A Generic Approach to Connector Architectures
Part I: The General Framework

Fernando Orejas
Universitat Politècnica de Catalunya, Dept. LSI
Campus Nord, Modul C5, 08034 Barcelona
orejas@lsi.upc.es

Hartmut Ehrig
Technische Universität Berlin, SeKr. FR6-1,
Franklinstr. 28/29, 10587 Berlin
ehrig@cs.tu-berlin.de

Markus Klein
Technische Universität Berlin, SeKr. FR6-1,
Franklinstr. 28/29, 10587 Berlin
klein@cs.tu-berlin.de

Julia Padberg
Technische Universität Berlin, SeKr. FR6-1,
Franklinstr. 28/29, 10587 Berlin
padberg@cs.tu-berlin.de

Elvira Pino
Departament LSI
Universitat Politècnica de Catalunya, Dept. LSI
Campus Nord, Mòdul C5, 08034 Barcelona
pino@lsi.upc.es

Sonia Pérrez
CEIS, Inst. Sup. Politécnico José Antonio Echevarría,
Havana, Cuba
sperezJ@ceis.cujae.edu.cu

Abstract. The aim of this paper is to present a generic framework for the modelling of component-based systems using architectural connectors. More precisely, concepts of component, connector and architecture are presented in a formal generic way, which are independent of any semi-formal or formal modelling approach. The idea is that one could use this framework to define component and connector notions for every given modelling formalism. As a main result, we define the semantics of architectures using graph transformation, showing that the semantics is independent of the order in which the connections are computed, and that the semantics is compatible with transformation. In the continuation of this paper, we show the applicability of our ideas. In particular, our framework is instantiated by Petri nets and CSP, including a case study using Petri Nets.

Keywords: Architectural connectors, formal software modelling, software architecture

1. Introduction

The development of component-based systems is nowadays an important area in software engineering. In this context, a lot of work has been dedicated to different issues related to this approach, such as the development of related methodologies, the development of techniques or criteria for the selection
of components-off-the-shelf, or the implementation of middleware and other related tools. Moreover, several Architecture Description Languages (ADL) have been proposed for the specification of components and their interconnection (e.g., [22, 26, 37, 27, 1, 39, 16, 28, 4, 3, 30, 2, 6]). See [29] for a classification and comparison of several ADL. Unfortunately, in many cases these languages concentrate on the description of the coordination of components, not considering their functionality. Moreover, the formalisms underlying these languages do not necessarily coincide with some traditional specification or modelling formalisms. This means that a number of standard software modelling techniques or formalisms lack adequate notions to support the architectural development of component-based systems. See, for instance, [20] for a discussion on how to extend UML with component concepts.

An approach that we consider very interesting is described in [1, 38]. This work presents an approach, based on the use of architectural connectors, for the architectural development of software systems that we consider highly relevant for practical applications. In this approach architectures are built in terms of two kinds of units: components and connectors. Components are not connected directly, but through connectors. Components offer some functionality and connectors describe policies of interaction of the connected components. The language used for the specification and modelling of components and connectors was Wright ([1, 38]), built over CSP [21]. For instance, the example below, taken from [1] describes a very simple pipe connector using WRIGHT that could be used to model pipe-filter architectures, where each component can be seen as a stream processing unit and the Pipe connectors would be used to interconnect these units. In particular, that connector receives messages from a Writer component and sends messages to a Reader component.

\[
\text{connector Pipe =}
\]

\[
\text{role Writer = write } \rightarrow \text{Writer } \sqcap \text{close } \rightarrow \sqrt{ }
\]

\[
\text{role Reader = (read } \rightarrow \text{Reader } \sqcap \text{eof } \rightarrow \text{close } \rightarrow \sqrt{ }) \sqcap (\text{close } \rightarrow \sqrt{ })
\]

\[
\text{glue = Writer.write } \rightarrow \text{glue } \sqcap \text{Reader.read } \rightarrow \text{glue }
\]

\[
\sqcap \text{Reader.close } \rightarrow \text{ReadOnly } \sqcap \text{Reader.close } \rightarrow \text{WriteOnly}
\]

\[
\text{where }
\]

\[
\text{ReadOnly = Reader.read } \rightarrow \text{ReadOnly }
\]

\[
\sqcap \text{Reader.eof } \rightarrow \text{Reader.close } \rightarrow \sqrt{ }
\]

\[
\sqcap \text{Reader.close } \rightarrow \sqrt{ }
\]

\[
\text{WriteOnly = Writer.write } \rightarrow \text{WriteOnly } \sqcap \text{Writer.close } \rightarrow \sqrt{ }
\]

The roles of the connector are its interfaces, and they describe the expected behavior of the components that will be connected to the connector. Then, the glue describes the coordination of these components.

The work using this approach was followed by Fiadeiro et al (e.g., see [16, 18, 25, 15]), who generalized in some sense the approach by putting it into a categorical context in the framework of the coordination language COMMUNITY. Moreover, in [18, 25] they defined the notion of Architectural School which allowed them to generalize their results to any specification formalism satisfying some given conditions. This latter work has similar aims to the work that we present in this paper. In the conclusions we present a detailed comparison of both approaches.

The aim of this paper is to present such a framework for the modelling of component-based software systems following the architectural connector’s approach. The idea is that one could use this framework to define component and connector notions for any given modelling approach. These definitions could serve as the basis for the definition of an ADL based on this approach. In this sense, we
have defined concepts of component, connector and architecture which are independent of any modelling formalism. Components and architectural connectors can be seen as some kind of templates that can be filled with specifications written using any given formalism, and an architecture can be seen as a certain form of interconnecting component and connectors, where the links used for this interconnection and the internal links in components and connectors have some specific meaning in the given formalism. We could think that these concepts are parameterized by the modelling formalism to be used and that, to use our approach in the context of a given modelling technique, we need to instantiate the framework by that technique.

It should be clear that here we are not proposing a new ADL. Not even after instantiation of the framework in terms of a given modelling approach we would have an ADL. As said above, we would just have the basic concepts that could serve for the definition of an ADL. For instance, if we want to define an ADL we would need some scope and visibility rules that allow the implicit identification of elements of a system description.

This kind of generic approach is interesting if we can ensure that, when used in connection to a given modelling formalism, the resulting framework after instantiation satisfies some kind of desirable properties. In this sense, the properties on which we have concentrated are compositionality, support to hierarchical design and compatibility with refinement. Compositionality means that the semantics of an architectural system should depend only on the components, the connectors and the connections involved. Support to hierarchical design means that it should be possible to see a given (sub)system as a component. Finally, compatibility with refinement means that if in a given system we replace one component by a more refined component, then the result is a refinement of the original system. In our case, refinements are considered a special case of what we call transformations. In particular, in the paper we prove compatibility with respect to this more general notion.

Obviously, one will not have these properties for free. Nor will we be able to define the semantics of a given architectural system if we do not know anything about the specification formalism that we are using. Therefore, we have identified some basic requirements (constructions satisfying some given basic properties) that allow us to define the semantics of architectural systems in the desired way. We have called a component framework any modelling technique satisfying these basic requirements. In these sense, we have shown how, given any component framework, we can define the semantics of an architectural system by computing its connections using graph transformation. In particular, the semantics of an architecture is just a component, allowing for hierarchical development of these systems. Moreover, we show that the semantics is independent of the order in which these connections are computed. Finally, we also show that the semantics is compatible with transformation.

There are other important properties that one would like to ensure for a given architectural approach that we do not consider in this paper. In particular, behavioral properties like deadlock absence. The reason for not considering these properties is that we would need in our framework much more detail about how a given formalism specifies a given behavior.

In this paper, we do not study how to reason about architectural systems in our framework. However we think that our semantic definitions provides a basis for this kind of study. Actually, in [35] we present some very preliminary work along these lines. We also think that using some deductive techniques for dealing with structured systems, like the ones presented in [5], may help.

To show how our framework works, in the second part of this work we consider as case studies the instantiation to two specification techniques: Petri nets and CSP. In addition, we use the Petri Net case to present a small case study to motivate our concepts and results. Nevertheless, in order to facilitate the understanding of the main concepts of our generic framework, in this paper we will use a simpler (though unrealistic) modeling formalism as a running example for the introduction of the framework. In particular, we will see how we can instantiate or component framework when models are just sets.
Summarizing, we think that this work may be considered interesting from several points of view. First, the paper provides various notions related to connector architectures that can be used uniformly in the context of different modelling formalisms. Second, the paper describes in detail, at an abstract level, what are the basic elements and properties that one should take into account when using this kind of constructions in the context of a given modelling formalism for ensuring the properties considered in this paper. In particular, we think that the definition of these basic constructions can be seen as guidelines to define architectural connectors and components in the framework of any given modelling approach. Actually, the two instances of our framework presented in the second part of this work, Petri Nets and CSP, are an example of this. Finally, it must be pointed out that our approach is not restricted to the use of formal modelling techniques. We believe that different degrees of formality may be needed when modelling a given system. In particular, when describing critical aspects of a system a formal technique is needed. However, when describing some less critical component, a semiformal approach could be sufficient. In this sense, as said above, our approach can be used in connection with formal and semiformal modelling techniques. The issue is related on how we instantiate the basic ingredients of our framework, in particular the notion of transformation that is used to connect the interfaces of connectors and components. If transformations are defined on the syntactic level (as we do in the second part, in the case of Petri Nets) then a complete formal semantics is not needed. Actually, in [8, 31, 32] we have sketched the use of our framework in the context of UML.

Something that may be obvious, that we think that is worth to point out, is that our approach may also be instantiated to the case where components and connectors use different formalisms in their interfaces and in their bodies (as it happens, e.g. in [17]), if we can provide the same kind of semantics for both formalisms. In particular, in this case it is enough to define the notions of embedding and transformation at this semantic level. For instance, this may be the case if we use a logical formalism in the interfaces, like some kind of temporal logic, and some process description technique in the bodies, like Petri nets, and if we provide some common semantics in terms of, for instance, some kind of state-transition systems.

This paper can be seen as a mature presentation of the basic approach in which we have been working in the last years [13, 8, 31, 24, 35]. More precisely, the paper [13] can be seen as a very preliminary version of this one. In particular, there are important differences between the two papers that concern some of the basic definitions and the main results. As said above, [8, 31, 32] sketch the use of our framework in the context of UML. In [24] we consider a possible extension of the framework, where each interface of a given component can be connected to more than one interface of another component. Finally, as mentioned above, in [35], we present some preliminary ideas about how to reason about architectural systems in our framework. Previously, in [11, 12, 34] we studied a much simpler component framework, where we considered a single kind of components, which were allowed to have only one import interface and one export interface. This restriction provides a considerable simplification when considering the semantics of composition and it would only allow us to define “linear” architectures (i.e. consisting of a sequence of components where each component is connected to the next one forming a sequence).

For editorial reasons, this work has been divided into two papers that are intended to be read together, although they are both self-contained. This first part is organized as follows. In the following section we study the basic notions of our approach describing the kind of constructions that a given modelling formalism should have to be used in connection with our approach. In section 3, we present how components can be composed by means of connectors and how we can build architectures. However these concepts are just presented syntactically. In particular, the semantics of component composition is studied in section 4 and the semantics of architectures is studied in section 5. Finally, in section 6 we draw some conclusions.
Then, as pointed out already, instantiations of our generic framework to Petri nets and to CSP are discussed in the second part of this work, together with a small case study providing a connector architecture based on Petri nets.

2. A generic framework for architectural components and connectors

In this section we describe the basic ingredients of our generic framework. More specifically, we will study what are the essential ingredients and constructions that, according to our framework, a given formalism should have to be used for the modeling of architectural components and connectors. To ease the understanding, we will use a very simple running example where models are just sets.

Following the approach introduced in [1], we consider two kinds of constructions: components and connectors. Components are units that provide some kind of services and that can be connected with other components through connectors. On the other hand, connectors are units that describe the interaction of the connected components.

Informally, a component is a unit consisting of a body and $n$ interfaces, which we call ports following [1]. In the body, the services provided by the component are fully described or implemented. On the other hand, a port provides an abstract view of the body i.e. of its behavior and of the services provided. Conversely, we may see the body of a component as a refinement of its interfaces. These refinement relations are called transformations.

**Definition 2.1. (Components)** A component $COMP = (B, p_1 : P_1 \Rightarrow B, \ldots, p_n : P_n \Rightarrow B)$ for $n \geq 0$ is given by the body $B$ and a family of ports $P_i$ with port transformations $p_i : P_i \Rightarrow B$ for $i \in \{1, \ldots, n\}$. A component is graphically represented by its diagram $D_{COMP}$, which is depicted in Fig. 1.

As said above, components are connected through architectural connectors. In particular, a connector is a unit consisting of two or more interfaces, called roles, and a body (defined in terms of the glue in [1]). The roles describe the expected behavior and services provided by the component to which they will be connected, while the body describes the interaction of the components that will be connected through the roles. In this sense, the body of a connector extends the specifications included in the roles by defining over them an interaction policy. We say that the roles in a connector are embedded in the body.

**Definition 2.2. (Connectors)** A connector $CON = (B, r_1 : R_1 \rightarrow B, \ldots, r_n : R_n \rightarrow B)$ for $n \geq 2$ is given by the body $B$ and a family of roles $R_i$ with role embeddings $r_i : R_i \rightarrow B$ for $i \in \{1, \ldots, n\}$. A connector is graphically represented by its diagram $D_{CON}$, which is depicted in Fig. 2.
Components are connected to an architectural connector by linking the roles of the connector to some ports of the connected components. To be able to link a role to a given port, they must be compatible. As said above, a role describes some required (abstract) behavior and services from a given component. On the other hand, a port describes some concrete services and behavior provided by a given component. So, to be compatible, the port should typically be some kind of refinement of the role. For instance, in the example presented in the introduction we could connect the pipe (through the Writer role) to any component that can produce a sequence of data and (through the Reader role) to any component that can consume a sequence of data. In these cases, these components would need to have ports that describe the specific behavior of the components with respect to the production/consumption of data, respectively.

However, before defining the semantics of this kind of composition, we must first specify in detail what are the basic elements (or parameters) that a concrete modelling or specification framework must provide to allow us to instantiate our generic framework for defining concrete notions of architectural components and connectors.

First of all, obviously, the given concrete framework must include a class of specifications or models. In particular, the bodies and the interfaces of connectors and components are assumed to be models in this class. The given framework should also provide a class of transformations and a subclass of embeddings between models, since embeddings can be seen as a special kind of refinements. However, to define an adequate semantics for the constructions of our generic framework, ensuring the compositionality of the interconnection operations, we must impose some requirements on the kinds of embeddings and transformations considered for the given specification or modelling formalism. In particular, first, we require that embeddings and transformations should be closed under composition, where this composition is associative and there is a special identity which is neutral with respect to embedding and transformation composition.

Example 2.1. As said in the introduction, we will study the instantiation of our framework in terms of a very simple modeling formalism. In particular, we will consider that models are sets, where its elements may be of any kind. This is clearly unrealistic, but the interested reader is addressed to the second part of this paper to study two realistic instantiations. Anyhow we may think that, in this context, the kind of components that we want to define in this modeling framework are not computing components, but just information components, i.e. components that store some information. Then, we may think that the interfaces of components are just abstract references (e.g. sets of keywords) to this information. In this context, the body of connectors may be seen as sets of keywords, which include the keywords in their roles, and that are used to link the information from different components.

Therefore, according to this intuition, we consider that the embeddings in this formalism are set inclusions and transformations are relations, since several keywords may refer to the same information and a keyword may refer to several information items. Then, obviously, embeddings can be seen as a special case of transformations and they are both closed under composition. Moreover, composition is associative and the identity relation is neutral with respect to composition.

When we compose a connector with several components we may think that this is like replacing the roles in the connector by the ports (or by the bodies) of the corresponding component. This is equivalent to ask embeddings and transformations to be compatible in the following sense. If a given model $M$ embeds a family of models $M_1, \ldots, M_n$ and we know that each of these submodels can be transformed into the models $M'_1, \ldots, M'_n$, respectively, then we should be able to transform in parallel all these submodels in the context of $M$ yielding a model $M'$, which embeds $M'_1, \ldots, M'_n$ and is a transformation of $M$. However this is not always possible. We may have situations where
it is not reasonable to think that this kind of parallel substitution, which we call parallel extension should exist. In particular, if a given model \( M \) embeds a family of models \( M_1, \ldots, M_n \) and we know that some of these submodels share some common parts, then a family of transformations of these submodels will have a parallel extension if these transformations agree on the common parts of the submodels. However, in some cases, even in the simplest situation, when \( n = 1 \) there may be some kind of inconsistency between the embedding and the transformation that makes them incompatible. This happens, for instance if models are some kind of graphs, embeddings are just injective graph morphisms, transformations are defined by double-pushout graph transformation and the embedding does not satisfy the so-called gluing condition ([10]). The condition that ensures the existence of parallel extension is called parallel consistency. This notion of consistency is also generic and has to be instantiated differently for different specification or modelling techniques.

**Definition 2.3. (Parallel Consistency)** A parallel consistency relation is a relation between families of transformations \( \{ t_j : M_j \Rightarrow M'_j \}_{j \in J} \) and families of embeddings \( \{ e_j : M_j \rightarrow M \}_{j \in J} \), with same \( M_j \) for all finite non-empty index sets \( J \).

If \( \{ t_j : M_j \Rightarrow M'_j \}_{j \in J} \) and \( \{ e_j : M_j \rightarrow M \}_{j \in J} \) are in this relation, then \( \{ t_j : M_j \Rightarrow M'_j \}_{j \in J} \) is called parallel consistent with respect to \( \{ e_j : M_j \rightarrow M \}_{j \in J} \).

**Example 2.2.** As said above, the intuition of parallel consistency is that the elements that are shared by some submodels must be transformed consistently by the corresponding transformation. In our example, this means that a family of relations \( \{ R_1 \subseteq A_1 \times C_1, \ldots, R_n \subseteq A_n \times C_n \} \) is parallel consistent with respect to the family of inclusions \( \{ A_1 \subseteq B, \ldots, A_n \subseteq B \} \) if for every \( a \) such that \( a \in A_i \) and \( a \in A_j \) and for every \( c \in C_i \cup C_j \) we have that \( aR_{i,c} \) if and only if \( aR_{j,c} \).

Parallel consistency should ensure the existence of parallel extension and other related properties:

**Definition 2.4. (Parallel Extension Property and Diagram)** A parallel consistency relation satisfies the parallel extension property if we have:

1. **Parallel extension and diagram.** For each parallel consistent family of transformations \( \{ t_j : M_j \Rightarrow M'_j \}_{j \in J} \) with respect to a family of embeddings \( \{ e_j : M_j \rightarrow M \}_{j \in J} \), there is a selected transformation \( t : M \Rightarrow M' \), together with a selected family of embeddings \( \{ e'_j : M'_j \rightarrow M' \}_{j \in J} \), called parallel extension of \( \{ t_j : M_j \Rightarrow M'_j \}_{j \in J} \) with respect to \( \{ e_j : M_j \rightarrow M \}_{j \in J} \), such that the diagram in Fig. 3, called parallel extension diagram, commutes.

2. **Parallel extension diagrams are closed under vertical composition.** This means that if \( t : M \Rightarrow M' \) is the parallel extension of \( \{ t_j : M_j \Rightarrow M'_j \}_{j \in J} \) with respect to \( \{ e_j : M_j \rightarrow M \}_{j \in J} \) and \( t' : M' \Rightarrow M'' \) is the parallel extension of \( \{ t'_j : M'_j \Rightarrow M''_j \}_{j \in J} \) with respect to \( \{ e'_j : M'_j \rightarrow M''_j \}_{j \in J} \), as in Fig. 4, then \( t' \circ t : M \Rightarrow M'' \) is the parallel extension of \( \{ t'_j \circ t_j : M_j \Rightarrow M''_j \}_{j \in J} \) with respect to \( \{ e'_j : M_j \rightarrow M''_j \}_{j \in J} \).

3. **Extension of embeddings.** In the special case when \( J = \{ 0 \} \), if the transformation \( t_0 : M_0 \Rightarrow M'_0 \) is an embedding then, for every embedding \( e_0 : M_0 \rightarrow M \), \( t_0 \) is parallel consistent with respect to \( e_0 \). This means that the parallel extension diagram in Fig. 5 always exists, and \( t' \) is also an embedding. In this case, that diagram is called extension diagram of \( t_0 \) with respect to \( e_0 \).
Given the extension diagram in Fig. 5, with \( t_0 \) and \( t \) being embeddings, and given a family \( \{ e_k : M_k \rightarrow M \} \) for \( k \leq k \leq n \), we have that the diagram in Fig. 6 is a parallel extension diagram, where \( e'_k \) is the composition of \( e_k \) and \( t \), for every \( k, 1 \leq k \leq n \).

The first condition in the above definition states that parallel consistency ensures the existence of parallel extension. Moreover, we assume that parallel extension is uniquely defined by the given \( \{ t_j \} \) and \( \{ e_j \} \). In particular, there may be several families \( \{ t'_j \} \) and \( \{ e'_j \} \) that could satisfy this extension property. Our assumption means that only one such \( \{ t'_j \} \) and \( \{ e'_j \} \) are chosen, in some well-defined way, as the parallel extension of \( \{ t_j \} \) with respect to \( \{ e_j \} \). The second condition states that we can iterate this kind of parallel transformations. The third condition states that an embedding is also parallel consistent with respect to another embedding. This is quite reasonable since, if the intuition of embeddings is that they are some form of inclusion, then this third condition essentially demands that there must exist some sort of union. Actually, in the case of graph transformation where, as pointed above, an arbitrary transformation may be incompatible (i.e. not parallel consistent) with respect to a given embedding, condition 3) also holds. Finally, condition 4) states that an extension of embeddings can be seen as a larger parallel extension involving other embedded submodels \( M_1, \ldots, M_n \), where these other submodels are left unchanged. This condition could have been weakened slightly by asking \( M_1, \ldots, M_n \) to be disjoint with respect to \( M_0 \).
Example 2.3. Let us see that the parallel consistency relation defined in Example 2.2 satisfies the parallel extension property for a certain definition of parallel extension. Given \( R = \{ R_1, \ldots, R_n \} \) and \( A_1 \subseteq B, \ldots, A_n \subseteq B \) where, for each \( j : (1 \leq j \leq n) \), \( R_j \) is a relation between \( A_j \) and a set \( C_j \) then we can define the parallel extension of the family of relations \( R_j \) with respect to \( i \) as follows:

- The resulting set \( D \) is defined:

\[
D = \bigcup_{1 \leq j \leq n} C_j \cup (B \setminus (\bigcup_{1 \leq j \leq n} A_j))
\]

where \( \cup \) denotes disjoint union. This means that \( D \) consists of all the elements in the sets \( C_j \) plus all the elements in \( B \) which are not included in an \( A_j \) (i.e. which have not been transformed by one of the given the elements in the sets \( C_j \) are considered different from the elements in \( B \setminus (\bigcup_{0 \leq j \leq n} A_j) \)). This is the reason for the disjoint union in the definition of \( D \).

- Obviously, each \( C_j \) is included into \( D \).

- The relation \( R \) between \( B \) and \( D \) is defined as follows: for every every \( b \) in \( B \) and every \( d \) in \( D \), \( b R d \) if there is some \( j \) such that \( b \in A_j \) and \( b R_j d \).

Moreover, we may see that, for every \( j : 1 \leq j \leq n \), the diagram in Fig. 7 commutes. Let \( a \in A_j \) and \( d \in D \) then \( a(i_j' \circ R_j)d \), where \( i_j' \) denotes the inclusion \( C_j \subseteq D \), if and only if \( a R_j d \). On the other hand, \( a(R' \circ i_j)d \), where \( i_j \) denotes the inclusion \( A_j \subseteq B \), if and only if there is a \( k : 1 \leq k \leq n \) such that \( a \in A_k \) and \( aR_kd \). However, since the family \( \{ R_1 \subseteq A_1 \times C_1, \ldots, R_n \subseteq A_n \times C_n \} \) is assumed to be parallel consistent with respect to \( \{ A_1 \subseteq B, \ldots, A_n \subseteq B \} \), this is equivalent to \( aR_kd \).

\[
\begin{array}{ccc}
A_j & \xrightarrow{R_j} & B \\
\downarrow & & \downarrow R \\
C_j & \xrightarrow{R} & D
\end{array}
\]

Figure 7. Parallel composition of inclusions and relations

It may be noted that there may be other reasonable ways of defining this parallel extension. The one above captures the intuition that we build \( D \) by replacing each \( A_j \) included in \( B \) by its transformation \( C_j \). In general, it may also be noted that our definition of parallel extension fails to satisfy the commuting properties of parallel extension, if the given relations are not parallel consistent with respect to the corresponding inclusions. Let us consider the following counterexample. Let us suppose that we have two sets \( A = \{ a_1, a_2 \} \) and \( A' = \{ a_1, a_3 \} \) included in \( B = \{ a_1, a_2, a_3 \} \), and let us suppose that we have two relations \( R \) and \( R' \) defined on \( A \) and \( C = \{ c_1, c_2, c_3 \} \), and on \( A' \) and \( C' = \{ c_1, c_3, c_4 \} \), respectively. Moreover let us suppose that \( R = \{ (a_1, c_1), (a_1, c_3), (a_2, c_2), (a_2, c_3) \} \) and \( R' = \{ (a_1, c_1), (a_1, c_4), (a_3, c_1) \} \). We may see that \( R \) and \( R' \) are not parallel consistent with respect to the inclusions of \( A \) and \( A' \) in \( B \). For instance, relation \( R \) transforms \( a_1 \) into \( c_1 \) and \( c_3 \), while relation \( R' \) transforms \( a_1 \) into \( c_1 \) and \( c_4 \). Now, in this context, the parallel extension \( D \) would be the set \( D = \{ c_1, c_2, c_3, c_4 \} \), which includes \( C \) and \( C' \). Moreover, the relation \( R'' \subseteq B \times D \) defined by the parallel extension would be:

\[
R'' = \{ (a_1, c_1), (a_1, c_3), (a_2, c_2), (a_2, c_3), (a_1, c_4), (a_3, c_1) \}
\]
However this relation does not satisfy the commuting properties of parallel extension. For instance, we have that, according to $R$, $a_1$ is related to $c_1$ and $c_3$. However, in the extension $R'' a_1$ is related to $c_1, c_3$, and $c_4$. Therefore the composition of the inclusion $A \subseteq B$ with the relation $R''$ is not the same as the composition of the relation $R$ with the inclusion $C \subseteq D$. The example becomes parallel consistent if we remove $\langle a_1, c_1 \rangle$ from $R$ and $\langle a_1, c_4 \rangle$ from $R'$, which implies that $\langle a_1, c_1 \rangle$ and $\langle a_1, c_4 \rangle$ are removed from $R''$. In this case, $R''$ would satisfy the commuting properties of parallel extension.

Let us now show that parallel consistency also satisfies conditions 2 - 4 of Definition 2.3:

2. Parallel extension diagrams are closed under vertical composition. Let us consider the vertical composition of two parallel extension diagrams shown in Fig. 8. We have to show, first, that the family of relations $\{R'_1 \circ R_1, \ldots, R'_n \circ R_n\}$ is parallel consistent with respect to the inclusions $\{A_1 \subseteq B, \ldots, A_n \subseteq B\}$, so that we can build the parallel extension diagram in Fig. 9. Then, we have to show that $C = C'$. Finally, we also have to prove that $R'' = R' \circ R$.

Now, suppose that $a \in A_1$ and $a \in A_j$ and $aR'_j \circ R_j c$. This is equivalent to say that there is a $b$ such that $aRb$ and $bR'_j c$. Since we assume that $\{R_1, \ldots, R_n\}$ is parallel consistent with respect to $\{A_1 \subseteq B, \ldots, A_n \subseteq B\}$ and $\{R'_1, \ldots, R'_n\}$ is parallel consistent with respect to $\{A'_1 \subseteq B', \ldots, A'_n \subseteq B'\}$, this is equivalent to $aRb$ and $bR'_j c$, which means $a(R'_j \circ R_j)c$. Therefore, $\{R'_1 \circ R_1, \ldots, R'_n \circ R_n\}$ is parallel consistent with respect to $\{A_1 \subseteq B, \ldots, A_n \subseteq B\}$.

Let us now prove that $C = C'$. According to our definition of parallel extension we have:

$$C = \bigcup_{1 \leq j \leq n} A'_j \cup (B \setminus (\bigcup_{1 \leq j \leq n} A'_j)) = \bigcup_{1 \leq j \leq n} A'_j \cup (B \setminus (\bigcup_{1 \leq j \leq n} A'_j)) = \bigcup_{1 \leq j \leq n} A'_j \cup (B \setminus (\bigcup_{1 \leq j \leq n} A'_j)) = C'. $$

Finally, let us show that $R'' = R' \circ R$. On one hand, we have that $b R'' c$ implies $\exists j : (1 \leq j \leq n) : b (R'_j \circ R_j) c$, meaning that there exists $j$ and $b' \in A'_j$ such that $b' R'_j \circ R_j b'$, i.e., $b (R' \circ R) c$. Conversely, if $b (R' \circ R) c$ this means exists $b' \in B'$ such that $b' R' \circ b R b'$ this means that there exist $j$ and $k$ such that $b' R'_j \circ b R_j b'$, $R_j b'$, Now, by parallel consistency, we have that $b' R'_j c$ and $b' \in A'_k$ implies $b' R'_k c$, i.e., $b' R'_k c \circ b R_k b'$. Therefore, $b R'' c$.

3. Extension of embeddings. If $n = 1$, in this case, parallel consistency relation trivially holds.
4. Extension of embeddings as parallel extension. Given the extension of inclusions in Fig. 10, we know that \( D = C_0 \cup (B \setminus A_0) \). On the other hand, we have that \( D' = (C_0 \cup \bigcup_{1 \leq j \leq n} A_j) \cup (B \setminus \bigcup_{0 \leq j \leq n} A_j) = C_0 \cup (B \setminus A_0) \), since each \( A_j \) is included in \( B \). Therefore \( D = D' \).

![Figure 10. Extension of inclusions](image1)

![Figure 11. Extension of inclusions as parallel extension](image2)

Summarizing, we can now provide a definition of a formalism supporting component architectural components and connectors (in short, a component framework):

**Definition 2.5. (Component Framework)** A component framework \( \mathcal{F} \) consists of:

- A class of models \( \mathcal{M} \), including bodies and ports of components and bodies and roles of connectors.
- A class of transformations \( \mathcal{T} \) and a class of embeddings \( \mathcal{E} \), with \( \mathcal{E} \subseteq \mathcal{T} \), such that they are closed under composition, in the sense that if \( t_1 : M_1 \Rightarrow M_2 \) and \( t_2 : M_2 \Rightarrow M_3 \) are in \( \mathcal{T} \) (respectively \( e_1 : M_1 \to M_2 \) and \( e_2 : M_2 \to M_3 \) are in \( \mathcal{E} \)) then their composition \( t_2 \circ t_1 \) is also in \( \mathcal{T} \) (respectively, \( e_2 \circ e_1 \) is in \( \mathcal{E} \)). Moreover, this composition is associative, and for every model \( M \) there is a special identity which is neutral with respect to embedding and transformation composition.
- A parallel consistency relation for \( J \)-indexed families of transformations with respect to \( J \)-indexed families of embeddings that satisfies the parallel extension property.

3. Architecture graphs and diagrams

In this section we study how we can build architectures by connecting components via connectors. The concepts that are presented are mainly syntactical. In particular, we describe under which conditions we can connect a set of components using a set of connectors and we provide graphical representations for these connections. However, the semantics of these constructions is only studied in the following section.

As said above, components are connected to a connector by linking its roles to compatible ports of the given components, where we consider that a role is compatible with a port if the port is a transformation of the role. However, not every possible connection is correct, even if each role is
compatible with the corresponding port. For instance, if two roles share some elements and these elements are refined differently in the corresponding components (i.e., if the associated transformations are not parallel consistent), then the connection should be considered incorrect. In this context, the following definition describes how we can connect a connector to a set of components and how we can graphically represent this connection:

**Definition 3.1. (Connection Diagram and Graph)** Given a connector \( CON = (B, r_1 : R_1 \rightarrow B, \ldots, r_n : R_n \rightarrow B) \), \( n \) components \( COMP_i = (B_i, p_{i1} : R_{i1} \Rightarrow B_i, \ldots, p_{im_i} : R_{im_i} \Rightarrow B_i) \), and \( n \) transformations \( con_i : R_i \Rightarrow P_{ki} \) with \( 1 \leq k_i \leq m_i \) then we can connect the set of components \( \{COMP_i\}_{i \in \{1,\ldots,n\}} \) by means of the connector \( CON \) via the transformations \( \{con_i\}_{i \in \{1,\ldots,n\}} \). This connection is represented by the connection diagram in Figure 12 and the (more abstract) connection graph in Figure 13.

![Figure 12. Connection diagram](image1)

![Figure 13. Connection graph](image2)

More precisely, the connection diagram is a labelled directed graph including two kinds of edges: embedding and transformation edges. The diagram includes subgraphs the graphical representation of the connector \( CON = (B, r_1 : R_1 \rightarrow B, \ldots, r_n : R_n \rightarrow B) \) and the \( n \) components \( COMP_i = (B_i, p_{i1} : R_{i1} \Rightarrow B_i, \ldots, p_{im_i} : R_{im_i} \Rightarrow B_i) \), and, finally, for every \( i \) there is a transformation edge, labelled by \( con_i \), from the node labelled by the role \( R_i \) to the node labelled by the port \( P_{ki} \). The associated connection graph is a labelled undirected graph consisting of one node per component \( COMP_i \), labelled by the component, and another node labelled by the connector \( CON \). In addition, for every \( i, 1 \leq i \leq n \) there is an edge between the node labelled by \( COMP_i \) and the node labelled by the connector. This edge is labelled by the transformation \( con_i \).

It may be noted that, according to the above definition, in a connection diagram every role of the given connector must be connected to a port of a given component. This means that we do not consider partial connections. There are two reasons for this. On one hand, we believe that hierarchical design and reasoning is possible if any given subsystem (i.e. the result of composing some components) can be seen itself as a component. On the other hand, we think that a main ingredient of this approach is the division between connectors and components, and the fact that components are always connected through connectors. In this context, when the connection is total (i.e. each role is bound to a port),
a connection diagram may be considered to denote a component (see Def. d.comp-compos below). However, if the connection would be partial then it would denote some kind of hybrid component including roles and ports.

The connection diagram is consistent if the family of transformations \( \{ p_{i_k} \circ \text{con}_{i_k} \}_{i \in \{1, \ldots, n\}} \) is parallel consistent with respect to the family of embeddings \( \{ r_i \}_{i \in \{1, \ldots, n\}} \).

These notions of connection graph and diagrams can be easily generalized to describe the interconnection of an arbitrary number of connectors and components, to form architectures. However, not any arbitrary form of connection is allowed. In this approach, architectures are built in a hierarchical way, although this may be not clearly displayed in the associated graphs or diagrams. In particular, after connecting a set of components by means of a connector, the result may be seen as a new component whose ports are the unused ports of the components involved. This is seen in detail in sections 4 and 5. Now, this new component may be connected with other components, by means of another connector, to build a new component, and so on. As said above, this hierarchical structure of an architecture is not necessarily explicit in a given diagram or graph. However, this constraint can be enforced by imposing that the associated graph has no cycles. We also assume that architecture diagrams and graphs are connected graphs. However, it would not be a problem to allow non-connectivity. The main difference would be found on the semantics of architectures (see the corresponding section below). In particular, we consider that the semantics of an architecture is a component. Not imposing the connectivity constraint would mean that the semantics of an architecture would be a set of components, one per each connected subgraph.

**Definition 3.2. (Architecture diagram and architecture graph)** An architecture diagram \( D \) over a finite set of components \( \text{COMPON} \) and a finite set of connectors \( \text{CONNEC} \) is the smallest connected labelled graph such that the following hold:

1. \( D \) includes the disjoint union of the graphical representations of the components in \( \text{COMPON} \) and the connectors in \( \text{CONNEC} \).

2. For each node labelled with a role in each connector subgraph there is one edge connecting it with a node labelled with a port in a component subgraph and for each node labelled with a port in a component subgraph there is at most one edge connecting it with a node labelled with a role in a connector subgraph. Moreover, these edges are labelled with a transformation of the role by the port, leading to connection diagrams in the sense of Def 3.1.

Its associated architecture graph \( G \) is the graph having a node for each connector and for each component and an edge linking a component and a connector, for each transformation linking a role from the latter to a port of the former. In addition, an architecture diagram to be considered consistent it is required that all connection diagrams are consistent and that the associated architecture graph has no cycles.

It may be noted that an architecture diagram can be obtained as the union of a set of connection diagrams. On the other hand, architecture graphs depict whole architectures clearer by abstracting from the direct interface connections and only revealing, which components are connected by which connectors.

**Definition 3.3. (Architecture)** An architecture \( A \) consists of a set of components \( \text{COMPON}_A \), a set of connectors \( \text{CONNEC}_A \), an architecture diagram \( D_A \) over \( \text{COMPON}_A \) and \( \text{CONNEC}_A \), and the corresponding architecture graph \( G_A \), such that \( G_A \) is cyclefree.
For example, Fig. 14 is an example of a (small) architecture diagram, which is union of the connection diagrams in figures 15, 16 and 17. Then, Fig. 18 represents its associated architecture graph. In this example, four components (with two ports each, except COMP₄ with one port) are connected using three connectors (with two roles each). The resulting architecture has one “free” port, which means that it could be connected to other components or architectures. It may be noted that, we could obtain the architecture depicted in Fig. 14, for instance, by composing the components COMP₁ and COMP₂ using the connector CON₁, composing the components COMP₃ and COMP₄ using the connector CON₃ and then composing the two results using CON₂. But it can also be constructed in a different way. For instance, composing the components COMP₁ and COMP₂ using the connector CON₁, composing the result and COMP₃ using the connector CON₂ and, finally, composing the result with COMP₄ using the connector CON₃.
4. Component composition

In this section we study the semantics of the operation of connecting a family of components by means of an architectural connector, which we call component composition. In particular, we define the semantics of this operation and then prove some of its properties. First, we see that component composition is associative. This is a key result for showing that the semantics of architectures, defined in the following section, is correct. Then we show that composition preserves transformations. This is needed to show, also in the following section, that if we replace a component $C$, in a given architecture $A$, by another component $C'$, where $C'$ is a transformation of $C$ (i.e. the body of $C'$ is a transformation of the body of $C$), then the resulting architecture $A'$ is a transformation of $A$.

**Definition 4.1.** (Component Composition) Given a connector $CON = (B, r_1 : R_1 \rightarrow B, \ldots, r_n : R_n \rightarrow B)$, $n$ components $COMP_i = (B_i, p_{i1} : R_{i1} \Rightarrow B_i, \ldots, p_{im_i} : R_{im_i} \Rightarrow B_i)$, and $n$ transformations $con_i : R_i \Rightarrow P_{ik_i}$ for $i \in \{1, \ldots, n\}$ such that the connection diagram in Fig. 19 is consistent, we define its composition as the component $COMP = (B', Ports)$, where $B'$ is the result of the parallel extension diagram in Fig. 20 and $Ports$ consists of all the transformations $r'_i \circ p_{ik_i}$ such that $1 \leq i \leq n, 1 \leq j_i \leq m_i$ and $j_i \neq k_i$.
Even if this shorthand notation may be ambiguous, the composition of a consistent diagram like the above one is denoted by

$$\text{COMP} = \text{CON}(\{(\text{COMP}_j)_{j \in J}, (\text{con}_j)_{j \in J}\})$$

or just

$$\text{COMP} = \text{CON}(\text{COMP}_j)_{j \in J}$$

if the transformations $$(\text{con}_j)_{j \in J}$$ can be considered as implicit.

The next theorem states that the result of two overlapping connections via two connectors is independent of the order to compute the single compositions. In particular, this means that composition is associative.

**Theorem 4.1.** Given a consistent architecture $A$ with architecture graph $G_A$ as in Figure 21, then the following expressions yield the same component:

1. $$(E1) \quad \text{CON}(\{(\text{COMP}_j)_{j \in \{1, \ldots, n-1\}}, (\text{CON}')_{(\text{COMP}^\prime)_k \in \{2, \ldots, m\}}\})$$
2. $$(E2) \quad \text{CON}'(\text{CON}(\{(\text{COMP}_j)_{j \in \{1, \ldots, n\}}, (\text{COMP}^\prime)_k \in \{2, \ldots, m\}\}))$$

**Proof:**

Let us consider the architecture diagram $D_A$ in Figure 22 corresponding to the architecture graph $G_A$ given above. Note that we do not display unused ports in this diagram (i.e. the ports which are not connected to any role), since they are not relevant for the proof.
Now, computing the compositions according to the expression \((E1)\), is displayed in the diagram below (Fig. 23). It must be noted that after having computed the composition \(CON'((COMP'_k)_{k \in \{1, \ldots, m\}})\) yielding a component \(COMP\) we can still compute the composition \(CON((COMP_j)_{j \in \{1, \ldots, n-1\}}, COMP)\) as a consequence of conditions 2., 3., and 4. of Definition 2.4. In particular, conditions 3. and 4. ensure that the family of transformations \(\{id_1 : B_1 \Rightarrow B_1, \ldots, id_{n-1} : B_{n-1} \Rightarrow B_{n-1}, e : B_n \Rightarrow B_n\}\), where \(e : B_n \rightarrow B_{res}\) is the embedding that is displayed in Fig. 23, is parallel consistent with respect to the embeddings \(\{e_1, \ldots, e_{n-1}\}\) in Fig. 24, and by the vertical composition of parallel extensions, transformations \(\{p_1 \circ con_1, \ldots, e \circ p_n \circ con_n\}\) are also parallel consistent with the embeddings \(\{r_j\}_{j \in \{1, \ldots, n-1\}}\).

Similarly, computing the compositions according to the expression \((E2)\), is displayed in the diagram below (Fig. 24).

Now, we have to prove that the resulting bodies, \(B1'_{res}\) and \(B2'_{res}\) (and also the corresponding arrows), coincide. In order to do so, let us consider, in Fig. 25, a third way of computing this stepwise composition.

In this case, we have computed the two compositions separately and, then, we have built the result
by means of the extension diagram (9) (one should remember that embeddings are just a special kind of transformations). Now, we may see that the results of the three diagrams coincide. In particular, the parallel extension diagram consisting of subdiagrams (1) and (2) in Fig. 23 is just the vertical composition of the parallel extension diagram consisting of subdiagrams (5) and (6) and the extension diagram (9) in Fig. 25, which can be seen as a parallel extension diagram in the sense of Def. 2.4 Fig. 6, where \( B_1, \ldots, B_{n-1} \) are bound by the identity transformation. Hence, uniqueness of parallel extension implies \( B_1'_{\text{res}} = B_3'_{\text{res}} \). Similarly, the parallel extension diagram consisting of subdiagrams (3) and (4) in Fig. 24 is just the vertical composition of the parallel extension diagram consisting of subdiagrams (7) and (8) and the extension diagram (9) in Fig. 25. This implies \( B_2'_{\text{res}} = B_3'_{\text{res}} \), and together with \( B_1'_{\text{res}} = B_3'_{\text{res}} \) we have \( B_1'_{\text{res}} = B_2'_{\text{res}} \).

Finally, we prove that composition is compatible with transformations, which means that composition is compatible with refinement in the standard case where transformations are considered to be some form of refinement. We consider that a component \( \text{COMP}' \) is a transformation of another component \( \text{COMP} \) if they share the same interfaces and there is a transformation from the body of \( \text{COMP} \) into the body of \( \text{COMP}' \). More precisely:

**Definition 4.2. (Component transformation)** Given components \( \text{COMP} = (B, p_1 : P_1 \Rightarrow B, \ldots, p_n : P_n \Rightarrow B) \) and \( \text{COMP}' = (B', p'_1 : P_1 \Rightarrow B', \ldots, p'_n : P_n \Rightarrow B') \), we say that \( \text{COMP}' \) is a transformation of \( \text{COMP} \) if there is a transformation \( t : B \Rightarrow B' \) such that for every \( i, 1 \leq i \leq n \), we have that \( p'_i = t \circ p_i \).

Now, the fact that composition preserves transformations means that if, on one hand, we compose a component \( \text{COMP}_1 \) together with other \( n - 1 \) components using a connector \( \text{CON} \) yielding as result the component \( \text{COMP} \) and, on the other hand, we compose another component \( \text{COMP}'_1 \) together with the same \( n - 1 \) components using the same connector \( \text{CON} \) yielding as result the component \( \text{COMP}' \) and if \( \text{COMP}'_1 \) is a transformation of \( \text{COMP}_1 \) then \( \text{COMP}' \) is a transformation of \( \text{COMP} \).

**Theorem 4.2. (Compatibility of transformation and composition)** Given two components \( \text{COMP} = \text{CON}((\text{COMP}_j)_{j \in \{1..n\}}, (\text{CON}_j)_{j \in \{1..n\}}) \), with parallel extension \( t_0 : B \Rightarrow B' \) and induced embeddings
\{r'_j\}_{j \in \{1, n\}}\) according to Fig. 27, and \(COMP' = CON(COMP'_1, (COMP_j)_{j \in \{2, n\}}, (CON_j)_{j \in \{1, n\}})\), if \(COMP'_1\) is a transformation of \(COMP_1\) via \(t : B_1 \Rightarrow B'_1\), such that the transformations \(\{r'_j\}_{j \in \{1, n\}}\) are parallel consistent with respect to \(\{r_j\}_{j \in \{1, n\}}\), where \(t_1 = t\) and \(t_j = id_{B_j}\) for \(j \neq 1\), then \(COMP'\) is a transformation of \(COMP\).

**Proof:**

It is enough to notice that the parallel extension diagram in Fig. 26 defining \(COMP'\) is the vertical composition of the parallel extension diagrams in Fig. 27, whose upper part defines \(COMP\) and whose lower part corresponds to the transformation \(COMP'_1\) of \(COMP_1\), with \(p'_1 = t \circ p_1\) (\(j = 1, ..., m_1\)). Uniqueness of vertical composition implies \(B'' = B'''\) and \(r''_1 = r'''_1 \ldots r''_n = r'''_n\). Then \(t'\) defines the transformation of \(COMP\) into \(COMP'\).
5. Semantics of Architectures

In this section, based on the semantics of composition defined in the previous section, we define the semantics of connector architectures. As discussed above, we show that the semantics of an architecture is a component (if we would allow architecture diagrams to be non-connected graphs, then the semantics would be a set of components). This means that in a bottom-up design process one can abstract a complete subsystem, defined by an arbitrary large diagram, considering it as a single component. Or, conversely, in a top-down design process, one can consider that a certain part of a system is a component and, later, refine it into a whole architecture. This semantics is defined by “computing” all the connections which are present in the diagram. This form of computation is described using (algebraic) graph transformation reduction rules, where the application of a rule corresponds to computing a composition operation. Obviously, at a given moment there may be many possible reductions to apply. However, we show that the reduction relation is finitely terminating and Church-Rosser, implying the well-definedness of the semantics. Obviously, the associativity of composition, proved in Theorem 4.1, plays an important role in this proof. To end the section, we show that the semantics of architectures preserves transformations, as a generalization of Theorem 4.2 in the previous section.

In the algebraic approach to graph transformation, more precisely in the Double Pushout approach (see [14, 9]), transformation rules have the form \( p = (L \leftarrow K \rightarrow R) \). Intuitively, \( L \) (the left-hand side) defines the pattern that we want to transform, \( K \) (the context), included in \( L \), is the part of that pattern that is preserved by the rule. This means that \( L \setminus K \) is the part of the pattern that is deleted by the rule. Finally, the right-hand side, \( R \), also including \( K \), describes what is added by the rule. Technically, a derivation or transformation step in this approach is given by two pushout diagrams (1) and (2) in Figure 28, written \( G \Rightarrow H \) via \( (p, m) \), where \( m : L \rightarrow G \) is a graph morphism, that represents the match of \( L \) in \( G \). Intuitively, a derivation can be explained as follows: first, we remove \( h(L \setminus K) \) from

\[
\begin{array}{ccc}
R_1 & \cdots & R_n \\
\downarrow_{r_1} & & \downarrow_{r_n} \\
B & & B \\
\downarrow_{r_n} & & \downarrow_{r_n} \\
P_{11} & \cdots & P_{nk} \\
\downarrow_{p_{11}} & & \downarrow_{p_{nk}} \\
B_1 & \cdots & B_n \\
\downarrow_{p_{nk}} & & \downarrow_{p_{nk}} \\
P_{1m} & \cdots & P_{nm} \\
\downarrow_{p_{1m}} & & \downarrow_{p_{nm}} \\
B_1' & \cdots & B_n' \\
\downarrow_{p_{nm}} & & \downarrow_{p_{nm}} \\
P_{1m'} & \cdots & P_{nm'} \\
\downarrow_{p_{1m'}} & & \downarrow_{p_{nm'}} \\
B_1'' & \cdots & B_n'' \\
\downarrow_{p_{nm'}} & & \downarrow_{p_{nm'}} \\
\end{array}
\]

Figure 27. Composition and transformation
Given a consistent architecture diagram \( A \) with architecture diagram \( D_A \), for each connector \( \text{CON} = (B, r_1 : R_1 \to B, ..., r_n : R_n \to B) \) in the diagram and for all possible components \( \text{COMP}_i = (B_i, p_{i1} : R_{i1} \Rightarrow B_i, ..., p_{in_i} : R_{in_i} \Rightarrow B_i) \in \{1, ..., n\} \) and transformations \( (\text{con}_1, ..., \text{con}_n) \) we have a diagram reduction rule

\[
\text{CON}((\text{COMP}_j)_{j \in \{1, ..., n\}}, (\text{con}_j)_{j \in \{1, ..., n\}})
\]

as depicted in Figure 29, where \( B' \) and \( p'_{ij} = r'_i \circ p_{ij} \) is defined by the composition:

\[
\text{COMP} = \text{CON}((\text{COMP}_1, ..., \text{COMP}_n, \text{con}_1, ..., \text{con}_n) = (B', (p'_{ij} : P_{ij} \Rightarrow B')_{(i,j) \in I \times J})
\]

for each \( (i, j) \in \{1, ..., n\} \times (\{1, ..., m\} \setminus \{k\}) = I \times J \).

**Remark:** It may be noted that, for our purposes it is enough to consider only graph inclusions as matching morphisms, i.e. there is no need to consider more general kinds of graph morphisms.

We can show (see below the Architecture Reduction Lemma) that a diagram reduction rule, as the one depicted in Fig. 29, reduces an architecture diagram \( D_A \) to a smaller and well-defined architecture \( D'_A \) as defined above. \( D'_A \) is smaller than \( D_A \) in the following sense: If \( D_A \) includes \( k \) components and \( l \) connectors then \( D'_A \) includes \( (k - n + 1 \) components and \( l - 1 \) connectors.

**Lemma 5.1. (Architecture Reduction)**

Given a consistent architecture diagram \( D_A \) over a set of \( k \) components, \( \text{COMPON} \), a set of \( l \) connectors, \( \text{CONNEC} \), and the diagram reduction rule \( \text{CON}((\text{COMP}_j)_{j \in \{1, ..., n\}}, (\text{con}_j)_{j \in \{1, ..., n\}}) \) depicted in Fig. 29, if the left-hand side of the rule is included in \( D_A \) then we can derive a new architecture diagram \( D'_{A'} \), denoted \( D_A \Rightarrow D'_{A'} \), where \( D'_{A'} \) is a new architecture diagram with \( k - n + 1 \) components and \( l - 1 \) connectors.
Proof:
According to definition 3.1, $D_A$ can be obtained as the union of the graphical representations of the components in $COMPON$ and of the connectors $CONNEC$, together with edges labelled with transformations linking all the role nodes with some port node. In addition, it must be connected and its associated graph must be acyclic. Let us check these properties for the result of the transformation:

- The rule replaces the graphical representation of $n$ components and one connector by the rule. So, $D_A'$ includes $k - n + 1$ components and $l - 1$ connectors. In addition, every role in the resulting diagram was already present in $D_A$, which means that its associated node was connected to a port with a role node. Now, this rule cannot be one of the nodes deleted by the rule, because this would have implied that the architecture graph associated to $D_A$ would have included a cycle. Therefore, the role node would still be connected to a port node. On the other hand, if a port node in $D_A'$ is connected by an edge to a role node, then this edge was also present in $D_A$. This means that every port node in $D_A'$ is connected with at most one role node $D_A'$.

- Suppose that $D_A'$ is not connected. This means that $D_A'$ includes other components or connectors in addition to the component in the result of the rule application. If there are only additional components (no additional connectors), then $D_A$ would have been connected if some of the role nodes in the connector, which has been deleted by the rule, were connected to some port nodes of these components. However, this is impossible, since a role node is connected with a single edge to a component, and the transformation rule includes already such an edge to a different component. Suppose that there are some additional connectors, if $D_A$ was connected, this means that some role nodes in these connectors were connected to port nodes which have been deleted by the rule. However, this is again impossible, because this would mean that, in $D_A$, these port nodes were connected to more than one role node.

- The associated architecture graph $G_A$ has no cycles: we know that a connected graph has no cycles if the number of nodes minus the number of edges is equal to one. Now, we know that

$$\#\text{nodes}(G_A') = \#\text{nodes}(G_A) - n \quad \text{and} \quad \#\text{edges}(G_A') = \#\text{edges}(G_A) - n$$

since in this graph we have one node per each component or connector in the diagram and one edge per each connection between a port node and a role node in the diagram. This means that

$$\#\text{nodes}(G_A') - \#\text{edges}(G_A') = \#\text{nodes}(G_A) - \#\text{edges}(G_A) = 1.$$
• Finally, we can see that the resulting architecture is also consistent. In particular, for any connection subdiagram in $D_A'$ we have two possible situations. Either the connection subdiagram does not involve any of the unused ports $P_{ij}$ of a component $COMP_i$ involved in the reduction rule, or the connection subdiagram involves such port. In the former case, the application of the transformation rule does not affect the subdiagram. Therefore if the subdiagram is consistent before applying the transformation rule then it will still be consistent after the transformation. In the latter case, the transformation rule replaces the transformation $p_{ij}$ binding the port $P_{ij}$ to the body $B_i$ by $p'_{ij}$ binding the port $P_{ij}$ to the body $B_i$, where $p'_{ij} = r'_{ij} \circ p_{ij}$ and $r'_{ij}$. Now if the subdiagram involving $P_{ij}$ is consistent before the application of the transformation rule, then it will still be consistent after the transformation because the consistency relation satisfies the embedding condition.

The lemma above has two important consequences. The first one is that the reduction relation is finitely terminating:

**Corollary 5.1. (Termination of Architecture Reduction)**

Given a consistent architecture diagram $D_{A_i}$ over a set of $k$ components and a set of $l$ connectors, any reduction sequence of $D_{A_i}, D_{A_i} \Rightarrow \ldots \Rightarrow D_{A_n}$ is at most of length $l$, i.e. $n \leq l$.

**Proof:**

According to the previous lemma, at every reduction step we have one connector less in the architecture diagram. Therefore, the length of any reduction sequence is bounded by the number of connectors in the diagram.

The second one refers to the normal forms of the reduction relation:

**Corollary 5.2.** Given an architecture diagram $D_A$, if $D_{A'}$ is a normal form of $D_A$, i.e. $D_A \Rightarrow^* D_{A'}$ and $D_{A'}$ is irreducible, then $D_{A'}$ is the graphical representation of a component.

**Proof:**

According to the lemma above, applying a reduction step to a diagram $D_A$ yields a well-formed diagram. Therefore, by induction we know that $D_{A'}$ is a well-formed architecture diagram. Now, $D_{A'}$ cannot include a connector, since being well-formed, it would include a connection subdiagram and it would be reducible. On the other hand, $D_{A'}$ cannot include two or more components since, not including any connector, the diagram would not be connected. Finally, $D_{A'}$ must include, at least, one component, since the right-hand side of a reduction rule is a component.

Finally, before defining the semantics of architectures, we can also prove that the reduction relation satisfies the Church-Rosser property and, therefore, is uniquely normalizing. Actually, since the reduction relation is finitely terminating, it is enough to show that the reduction relation is locally Church-Rosser:

**Theorem 5.1. (Local Church-Rosser property of reduction sequences)**

Given a consistent architecture diagram $D_{A_i}$ with reduction steps $CON_i : D_{A_i} \Rightarrow D_{A_i}$, for $i = 1, 2$, there is an architecture diagram $D_{A_3}$ and there are reduction steps $CON_3 : D_{A_1} \Rightarrow D_{A_3}$ and $CON_4 : D_{A_2} \Rightarrow D_{A_3}$ so that the diagram below is commutative.
**Proof:**
If the matches of \(CON_1\) and \(CON_2\) in \(D_{A_0}\) are disjoint then the reduction steps are independent and we can choose \(CON_3\) and \(CON_4\) to be, respectively, the same reductions as \(CON_2\) and \(CON_1\), using the local Church-Rosser theorem for graph transformations [10]. If the matches are not disjoint we have an overlap of two different connection diagrams corresponding to two different connectors. Moreover, in this case, we know that the two connection diagrams can only overlap in one component subdiagram. The reason is that, if they overlapped in at least two component diagrams, then the corresponding graphs would overlap in at least two component nodes leading to a subgraph of the architecture graph \(G_{A_0}\) as depicted in Figure 31. However, this would mean that the architecture graph is not acyclic.

Therefore, the situation corresponds to the one shown in Figure 21 and Figure 22. Therefore, according to Theorem 4.1, the final result is independent of the order of the reductions. \(\square\)

Finally, we can define the semantics of a connector architecture as the result of the application of as many reduction steps as possible since, after all the results above, we know that this result exists and is unique and well-formed.

**Definition 5.2. (Architecture semantics)**
The semantics of a consistent architecture \(A\) is the component \(COMP\) obtained as the (unique) normal form of the architecture diagram \(D_A\)

To end this section we show that the semantics of architectures preserves transformations. This means that given an architecture \(A\), whose semantics is the component \(COMP\) and given a component \(COMP'_0\), which is a transformation of a component \(COMP_0\) in \(A\). If we replace \(COMP_0\) in \(A\) by \(COMP'_0\), leading to a new architecture \(A'\), then the semantics of \(A'\) is a transformation of \(COMP\).

**Theorem 5.2.** Given consistent architectures \(A\) and \(A'\) and given components \(COMP_0 = (B, p_1 : P_1 \Rightarrow B_0, ..., p_n : P_n \Rightarrow B)\) in \(COMPON_A\) and \(COMP'_0 = (B', p'_1 : P_1 \Rightarrow B'_0, ..., p'_n : P_n \Rightarrow B')\) in \(COMPON_A'\), such that \(D_A\) and \(D_A'\) are identical except that we have replaced in \(D_A\) the subdiagram associated to \(COMP_0\) by the diagram associated to \(COMP'_0\). Let \(COMP\) and \(COMP'\) be, respectively, the semantics of \(A\) and \(A'\). If \(COMP'_0\) is a transformation of \(COMP_0\) then \(COMP'\) is a transformation of \(COMP\).
Proof:
We proceed by induction on the length of the reduction $D_A \xrightarrow{\ast} D_{COMP}$. If the length of the reduction is zero, this means that $D_A = D_{COMP}$ and $D_A' = D_{COMP}'$, hence the theorem trivially holds.

Let us suppose that $D_A \Rightarrow D_{A_1} \xrightarrow{\ast} D_{COMP}$. We have two cases. If the reduction step $D_A \Rightarrow D_{A_1}$ does not involve the component $COMP_0$ then we can apply the same reduction on $D_{A_1}'$, $D_{A_1}' \Rightarrow D_{A_1}'$, where $D_{A_1}$ and $D_{A_1}'$ are identical except that we have replaced in $D_{A_1}$ the subdiagram associated to $COMP_0$ by the diagram associated to $COMP_0'$. Then, by induction, we know that $COMP'$ is a transformation of $COMP$.

Finally, let us assume that the reduction step $D_A \Rightarrow D_{A_1}$ involves the component $COMP_0$, i.e. we have transformed $D_A$ using the rule in Fig. 32:

![Figure 32. Diagram reduction rule](image)

Then, we can transform diagram $D_{A_1}'$ using the rule in Fig. 33:

![Figure 33. Diagram reduction rule](image)
Now, according to theorem 4.2, $COMP'_0$, defined in Figure 33 is a transformation of $COMP_0^*$, defined in Figure 32. This means that the resulting diagrams after these transformations are identical except that we have replaced in $D_A$ the subdiagram associated to $COMP_0^*$ by the diagram associated to $COMP_0^*$. Then, by induction, we know that $COMP'$ is a transformation of $COMP$. \[ \square \]

As a consequence of the previous results, we have shown that the semantics of architectures, as given in Def. 5.2, is well-defined:

**Theorem 5.3.** The semantics of a consistent architecture $A$ exists and is unique.

### 6. Related work

As said in the introduction, many Architecture Description Languages (ADL) or related concepts have been proposed for the specification of components and their interconnection (e.g., [22, 26, 37, 27, 1, 39, 16, 28, 4, 3, 30, 2, 6]). However, in many cases, these languages concentrate on the description of the coordination of components or the overall behavior of an architecture, with the aim of analyzing properties like deadlock-freeness or to be able to evaluate its performance. In these cases, little attention is paid to components and their functionality or to the compatibility of the interfaces of components which are connected. Essentially, in these approaches components are viewed as objects or processes and their interfaces are, often, just a communication channel. For instance, in CHAM [22] architectures are described as abstract machines, whose transitions are described as chemical reactions. This is used as a basis for the analysis of certain behavioral properties. In Darwin [27] architectures are defined in terms of the $\pi$-calculus: essentially components may be seen as $\pi$-calculus where connecting ports are just communication points. A similar approach, also based in the $\pi$-calculus is $\pi$-ADL [30]. Another approach which is similar to the previous ones [4], where the basis of the language is not the $\pi$-calculus, but a different process algebra. In all these cases, the aim is to use this kind of architectural descriptions for the analysis of behavioral properties. A similar approach with a different aim is AEMILIA [3], an extension of PADL, which is based on stochastic process algebra. In this case, the aim of the language is to allow for the analysis of the performance of architectures. An approach that we can consider in this group, but which is technically quite different is Reo. In this case, the language is in a way independent of the components, defining their composition and providing exogenous coordination.

Other approaches are somewhat close to implementation, in the sense that their basic constructions can be easily implementable. In these cases, the interfaces are typically a signature, or just an operation, and compatibility is essentially type checking. This is the case of UniCon, Rapide and C2SADL [26, 37, 28].

Some approaches that are, in a sense, closer to our work are [39, 6]. None of them presents an ADL, but they are concerned with the modeling and interconnection of components. In particular, the approach of Zaremski and Wing [39] does not consider any specific notion of component, but concentrate on the study of their interconnection. In particular they assume that the interfaces are described using the LARCH specification language and study their compatibility in this context. On the other hand, in the approach of Bidoit and Hennicker [6] a notion of component is introduced using Abstract State Machines as a modeling language. They study the compatibility of provided and required interfaces and define a semantics for the composition of components. Unfortunately, their approach is limited to components having just one provided interface and one required interface, which means that the only architecture that one would be able to build would consist of a sequence of components.
There are two approaches that are very related with our work. On one hand, the work of Allen and Garlan in connection to the Wright language [1], which we discuss in different parts of this paper, especially in its second part. On the other hand, the work of Fiadeiro, Lopez and Wermelinger around the language COMMUNITY [16, 25]. This approach is also inspired in [1] but instead of using CSP to describe components and connectors they use UNITY, and the semantics is given in terms of Category Theory. In addition, Fiadeiro, Lopez and Wermelinger in [18, 25] provide also a general framework, called Architectural Schools, that generalizes COMMUNITY [16], with similar aims to our framework. We believe that our approach is simpler and more general than architectural schools. This is partly motivated by the different granularity of both approaches. In our approach, we consider a class (a category) of models or specifications to define all our notions. The morphisms in this category are the transformations, where the embeddings are special cases of transformations. In the case of the architectural schools three categories are considered (together with some functors to relate them). On the one hand, we have the category of descriptions that is used to define components and connectors. For instance, in their case, an architectural connector is a diagram where, what we call the body, would be defined as the colimit of the diagram. We think that assuming that the bodies of connectors or components have to be built by means of a colimit of some diagram of specifications (descriptions) is not always justified. We may be writing our specifications using some given specification language where we may already have operations for defining a specification (the body) extending some other specifications (the interfaces), without having to use explicitly a colimit construction. The second category considered in architectural schools is a category of signatures. More precisely, to separate coordination from functionality, it is considered that a functor \( \text{sig} \) associates to each description its signature (its functionality). We do not consider this separation in our framework, since it would be useless: as we say in the introduction, in our approach we do not deal with coordination properties. Finally, the third category considered in architectural schools is the category of refinements. This category has the same objects as the category of descriptions, but has refinements as morphisms, which are used for connecting ports to roles and, in general, for refining descriptions. So, the category of refinements would correspond to our categories of models, but without considering the special case of embeddings.

Another reason in favour of the simplicity and generality of our framework is related to the categorical approach followed in the formulation of the Architectural Schools. More precisely, in that framework the categories involved are assumed to be (finitely) cocomplete. Then, the basic construction to define the semantics of a system is the colimit of the diagram describing the system. In that context, the fact that the semantics is uniquely determined by the specifications involved and their connections is just a consequence of the universal property of colimits. Instead, we preferred not to assume a priori that our formalisms had colimits and we tried to find out some minimal requirements to ensure the properties of our semantics. Actually, we think that defining parallel extensions and checking their properties is, in general, simpler than proving the existence of colimits because the universal property of colimits may be involved to prove. Moreover, as a consequence of dealing with different categories for descriptions and refinements, the existence of colimits in both categories is not enough to prove that the semantics of architectural systems is compatible with refinement. So this is directly required by architectural schools.

In [23] a similar generic component framework is presented which does not focus on architectural connectors and component transformation but rather on component specifications with several require and provide interfaces and the reduction of corresponding architectures. The corresponding instantiations include UML models, elementary Petri nets and adhesive high-level replacement categories [9]. These instantiations efforts could be carried over to the approach presented in this paper easily.
7. Conclusion

In this paper a generic concept for the specification of component connector architectures is introduced. This includes the definition of generic notions for components, connectors, corresponding connecting transformations and the calculation of the composition of several components along a coordinating connector. It is shown that the result of two such calculations is independent of the calculation order, even if both connectors involved are linked to the same component. Moreover, a generic notion of component transformation is introduced, which is proven to be compatible with composition. Architectures are then given by sets of components and connectors. Using the previous results it is shown that well-formed architectures can be uniquely reduced to an architecture semantics, which is a component again, and that component transformation is compatible with the semantics of architectures. Moreover, in the continuation of this paper the generic concept is instantiated to the sample specification techniques of CSP and of marked elementary nets, which is also used for a small case study modelling a door security system.

The present work features several points for possible extensions. In our opinion, the most interesting one would be to provide deductive methods to reason about architectural systems in our framework. In [35] we provide a very preliminary approach along these lines.

Acknowledgements: This work is partially supported by the Spanish project GRAMMARS (TIN2004-07925-C03-01) supported by FEDER, and by the CIRIT Grup de Recerca Consolidat 2001SGR 00254

References


