Electronic Notes in Theoretical Computer Science 108 (2004) 53–67 

[www.elsevier.com/locate/entcs](http://www.elsevier.com/locate/entcs)

A Generic Framework for Connector Architectures based on Components and Transformations

# H. Ehrig, J. Padberg, B. Braatz, M. Klein[1](#_bookmark0)

*Institute for Communication and Software Technology* *Technical University Berlin, Germany*

# F. Orejas, S. Perez, E. Pino[2](#_bookmark0)

*Departament de Llenguatges i Sistemes Informa`tics, Universitat Polit´ecnica de Catalunya, Barcelona, Spain*

Abstract

The intention of this paper is to extend our generic component framework presented at FASE 2002 [[4](#_bookmark28)] to a specific kind of connector architectures similar to architectural connections in the sense of Allen and Garlan [[1](#_bookmark26)]. In our generic component framework we have considered compo- nents with explicit import, export and body parts connected by embeddings and transformations and composition of components with a compositional transformation semantics. Our framework, however, was restricted to components with a single import and export interface. Here we study architectures based on connectors with multiple imports and components with multiple exports. Architectures studied in this paper are built up from components and connectors in a noncircular way. The semantics of an architecture is defined by reduction step sequences in the sense of graph reductions. The main result shows existence and uniqueness of the semantics of an architecture as a normal form of reduction step sequences. Our generic framework is instantiated on one hand to connector architectures based on CSP as the formal specification technique in the approach by Allen and Garlan. On the other hand it is instantiated to connector architectures based on high-level-replacement systems in general and Petri nets in particular. A running example using Petri nets as modeling technique illustrates all concepts and results.

*Keywords:* Components, connector architecture, Petri net transformations

1 Email:{ehrig,padberg,bbraatz,klein}@cs.tu-berlin.de

2 Email:{orejas,sperezl,pino}@lsi.upc.es

1571-0661 © 2004 Elsevier B.V. Open access under [CC BY-NC-ND license.](http://creativecommons.org/licenses/by-nc-nd/3.0/)

doi:10.1016/j.entcs.2004.01.012

# Introduction

The importance of architecture descriptions has become most obvious over the last decade (see e.g. [[17](#_bookmark37),[18](#_bookmark38),[7](#_bookmark31),[10](#_bookmark34),[6](#_bookmark30)]). Various formalisms have been proposed to deal with the complexity of large software systems. In order to build up large software systems from smaller parts, a flexible component concept for soft- ware systems and infrastructures are a useful and widely accepted abstraction mechanism (see e.g. [[19](#_bookmark40),[12](#_bookmark36),[8](#_bookmark32)]). Although there are many approaches available, only few are general enough to be used for different specification techniques. To achieve a generic concept the focus has to be on the fundamental issues of components and component-based systems. These are the interfaces, the compositionality of components and its embedding into the environment.

In our FASE 2002 paper [[4](#_bookmark28)] we have presented a generic component framework for system modeling that can be used for a large class of semi-formal and for- mal modeling techniques. According to this concept a component consists of a body, an import, and an export interface, and connections between import and body as well as export and body. These connections are again generic to allow a great variety of instantiations. We only require having suitable notions of embeddings, called inclusions in [[4](#_bookmark28)], and transformations (e.g. refinements) between specifications, such that the import connection of a component defines an embedding and the export connection a transformation. The connection between import and export interfaces of different components are also repre- sented by transformations. In fact, one of the key concepts of our framework is a generic notion of transformations of specifications, especially motivated by - but not limited to - rule-based transformations in the sense of graph transformation and high-level replacement systems [[5](#_bookmark29),[2](#_bookmark27),[16](#_bookmark39)]. In this paper we extend our generic component framework discussed above to a specific kind of connector architectures motivated by architectural connections in the sense of Allen and Garlan [[1](#_bookmark26)]. In fact, our generic framework in [[4](#_bookmark28)] is restricted to components with a single import and export interface. We consider architec- tures using connectors with multiple imports and components with multiple exports that allow connecting one connector to several different components. The key concept for the corresponding composition of a connector with com- ponents is a parallel extension diagram for transformations. This generalizes the notion of an extension diagram in [[4](#_bookmark28)] which is the key concept to define transformation semantics for components and to prove compositionality. Ar- chitectures studied in this paper are built up from components and connectors in a noncircular way. The semantics of an architecture is defined by reduction step sequences, where in each reduction step one connector is composed with all adjacent components. The main result shows existence and uniqueness of the semantics as a normal form of reduction step sequences.

This paper continues with the construction of the connector architecture in Section [2](#_bookmark1). In this and the following sections we have an ongoing example using Petri nets. In Section [3](#_bookmark10) we show the composition of components with connectors. Subsequently we show the existence of a unique semantics for architectures in Section [4](#_bookmark18). In Section [5](#_bookmark24) we give a concrete instantiation to CSP and a more abstract instantiation to high-level replacement systems including Petri nets as a concrete case. In Section [6](#_bookmark25) we conclude with a brief discussion of related work and an outlook to future research.

# Construction of Connector Architectures

In this section we present the main syntactical concepts of our general frame- work for connector architectures. The framework in [[4](#_bookmark28)] as well as the frame- work in this paper is generic not only concerning the underlying concept of semi-formal or formal specifications, but also concerning the concept of transformations in order to model abstraction and refinement between inter- faces and body of one component, or between import and export interfaces between different components. In this section we only require that trans- formations are closed under composition and that we have a special kind of transformations, called embeddings, which intuitively model inclusion of specifications, and a notion of independence of embeddings explained below. Motivated by architectural connections in the sense of Allan and Garlan [[1](#_bookmark26)] we distinguish in this paper components and connectors with multiple in- terfaces, while we have considered only components with single interfaces in [[4](#_bookmark28)]. Now a component *COMP* = (*B, e*1 : *E*1 =⇒ *B, ..., en* : *En* =⇒ *B*) for *n* ≥ 0 is given by the body *B* and a family of export interfaces *Ei* with export transformations *ei* : *Ei* =⇒ *B* for *i* ∈ {1*, ..., n*}. A connector *CON* = (*B, b*1 : *I*1 → *B, ..., bn* : *In* → *B*) for *n* ≥ 2 is given by the body *B* and a family of import interfaces *Ii* with body embeddings *bi* : *Ii* → *B* for *i* ∈ {1*, ..., n*}. We assume that the family of embeddings *bi* : *Ii* → *B* for each connector is independent. This means intuitively that the import interfaces *Ii* of *B* are pairwise disjoint. Now we can define formally how a connector con- nects different components. Given a connector *CON* = (*B, b*1*, ..., bn*) of arity *n*, and *n* components *COMPi* = (*Bi, ei*1 *, ..., eim* ) of arity *mi* with connector

*i*

transformations *coni* : *Ii* =⇒ *Eik* with 1 ≤ *k* ≤ *mi* for *i* ∈ {1*, ..., n*} then we

obtain the connector diagram in Figure [1](#_bookmark2) and the connector graph in Figure [2](#_bookmark2):

*I*  *bi*  *B*

*i*

*coni*

c*z*

*i*∈{1*,...,n*}

*.CON¸¸*

*Eik*

*¸¸*

*eik*

*e* c*z*

*con.*1*......*

\_*s*

*....*

*...*

*¸¸¸¸conn*

*¸¸¸* z*\_*

*Eij*

*i*

*ij* z*B*

*j*∈{1*,...,mi*}\{*k*}

*COMP*1

*COMPn*

Fig. 1. Connector Diagram

Remarks:

Fig. 2. Connector graph

* A connector diagram coΣnsists of *n* import interface nodes *Ii* of *n* +1 body

nodes *Bi* and *B*, and of

*n*

*i*=1

*mi* export interface nodes *Eij* , even if some of

the specifications may be equal, e.g. *B*1 = *B*2

* Circular connections as in (2) are forbidden, unless we duplicate the body as in (1) of Figure [3](#_bookmark3). Otherwise the semantics of such a circular architecture is not defined, as it would cause the identification of the export interfaces *E*11 and *E*12 of component *COMP*1 or other kinds of unwanted side effects.

*I*  *b*1 *B* ¸*, I I*  *b*1 *B* ¸*, I*

1

*con*1

c*z*

*b*2 2

*con*2

c*z*

1

*con*1

c*z*

*b*2 2

*con*2

c*z*

*E*11

c*z*

* 1. *E*12

c*z*

*E*11***¸****¸¸*

***¸¸¸***

***¸***

***¸¸***

* 1. *E*12

z,

*E*1*j * z*B* 1 *B*1 ¸*c E*1*j E*1*j*  z*B* 1

Fig. 3. Noncircular and circular connector diagrams

* + - A well-known result from graph theory for undirected graphs (i.e. forget- ting the direction of the arcs) is that in a connected, noncircular graph the number of nodes exceeds the number of edges by one: #*nodes*(*G*) = #*edges*(*G*) + 1. In Figure [3](#_bookmark3) we have #*nodes*(1) = 9 = #*edges*(1) + 1, but #*nodes*(2) = 7 /=8 = #*edges*(2) + 1

Now, we introduce the architecture based on connectors and components. Similarly to connectors we obtain an architecture diagram and an architecture graph. The first describes the architecture at the level of specifications and the second as a graph, where nodes are connectors or components.

An architecture *A* of arity (*k, l*) consists of *k* components and *l* connectors, an architecture diagram *DA* and an architecture graph *GA*: The architecture diagram *DA* is a diagram built up from the *l* connector diagrams and the connection transformations satisfying the following conditions

1. Connector Condition: Each import interface *I* of a connector is con- nected by an arrow, labeled with a connection transformation *con* : *I* =⇒

*E*, to exactly one export interface *E* of one component.

1. Component Condition: Each export interface *E* of a component is connected at most to one import interface *I* of a connector by an arrow from *I* to *E*, labeled with a connection transformation *con* : *I* =⇒ *E*.
2. Noncircularity: The architecture diagram *DA* is connected and noncir- cular aside from the arrows’ direction.

The architecture graph *GA* of architecture *A* is obtained from the archi- tecture diagram *DA* by shrinking each connector diagram in *DA* to the corre- sponding connector graph. Hence, it consists of nodes labeled by the connec- tors and components and arrows in between labeled with the corresponding connection transformations.

Petri Net Example for a Connector Architecture: Preparing a Party To provide a concrete example for a connector architecture we have chosen place/transition nets as the underlying specification technique. In Section [5](#_bookmark24) we discuss the instantiation of connector architectures to Petri nets. The main focus of this simple and small example is to illustrate the introduced concepts, using a scenario from everybody’s life: preparing a party. In order to show the Petri nets explicitly, a component is drawn as a rectangular, including one or more rectangulars in the first row, where each contains one export net. The rectangular in the second row contains the body net and the component name. A connector is drawn as a rectangular as well. In the first row we have exactly one rectangular, containing the body. The second row comprises at least two rectangulars, each containing one import net.

The components comprise the following activities: Comp invite in Figure [4](#_bookmark5) the invitation of guests and the management of cancellations. The component has merely one export interface Ei. The places of the export net are preserved under the transformation, but the transition is replaced by the whole subnet in between the places. In Figure [5](#_bookmark4) we give the usual diagram for the component Comp invite, but without the explicit Petri nets.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Ei**  *guests invite guest list* | | | | |
| *guests guest list*  **Comp\_invite** | *cancelation* | *invite* | *finished* | *guest list* |

**Ei Comp\_invite**



Fig. 4. Component Comp invite Fig. 5. Classical Diagram

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Es1**  *1. list shop* | *basics* | **Es2** | | | | |
| *basics* | *2. list* |  | *shop* | *fresh food* |
| *1. list*  **Comp\_shop** | *shop* | *2. list*  *basics* |  | *shop* | | *fresh food* |

Fig. 6. Component Comp shop

|  |  |
| --- | --- |
| **Ec1**  *ingredients cook food* | **Ec2**  *food prepare buffet buffet*  *ready* |
| *fry*  *ingredients prepare bake food put buffet*  *ready*  *broil*  *repeat garnish*  **Comp\_cook** | |

Fig. 7. Component Comp cook

The component Comp shop models the shopping, where we assume that the party requires shopping two times sequentially, once for the basics (beverages, potato crisps, olives, etc.) and once for the fresh food (bread, cheese, fruits, etc.). So we have two export nets Es1, Es2 that are transformed into the body net. These transformations keep the places and replace the transition by the subnet in between. The last component Comp cook comprises the cooking and preparations of the buffet, and provides therefore two export interfaces Ec1 and Ec2.

The connector Con week in Figure [8](#_bookmark6) connects the activities that concern the preparations of the week before the party and provides two import nets Iw1 and Iw2. These are mapped by inclusion into the body net. In Figure [9](#_bookmark7) we give the usual diagram for the connector Con week, but without the nets.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Con\_week**  *invite do shopping list get beverages* | | | | | | | |
| **Iw1** | *invite* |  |  | **Iw2** | *get beverages* |  |  |
|  |  |

**Iw1 Con\_week**

**Iw2**



Fig. 8. Connector Con week Fig. 9. Classical Diagram

The connector Con day in Figure [10](#_bookmark8) connects the preparations on the day of the party, and provides two imports as well.

Fig. 10. Connector Con day



**Con\_day**

*shoppinglist*

*basics*

**Id1**

*shoppinglist*

*prepare food*

*prepare food*

*shop fresh food*

*shop fresh food*

**Id2**

*basics basics*

The architecture diagram *DA* is given in Figure [11](#_bookmark9), where the transformations between the import nets and the corresponding export nets merely rename places or transitions. In Figure [11](#_bookmark9) we use the names of connectors and com- ponents for the corresponding bodies.

*C,o* *n¸we*,*¸e¸k ¸*

*,,*

*Co* *n,da¸*¸*y ¸*

*Iw*1

*,,,,*

*,,*

*¸¸¸¸*

*¸*

*Iw*2

*Id*1

*¸¸¸¸¸*

*Id*2

c*z* c*z*

*Ei Es*1*¸¸¸¸*

c*z*

*.Es*2

*...*

c*z*

*Ec*1

*ssss*

*sEc*2

c*z ¸¸¸¸¸¸*

*.....*

z*˛* *. s*

*ssssss*

c*z* *s s*

*Comp invite Comp shop Comp cook*

Fig. 11. Architecture diagram *DA*

# Composition of Components

In this section we present our concept for the composition of components by a connector, which is the basic step for the construction of the semantics of architectures in the next section. For this purpose we have to require in our generic framework an essential property for embeddings and transforma- tions: The parallel extension property is a key concept which generalizes the extension property in [[4](#_bookmark28)] from single (sequential) to multiple (parallel) transformations. The intuitive idea is that each family of transformations (*ti* : *SPECi* ⇒ *SPEC*')*i*∈*I* can be extended to a parallel transformation

*i*

*t* : *SPEC* ⇒ *SPEC*', provided that we have independent embeddings *bi* :

*SPECi* ⇒ *SPEC*'. More precisely, the parallel extension property means the following: For each family (*bi* : *SPECi* → *SPEC*)*i*∈*I* of independent em- beddings and for each family of transformations (*ti* : *SPECi* =⇒ *SPEC*')*i*∈*I*

*i*

there is a canonical (parallel) transfor- mation *t* : *SPEC* =⇒ *SPEC*' together with the independent embeddings

(*b*' : *SPEC*' → *SPEC*)*i*∈*I* leading to

*i*

*i*

*SPEC*  *bi*  *SP EC*

*ti* (1) *t*

*i*

c*z b*' c*z*

*SPEC*' *i*  *SP EC*'

*i*

the parallel extension diagram (1) Fig. 12. Parallel extension diagram

in Figure [12](#_bookmark11). Moreover, parallel extension diagrams are required to be closed

under vertical composition and to include the (classical) extension diagram in the sense of [[4](#_bookmark28)] as a special case where all but one of the trans- formations *ti* are identical transformations. Now we are able to define the composition of components.

The composition of *n* components by a con-

nector of arity *n* is defined as follows: Given the corresponding connector diagram (see Figure [1](#_bookmark2)) we

*i*

*I*  *bi*  *B*

construct the corresponding parallel extension di- agram (1) in Figure [13](#_bookmark12). The result of the compo-

sition of the components *COMP*1*, ..., COMPn* by

*ti*=*eik* ◦*coni*

c*z*

(1) *t*

*b*' c*z*

the connector *CON* with the connection transfor- mations *con*1*, ..., conn* is again a component

*Bi*  *i*  *B* '

Fig. 13. Composition

*COMP* = (*B*'*,* (*e*' : *Eij* =⇒ *B*')*ij*∈*I*×*J*) with

*ij*

' := *b*' ◦ *eij* : *Eij* =⇒ *B*' for each *ij* ∈ {1*, ..., n*}× ({1*, ..., mi*}\ {*k*})= *I* × *J* .

*e*

*i*

*ij*

In this case we say that *e*' are extensions of *ei* .

*ij*

*j*

In case of binary components and binary connectors we use the following nice infix notation *COMP* = *COMP*1 +*CON COMP*2. Otherwise we use the notation *COMP* = *CON* (*COMP*1*, ..., COMPn, con*1*, ...conn*). The result below can be extended to the composition of connectors with multiple import interfaces.

Theorem 3.1 (Associativity of Binary Component Composition)

*Given an architecture A with*

*binary components and binary*

*CON¸*1*¸¸*

*CON¸*3*¸¸*

*connectors with the architec-*

*...*

\_*s*

*..*

*¸¸¸¸* z

*\_ ..........*

*¸¸¸¸¸¸¸* z*\_*

*ture graph GA in Figure* [*14*](#_bookmark14)*,*

*then we have the following as- sociativity law:*

*COMP*1 *COMP*2 *COMP*3

Fig. 14. Binary connectors and components

(*COMP*1 +*CON*1 *COMP*2) +*CON*2 *COMP*3 = *COMP*1 +*CON*1 (*COMP*2 +*CON*2 *COMP*3)

\_*s*

Proof idea: Each side of the equation can be shown to be equal to a parallel composition of *COMP*1, *COMP*2, and *COMP*3 via *CON*1 and *CON*2 using the parallel extension property.

Composition for the Petri Net Example

Composing first the components Comp invite and Comp shop using the connector Con week we obtain the new component Comp invshop illus- trated in Figure [15](#_bookmark15).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Es2** *basics* | *2. list* | *shop* | | *fresh food* | |
| *guests guest list invite* *cancelation*  *do shopping list shop*  *1. list 2. list*  **Comp\_invshop** *basics* | *finished* | | *shop* | | *guest list*  *fresh food* |

Fig. 15. Comp invshop

The corresponding parallel extension

*Iw*1 *C on week* ¸*, Iw*2

diagram for the construction of the c*z*

body *B*' of Comp invshop is de- *Ei*

picted in Figure [16](#_bookmark16), where we again c*z*

c*z*

*Es*1

c*z* c*z*

use the names of components or con- nectors also for their bodies.

*Comp invite*  *B* ' ¸*, Comp shop*

Fig. 16. Parallel Extension diagram

Connecting this component via the connector Con day with the components Comp cook we obtain the component Comp invshopcook in Figure [17](#_bookmark17). But we can also change the order in which we compose.

Fig. 17. Comp invshopcook



**Ec2**

*buffet ready*

*guests*

*guest list*

*1. list*

*2. list*

*buffet ready*

*basics*

*food*

*put*

**Comp\_invshopcook**

*prepare*

*finished*

*invite*

*guest list*

*garnish*

*repeat*

*broil*

*bake*

*fry*

*shop*

*shop*

*do shopping list*

*cancelation*

*prepare buffet*

*food*

So, using the connector Con day we first compose the components Comp shop and Comp cook. Connecting this one via the connector Con week with the component Comp invite we again obtain the compo- nent Comp invshopcook in Figure [17](#_bookmark17). This commutativity is ensured by Theorem [3.1](#_bookmark13).

# Semantics of Architectures

In this section we define the semantics of architectures. In fact, we construct a well-defined single component as semantics, which corresponds to the compo- sition of all components using all connectors of the given architecture. More precisely, for an architecture there are reduction rules that visualize step by step the composition of components via connectors. Both reduction rules are productions *p* = (*L* ← *K* → *R*) in the sense of the

algebraic approach to graph transformation [[5](#_bookmark29)]. A derivation step in this approach is given by two pushout diagrams (1) and (2) in Figure [18](#_bookmark19), written *G* =⇒ *H* via (*p, m*), where *m* : *L* → *G* is a graph morphism, that represents the match of *L* in *G*. Intuitively, we remove

*L* ¸*, K*  *R*

*m* (1) (2)

JJJ

*G* ¸*, D*  *H*

Fig. 18. *G* =⇒ *H*

(*L* − *K*) from *G* in step 1 leading to the context graph *D* in (1). And then

we add (*R* − *K*) leading to the result *H* in (2). The pushout property of (1) and (2) means intuitively that *G* is the gluing of *D* and *L* along *K* in (1), respectively *H* is the gluing of *D* and *R* along *K* in (2).

Given an architecture *A* with the architecture diagram *DA* and the archi- tecture graph *GA* there is for each connector *CON* the following diagram reduction rule *COND*:

*Ii*

*bi*

*B*

*coni*

c*z*

*Eik i*∈{1*,...,n*}

*eik*

c*z*

*E*

*eij*

*ij*

z*B i*

*j*∈{1*,...,mi*}\{*k*}

*Eij*

*COND*

:

¸*lD,*

*rD*

where *B*' and *e*'

*ij*

= *eij*

Fig. 19. Diagram reduction rule

* *b*' is defined by the composition:

*i*

*Eij*

*e*'

*ij* z*B* '

*COMP* = *CON* (*COMP*1*, ..., COMPn, con*1*, ...conn*)

= (*B*'*,* (*e*'

*ij*

: *Eij* =⇒ *B*')*ij*∈*I*×*J* )

for each *ij* ∈ {1*, ..., n*}× ({1*, ..., mi*}\ {*k*})= *I* × *J* .

The corresponding graph reduction rule *CONG* is given in Figure [20](#_bookmark20) where *COMP*1*, ..., COMPn* are mapped to *COMP* . So, a reduction step *COND* : *DA* =⇒ *DA*' , respectively *CONG* : *GA* =⇒ *GA*' is given by a derivation step at the level of architecture diagrams, respectively architecture graphs.

For both derivation steps we have inclusions for the matches. Note that *rG*

is neither injective nor label-preserving, nevertheless for the reduction rule

*CONG* : *GA* =⇒ *GA*' the labels of *G*'

*con*1 ***ccc*** *,****,,****,conn*

*CON****,****,*

,

*COM ...*

***ccc***

*,****,,***

*P*1

*COMPn*

z

*A*

¸*l,G*

are well-defined by *GA* and *COMP* .

*rG*

*COMP*1*...COM Pn*

*CONG*:

Fig. 20. Graph reduction rule

*COMP*

We can show by an Architecture Reduction Lemma that an architecture reduction rule *CON* = (*COND, CONG*) reduces an architecture *A* to a well-defined smaller architecture *A*' with *DA*' and *GA*' as defined above. The

application of *CON* is denoted by *A C*=*O*⇒*N*

*A*'. *A*' is smaller than *A* in the

following sense: If *A* is of arity (*k, l*) we can show that *A*' is of arity (*k* − *n* + 1*,l* − 1), if *CON* has arity *n*. This means: Given an architecture *A* consisting of *k* components and *l* connectors we obtain the architecture *A*' with *k* − *n* +1 components and *l* − 1 connectors.

Now we can give the semantics of an architecture as the result of as many reduction rules as possible. The semantics of an architecture *A* is any component *COMP* obtained by a sequence of architecture reduction steps

*A* =⇒∗ *COMP* from *A* to *COMP* . The main result given in Theorem [4.1](#_bookmark21)

shows that this semantics always exists and is unique. The main reason there- fore is given by the Church-Rosser property, stating that any two architecture reduction steps are locally confluent.

Theorem 4.1 (Existence and uniqueness of architecture semantics) *For each architecture A there is a unique component COMP which is the semantics of A. COMP is obtained by* any *reduction sequence, where connectors of A are reduced in arbitrary order:*

*A* =⇒∗ *COMP*

Proof idea: The architecture reduction step sequences have unique normal forms, as they satisfy the Church-Rosser property due to the following rea- soning: If the matches of *CON*1 and *CON*2 in the architecture diagram *DA*0 of *A*0 are disjoint, then they are independent and we obtain the result by the Church-Rosser Theorem for graph transformations. Otherwise, due to non- circularity of *DA*0 , the matches can overlap at most in one component which allows applying Theorem [3.1](#_bookmark13), respectively an extension with multiple import

interfaces. So, each maximal sequence has length *n*, where *n* is the number of connectors in *A*.

Semantics of the Petri Net Example

The application of the reduction rule *DA*

*CONWEEK* z*D*

that eliminates the

connector Con week results in the architecture diagram *DA*1 given in Figure

*A*1

[21](#_bookmark22). There we have the new component Comp invshop with one export Es2, that is connected as before to the import Id1 of connector Con day. The

*A*3

application of *DA*1

*CONDAY* z*D*

that eliminates the connector Con day

yields then the semantics of *A*. This is the component Comp invshopcook

in Figure [17](#_bookmark17) with only one export left, namely Ec2. Theorem [4.1](#_bookmark21) ensures that this semantics exists uniquely. If we start for example with the application

of *DA*

*CONDAY* z*D*

that eliminates the connector Con day we obtain a

component Comp shopcook with the two exports Es1 and Ec2. The cor- responding architecture diagram *DA*2 is given in Figure [22](#_bookmark23). The subsequent

*A*2

*A*3

application of *DA*2

*CONWEEK* z*D*

eliminating the connector Con day yields

then again the component Comp invshopcook.

*Con day*

*Co* *n ¸we*¸*e¸k¸¸*

*¸ ¸*,*¸¸¸ ¸¸¸¸*

*Id*1

*¸¸¸*

*Id*2

*Iw*1

*¸¸*

*Iw*2

c*z*

*Es*2

c*z*

*Ec*1

c*z*

*Ec*2 *Ei*

c*z*

*Es*1

*ssss*

*sEc*2

c*z* c*z*

*ssssss*

c*z* *s s*

*Comp invshop*

c,*zj*

*Comp cook*

*Comp invite Comp shopcook*

Fig. 21. Architecture Diagram *DA*1

Fig. 22. Architecture Diagram *DA*2

# Instantiations

In [[1](#_bookmark26)] Allen and Garlan introduced architectural connectors using CSP [[9](#_bookmark33)] as specification formalism. In this section we shall very briefly sketch how CSP and Petri nets fit into our transformation framework.

CSP can be seen as an instance of our transformation framework as follows. First, we consider that a CSP specification *P* = (Σ*, Exp*) where Σ isa *process signature* (the set of symbols that can be used in *P* ) and *Exp* is a CSP process expression built over symbols in Σ. Then, we consider that a process *P* is embedded in a process *Q* if *P* is a parallel component of *Q*. More precisely, *P*1 = (Σ1*, Exp*1) is embedded in *P*2 = (Σ2*, Exp*2) if Σ1 ⊆ Σ2) and *Exp*2 ≡ *Exp*1|*Exp*' for some process expression *Exp*'. *P*1 and *P*2 are independently

embedded in *P*3 iff *Exp*3 ≡ *Exp*1|*Exp*2|*Exp*' for some process expression *Exp*'.

Finally, we consider that transformations are just CSP refinements modulo a signature embedding. This means that a CSP transformation *t* : *P*1 = (Σ1*, Exp*1) ⇒ *P*2 = (Σ2*, Exp*2) is an injective mapping *t* : Σ1 → Σ2 such that the translation of *Exp*1 through *t* satisfies that the failures and divergences of *Exp*2 are, respectively, a subset of the failures and divergences of *t*(*Exp*1), where *t*(*Exp*1) denotes the translation of *Exp*1 by the renaming of events defined by *t*.

Then, CSP can be seen as an instance of our generic approach as a consequence of the existence of parallel extensions:

If, for *i* = 1*,* 2, *ti* : *Pi* = (Σ*i, Expi*) ⇒ *P* ' = (Σ' *, Exp*') are transfor-

*i i* *i*

mations and *P*1*, P*2 are independently embedded in *P*3 = (Σ3*, Exp*3),

then *t*3 : *P* ' = (Σ' *, Exp*' ) is a parallel extension of *t*1 and *t*2, where

3 3 3

Σ' = Σ3 + (Σ' − *t*1(Σ1)) + (Σ' − *t*2(Σ2)), where + and − denote disjoint union

3 1 2

3

and set subtraction, *t*3 : Σ3 → Σ'

3

is the identity and *Exp*'

is the process

expression obtained by substituting *Exp*1 by *Exp*' and *Exp*2 by *Exp*' in

1

2

*Exp*3.

High-level replacement systems have been introduced in [[2](#_bookmark27)] as a generalization of the *Double Pushout* approach for graph transformations from graphs to several kinds of high-level structures in suitable categories including a large variety of different kinds of graphs and Petri nets ([[4](#_bookmark28)]). A transformation in the framework of this paper corresponds to a derivation sequence of high- level structures (e.g. Petri nets). The extension property for transformations considered in our general framework for components [[4](#_bookmark28)] is well-known as the Embedding Theorem in the theory of high-level replacement systems that holds if a consistency condition between embeddings and transformations is required. A similar consistency condition has to be formulated for the parallel extension property considered in this paper. This property can be shown for high-level replacement systems by considering the Embedding Theorem and the Parallelism Theorem shown in [[2](#_bookmark27)]. Moreover, we may allow also some overlapping of the embeddings, provided that the transformations preserve the overlapping part in the sense of parallel independence. This situation occurs for the embeddings of the connector Con day in Figure [10](#_bookmark8) in our running example.

# Conclusion

In this paper we have presented a general framework for connector architec- tures, based on our generic framework for components in [[4](#_bookmark28)], and motivated by the architectural connectors approach of Allen and Garlan in [[1](#_bookmark26)] which is shown to be a specific instance. It may be mentioned that the papers

[[11](#_bookmark35),[20](#_bookmark41)] concerning architecture reconfiguration based on graph transformation techniques can be considered complementary to ours, because graph trans- formations and high-level replacement play already a fundamental role in our approach.

Future work comprises further and more concrete instantiations especially high-level Petri nets and graph transformation systems. Moreover, larger case studies to evaluate the practical impact are an important task. On the theo- retical side the instantiations to general high-level replacement systems has to be worked out in more detail and our framework should be extended to other aspects studied in [[4](#_bookmark28)] and architecture reconfiguration in [[11](#_bookmark35),[20](#_bookmark41)].

# Acknowledgement

This work is partially supported by the TMR network SEGRAVIS and the Spanish project MAVERISH (TIC2001-2476-C03-01) and by the CIRIT Grup de Recerca Consolidat 2001SGR 00254.

# References

1. R. Allen and D. Garlan. A formal basis for architectural connection. *ACM Transactions on* *Software Engineering and Methodology*, 1997.
2. H. Ehrig, A. Habel, H.-J. Kreowski, and F. Parisi-Presicce. Parallelism and concurrency in high-level replacement systems. *Math. Struct. in Comp. Science*, 1:361–404, 1991.
3. H. Ehrig and B. Mahr. *Fundamentals of Algebraic Speciﬁcation 2: Module Speciﬁcations and Constraints*, volume 21 of *EATCS Monographs on Theoretical Computer Science*. Springer Verlag, Berlin, 1990.
4. H. Ehrig, F. Orejas, B. Braatz, M. Klein, and M. Piirainen. A generic component concept for system modeling. In *Proc. FASE ’02*, LNCS 2306. Springer, 2002.
5. H. Ehrig, M. Pfender, and H. Schneider. Graph grammars: an algebraic approach. In *14th* *Annual IEEE Symposium on Switching and Automata Theory*, pages 167–180. IEEE, 1973.
6. D. Garlan, R. Monroe, and D. Wile. Acme: An Architecture Description Interchange Language. In *Proc. of CASCON’97*, pages 169–183, 1997. [http://www.cas.ibm.ca/cascon/cfp.html.](http://www.cas.ibm.ca/cascon/cfp.html)
7. F. Griffel. *Componentware – Konzepte und Techniken eines Softwareparadigmas*. dpunkt Verlag, 1998.
8. V. Gruhn and A. Thiel. *Komponentenmodelle: DCOM, JavaBeans, EnterpriseJavaBeans, CORBA*. Addison-Wesley, 2000.
9. C. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.
10. C. Hofmeister, R. Nord, and D. Soni. *Describing Software Architecture in UML*, pages 145–159. Kluwer Academic Publishers, 1999.
11. M. L¨owe. Evolution pattern. postdoctoral thesis, Technical University of Berlin, 1997.
12. S. Mann, B. Borusan, H. Ehrig, M. Große-Rhode, R. Mackenthun, A. Su¨nbu¨l, and H. Weber. Towards a component concept for continuous software engineering. Technical Report 55/00, FhG-ISST, 2000.
13. J. Padberg. *Abstract Petri Nets: A Uniform Approach and Rule-Based Reﬁnement*. PhD thesis, Technical University Berlin, 1996. Shaker Verlag.
14. J. Padberg. Petri net modules. *Journal on Integrated Design and Process Technology*, 6(4):105– 120, 2002.
15. J. Padberg, H. Ehrig, and L. Ribeiro. Algebraic high-level net transformation systems.

*Mathematical Structures in Computer Science*, 5:217–256, 1995.

1. G. Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations*. World Scientific, 1997.
2. M. Shaw, R. Deline, D. V. Klein, T. L. Ross, D. M. Young, and G. Zelesnik. Abstractions for Software Architecture and Tools to Support Them. *IEEE Transactions on Software Engineering*, 21(4):314–315, 1995.
3. M. Shaw and D. Garlan. *Software Architecture - Perspectives on an Emerging Discipline*. Prentice Hall, 1996.
4. C. Szyperski. *Component Software – Beyond Object-Oriented Programming*. Addison-Wesley, 1997.
5. M. Wermelinger and J. Fiadeiro. A graph transformation approach to software architecture reconfiguration. *Science of Computer Programming*, 44(2):133–155, 2002.