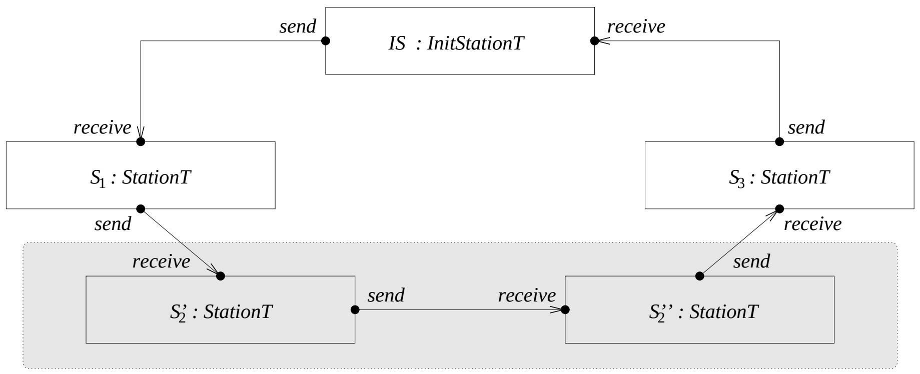

Fig. 3. Flow graph of an endogenous extension of *Ring*

We finally investigate whether the compatibility and interoperability results proved on an AT scale to all of its endogenous extensions. In the case of the architectural compatibility check, which is concerned with acyclic ATs, we always get the desired scalability from an AT to all of its endogenous extensions provided that no cycles are introduced. Note that cycles can be introduced in an acyclic AT when performing an endogenous extension, as is the case with *PipeFilter*$'$ if we perform an endogenous extension at both downstream filters.

Theorem 7. *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 $i =$*

$1, \dots, n$, then $[\![K; C_1, \dots, C_n]\!]$ and all of its acyclic endogenous extensions taking place at some among $C_1, \dots, C_n$ are deadlock free.

Proof. *The first part of the result stems directly from Thm. 2. Suppose now that an endogenous extension takes place at $C_{i_1}, \dots, C_{i_m}$ with $1 \leq i_1, \dots, i_m \leq n$. Since for each i_j the new AEI C'_{i_j} attached to K in place of C_{i_j} must be of the same type as C_{i_j}, the second part of the result follows by virtue of Thm. 2.* ■

Corollary 3. *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 and all of its acyclic endogenous extensions are deadlock free.*

Proof. *It follows from the theorem above and the three constraints that establish that any replacing set of AEIs must topologically conform to the AT.* ■

In the case of the architectural interoperability check, which is concerned with cyclic ATs, we obtain the desired scalability (for individual cycles) from an AT only to those of its endogenous extensions such that the set of replacing AEIs is weakly bisimilar to the set of replaced AEIs.

Theorem 8. *Given an AT, let $C_1, \dots, C_n$ be AEIs forming a cycle. If there exists C_i such that $[\![C_i]\!]^{\mathrm{c}}_{C_1,\dots,C_n}$ is deadlock free and C_i interoperates with $C_1, \dots, C_{i-1}, C_{i+1}, \dots, C_n$, then $[\![C_1, \dots, C_n]\!]^{\mathrm{c}}$ is deadlock free and so is each of its endogenous extensions substituting $C'_1, \dots, C'_{m'}$ for $C_{j_1}, \dots, C_{j_m}$, with $1 \leq j_1, \dots, j_m \leq n$, such that, denoted by $\mathcal{I}'$ the set of simple interactions of $C'_1, \dots, C'_{m'}$ that are not attached to the interactions of the AEIs formerly attached to $C_{j_1}, \dots, C_{j_m}$, there exist two relabelings φ and φ', which may not be injective only on the AIs of $C'_1, \dots, C'_{m'}$, such that $[\![C'_1, \dots, C'_{m'}]\!]^{\mathrm{c}}_{C'_1,\dots,C'_{m'}} / \mathcal{I}'[\varphi'] \approx_{\mathrm{B}} [\![C_{j_1}, \dots, C_{j_m}]\!]^{\mathrm{c}}_{C_1,\dots,C_n}[\varphi]$.*

Proof. *The first part of the result stems directly from Thm. 3. If an endogenous extension takes place at $C_{j_1}, \dots, C_{j_m}$, then the resulting extended cycle is still deadlock free, because the original cycle is deadlock free by Thm. 3, the extended cycle is weakly bisimilar to the original one as $[\![C'_1, \dots, C'_{m'}]\!]^{\mathrm{c}}_{C'_1,\dots,C'_{m'}} / \mathcal{I}'[\varphi'] \approx_{\mathrm{B}} [\![C_{j_1}, \dots, C_{j_m}]\!]^{\mathrm{c}}_{C_1,\dots,C_n}[\varphi]$ and $\approx_{\mathrm{B}}$ is a congruence w.r.t. the static operators, and $\approx_{\mathrm{B}}$ preserves deadlock freedom.* ■

As an example of application of the architectural checks above, let us consider the AT *Ring*. It is easy to see that each of IS, S_1, S_2, S_3 is deadlock free and interoperates with the others, hence we get from Thm. 3 that $[\![Ring]\!]$ is deadlock free. Let us now consider the endogenous extension of *Ring* depicted in Fig. 3. Since $[\![S'_2, S''_2]\!]^{\mathrm{c}}_{S'_2,S''_2}$ is weakly bisimilar to $[\![S_2]\!]^{\mathrm{c}}_{IS,S_1,S_2,S_3}$ up to relabeling when the interactions in the DAAs between S'_2 and S''_2 are hidden, from Thm. 8 we obtain that the endogenous extension of *Ring* in Fig. 3 is deadlock free.

We conclude by observing the endogenous extensions are more expressive than the exogenous ones.

Theorem 9. *Given an AT, each of its exogenous extensions has an associated endogenous extension such that the two corresponding invocations of the AT result in the same instance of the AT.*

Proof. *Given an arbitrary exogenous extension of the AT, its associated endogenous extension takes place at the AEIs whose AIs are involved in the exogenous extension, and it is obtained by considering as set of new AEIs the same new AEIs as the exogenous extension together with, for each AEI whose AIs are involved in the exogenous extension, a new AEI of the same type.* ■

5 Conclusion

In this paper we have enriched the notion of AT of [4] by introducing the capability of expressing exogenous and endogenous extensions of the topology, in such a way that the architectural checks of [5] scale w.r.t. the number of additional software components for all the exogenous extensions as well as for all the endogenous extensions satisfying a certain contraint. Finally, we have proved that the endogenous extensions are more expressive than the exogenous ones.

As far as future work is concerned, first we would like to investigate whether information can be gained about the interoperability of cycles that are generated when performing an exogenous extension on an AT, starting from the compatibility of the involved AEIs of the AT. Second, we would like to study whether the additional constraint for the interoperability result in the case of endogenous extensions can be weakened. Third, we would like to compare our approach to topological extensions with graph grammar based approaches.

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