Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 206 (2008) 111–131
www.elsevier.com/locate/entcs

Architectural Connectors for Sequence Diagrams

Fernando Orejas

Departament de LSI
Universitat Politècnica de Catalunya
Barcelona, Spain

Sonia Pérez

Inst. Sup. Politécnico J. A. Echevarría
Havana, Cuba

Abstract

The notion of architectural connector was developed by Allen and Garland as an important concept for the design of software architectures. In this paper, based on previous work introducing a generic approach for the definition of component-based concepts, we study how architectural connectors and components can be defined for UML2 sequence diagrams as a first step for applying this approach to full UML. A case-study of a small lift system is used to illustrate these ideas.

1 Introduction

The development of component-based systems and service oriented systems is currently an important area in software engineering. Unfortunately, little work has been dedicated to the modelling and specification phase of this kind of systems. As a consequence most modelling techniques or formalisms lack adequate notions to support the architectural development of component-based systems. See, for instance, [7] for a discussion on how to extend UML with adequate component concepts.

A modelling approach that we consider very interesting is based on the use of *architectural connectors* [1,13]. 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 implement some functionality while connectors describe policies of interaction of the connected components. Originally, the language used in [1,13] for the specification and modelling of components and

1571-0661 © 2008 Published by Elsevier B.V. Open access under CC BY-NC-ND license.
doi:10.1016/j.entcs.2008.03.078

connectors was CSP [8]. The work using this approach was followed by Fiadeiro (e.g. see [6]), who (in some sense) generalized the approach by putting it into a categorical context In this case, the modelling language used was the coordination language COMMUNITY.

In [3], we developed a very generic approach for the modelling of component-based systems whose aim was to allow the definition of component concepts associated to arbitrary formal or semiformal specification methods. The idea was that one could instantiate this generic approach to any arbitrary method, as long as one could prove that it satisfied certain properties. In particular, different instantiations were sketched in terms of Petri Nets, graph transformation systems or algebra transformation systems. Later, we generalized this approach by allowing, for instance, that components could have several interfaces and adapted it to deal with architectural connectors. The approach is described in detail in [5], but preliminary results were presented in [4].

In the case of UML [12], a de facto standard for many industrial applications, the notion of component in UML 1.4 [2] has a very limited nature, being essentially a kind of syntactic package. In the case of UML 2.0 the situation has improved considerably, however the construction has stil limitations [9]. In this paper, we present the instantiation of our approach to the case of the sequence diagrams of UML2 overcoming these limitations. More precisely, we study how the notions of embedding and transformation (which are key notions in our approach) can be defined. And we show that these notions satisfy the required properties. Especially, the so-called extension and parallel properties.

Based on these results we have defined stereotypes in UML for our components and connectors and, to provide some tool support, we have implemented a plugin for the tool Visual Paradigm. This work is not presented in this paper, but can be found in [11]

A preliminary version of the work presented here was presented in [10]. However, there are important differences between these two versions. On one hand, in [10] only the case of basic sequence diagrams was covered. On the other, in that paper we imposed an important restriction on connectors. In particular, the roles of a connector had to be disjoint. This restriction may be reasonable for many modelling techniques, but it is too severe when dealing with sequence diagrams. Actually, the example that we use to illustrate our approach does not satisfy it: a lifeline is shared by the roles of the given connector. In this sense, eliminating this restriction implied a significant reformulation of the main concept and results, including the introduction of the parallel consistency property.

The paper is organized as follows. In the next section we review our generic approach to components and architectural connectors. The third section describes the instantiation of our approach to the case of basic sequence diagrams in some detail, i.e. diagrams that are not built using the composition operators provided by UML2. In section 4, we extend the instantiation to cover also these operators. In section 5, we present a small case study, a lift system, to show how these concepts can be applied in practice. Finally, in the last section, we draw some conclusions.

2 A generic framework for architectural components and connectors

In this section we will describe the basic ingredients of our generic framework. More specifically, following the approach introduced in [1], we consider two kinds of constructions: components and connectors. Components are units that provide some kind of services. 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 will be called transformations.

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

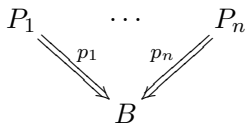

Fig. 1. Diagram of a component

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]). Each role describes the expected behavior and services provided by the component to which it will be connected, while the body describes the interaction of the components that will be connected through the given roles. In this sense, the body of a connector extends the specifications included in the roles by defining over them an interaction policy. We will say that in a connector the roles are *embedded* in the body.

Definition 2.2 (Connectors) A connector $CON = (B, r_1 : R_1 \to B, ..., r_n : R_n \to B)$ for $n \geq 2$ is given by the body B and a family of roles R_i with role embeddings $r_i : R_i \to B$ for $i \in \{1, ..., n\}$. A connector is graphically represented by the diagram in Fig. 2.

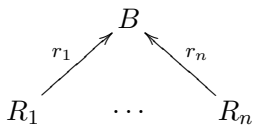

Fig. 2. Diagram of a connector

To be able to link a role to a given port, they must be compatible. In particular, to be compatible, the port should be some kind of refinement of the role. Now, 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, 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.

Embeddings and transformations are assumed to 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. In addition, we may expect to apply (local) transformations in a larger context. In particular, if we know that $t_1 \colon M_1 \Longrightarrow M_1'$ is a transformation of a given model M_1 yielding another model M_1', then when M_1 is embedded in a larger model M_2, it seems reasonable to consider that we should be able to apply locally t_1 in the context of M_2 yielding a model M_2' which embeds M_1' and is a transformation of M_2. However, there may be some kind of incompatibility between the t_1 and the embedding $e \colon M_1 \to M_2$: for instance, t_1 may add some new names to M_1 but these names may have already a meaning in M_2 which is incompatible with their expected role in M_1'. As a consequence, we consider that our framework must include a *consistency relation* between embeddings and transformations such that the following extension property holds:

Definition 2.3 (Extension Property) For each transformation $t_1 \colon M_1 \Longrightarrow M_1'$, and each embedding $e \colon M_1 \to M_2$, such that e and t_1 are consistent, there is a selected transformation $t_2 \colon M_2 \Longrightarrow M_2'$, with embedding $e' \colon M_1' \to M_2'$, called the *extension* of t_1 with respect to e, leading to the following extension diagram:

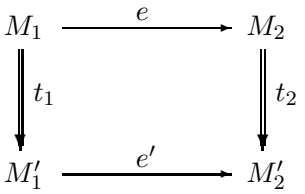

Fig. 3. Extension

It must be pointed out that, in a framework $\mathcal{F}$, given t_1 and e as above, there may be several t_2 and e', satisfying this extension property. Our assumption means that only one such t_2 and e' are chosen, in some well-defined way, as the extension

of t_1 with respect to e.

This extension property should be generalized 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, and if these transformations are consistent with each other 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. More precisely, our framework must include a *parallel consistency relation* defined on families of transformations with respect to corresponding families of embeddings such that the parallel extension property stated below holds. This notion of parallel consistency of a family of transformations with respect to a family of embeddings is also generic and can be instantiated differently for different specification or modelling techniques. The intuition of parallel consistency should be clear: if two submodels M_i and M_j share a common part, then this part should be refined in a consistent way in the corresponding models M'_i and M'_j. In particular, this means that if the roles are disjoint (which will probably happen in many cases) then parallel consistency will probably hold in a trivial way.

Definition 2.4 (Parallel Extension Property) Given transformations $\{t_j : M_j \Longrightarrow M'_j\}_{j\in J}$ and embeddings $\{e_j : M_j \to M\}_{j\in J}$ for some finite set J, such that each embedding e_j is consistent with the corresponding transformation t_j for every $j \in J$ and the family of transformations $\{t_j : M_j \Longrightarrow M'_j\}_{j\in J}$ is parallel consistent with respect to the family of embeddings $\{e_j : M_j \to M\}_{j\in J}$, then there is a selected transformation $t : M \Longrightarrow M'$, with embeddings $\{e'_j : M'_j \to M'\}_{j\in J}$, called the *parallel extension* of $\{t_j : M_j \Longrightarrow M'_j\}_{j\in J}$ with respect to $\{e_j : M_j \to M\}_{j\in J}$, leading to the parallel extension diagram in Fig. 4, such that parallel extension coincides with extension in the case where J consists just of one index.

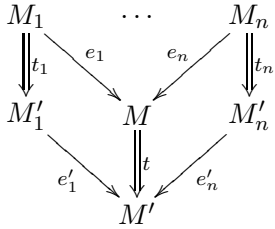

Fig. 4. Parallel Extension

Moreover, it should be possible to compose parallel extension diagrams vertically. This means that if $t : M \Longrightarrow M'$ is the parallel extension of $\{t_j : M_j \Longrightarrow M'_j\}_{j\in J}$ with respect to $\{i_j : M_j \to M\}_{j\in J}$ and $t' : M' \Longrightarrow M''$ is the parallel extension of $\{t'_j : M'_j \Longrightarrow M''_j\}_{j\in J}$ with respect to $\{i'_j : M'_j \to M'\}_{j\in J}$, as in Fig. 5, then $t' \circ t : M \Longrightarrow M''$ is the parallel extension of $\{t'_j \circ t_j : M_j \Longrightarrow M''_j\}_{j\in J}$ with respect to $\{i_j : M_j \to M\}_{j\in J}$.

Again, we assume that parallel extension is uniquely defined by the given $\{t_j\}_{j\in J}$ and $\{e_j\}_{j\in J}$. Now, using parallel composition we are able to define the composition

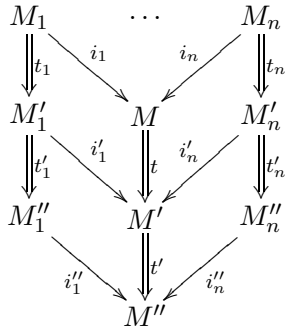

Fig. 5. Vertical composition of parallel extensions

of components through an architectural connector.

Given a connector $CON = (B, r_1, ..., r_n)$, and components $COMP_i = (B_i, p_{i_1}, ..., p_{i_{m_i}})$ with connector transformations $con_i : R_i \Longrightarrow P_{i_k}$ with $1 \leq k \leq m_i$ for $i \in \{1, ..., n\}$ we obtain the **connection diagram** in Figure 6. Then, the **composition** of the connection diagram in Figure 6, is defined in terms of the parallel extension diagram (1) in Figure 7, where, for every i, $t_i = p_{i_{k_i}} \circ con_i$. In particular, the result of the composition of the components $COMP_1, ..., COMP_n$ by the connector CON with the connection transformations $con_1, ..., con_n$ is the component whose body is B' and that includes all the ports $r'_{i_j} \circ p_{i_j} : B_{i_j} \Longrightarrow B'$ in each component $COMP_i$ which is not connected to the role r_i in the above diagram.

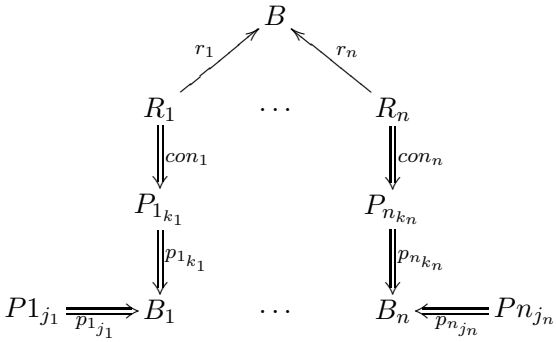

Fig. 6. Connector Diagram

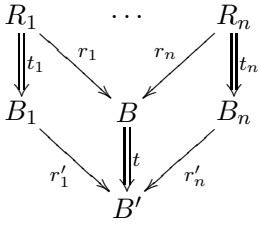

Fig. 7. Composition

Based on this kind of connection diagrams, we can define a notion of architecture as a diagram involving several components and connectors interconnecting them

in a non-circular manner. In [5] we proved that such architectures denote, after the evaluation of all the composition operations involved, a component which is independent on the order of evaluation of these operations.

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. 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 consistency relation between transformations and embeddings.
- An extension construction for consistent transformations and embeddings according to definition 2.3.
- An extension construction transformations and embeddings according to definition 2.3.
- A parallel consistency relation for J-indexed families of transformations with respect to J-indexed families of embeddings.
- A parallel extension construction for every J-indexed family of transformations which is parallel consistent with respect to a J-indexed family of embeddings satisfying the properties in definition 2.4.

3 Components and connectors for basic sequence diagrams

In this section we will study how to instantiate our generic framework for the case of basic sequence diagrams, i.e. diagrams that are built using just lifelines and interactions between the objects involved. Then defining the instantiation means to define notions of embedding and transformation and showing that they satisfy the required properties.

In the case of sequence diagrams, specifications typically consist of several diagrams, and not just of a single one, describing different scenarios. As a consequence, the notions of embedding and transformation should be defined on sets of diagrams. However, for simplicity, in this paper we will just define these notions on single diagrams. It should be clear that the pointwise generalization of these notions to deal with sets is straightforward

Definition 3.1 A *sequence diagram* S over a set of messages M is a triple (L, Loc_L, I), where L is the set of lifelines corresponding to the objects that are shown in the diagram; for each $l \in L$ Loc_l is the partially ordered set of locations corresponding to the lifeline l; and I is a set of interactions, where an interaction ε is a triple (loc_1, loc_2, m) where loc_1 and loc_2 are locations associated to some lifelines

in L and m is a message in M and such that two interactions never occur simultaneously, i.e. if $(loc_1, loc_2, m), (loc_3, loc_4, m') \in I$, with $(loc_1, loc_2, m) \neq (loc_3, loc_4, m')$ and $loc_i, loc_j \in Loc_l$, $i \in [1,2]$, $j \in [3,4]$ then $i \neq j$.

Moreover, we assume that sequence diagrams must satisfy that the *precedence relation* defined over the set of interactions of the diagram, $prec_I$, is a partial order, where $prec_S$ is the reflexive and transitive closure of the least relation satisfying that given $(loc_1, loc_2, m), (loc'_1, loc'_2, m') \in I$ if loc_i, loc'_j $(i, j \in [1,2])$ are in the same lifeline and $loc_i < loc'_j$ then $(loc_1, loc_2, m), (loc'_1, loc'_2, m') \in prec_I$.

We have considered that the locations on a lifeline are just partially ordered and not totally ordered. The reason is that in UML 2 one may have co-regions within lifelines, i.e. parts of lifelines whose locations are not considered ordered. A sequence diagram may involve any number of "useless" locations, i.e. locations that do not take part in any interaction.

Definition 3.2 A sequence diagram $S = (L, Loc_L, I)$ is *minimal* if for every location $loc \in Loc_L$ there is an interaction $(loc_1, loc_2, m) \in I$ such that $loc = loc_1$ or, $loc = loc_2$

In what follows, we will assume that all diagrams are minimal.

We will consider that a sequence diagram is embedded into another one if the latter describes the same set of interactions (up to renaming) in the same partial order, perhaps intertwined with some other additional interactions. To be more precise:

Definition 3.3 A *message renaming* $h : M \to M'$ is an injective mapping on the sets of messages.

Let S be a sequence diagram over M, the *renaming* of $S = (L, Loc_L, I)$ through $h : M \to M'$, denoted $h(S)$ is the sequence diagram (L, Loc_L, I'), where I' is the set of interactions:

$$\{(loc_1, loc_2, h(m))/(loc_1, loc_2, m) \in I\}$$

Let $S = (L, Loc_L, I)$ be a sequence diagram over M, $S' = (L', Loc'_{L'}, I')$ a sequence diagram over M' and $h : M \to M'$ a message renaming such that $L \subseteq L'$. An *h-based embedding* $i : S \to S'$ is an L-indexed family of injective mappings $t = \{i_l : Loc_l \to Loc'_{L'}\}_{l \in L}$ preserving the order relations (i.e., they must be poset monomorphisms), such that:

(i) For every $(loc_1, loc_2, m) \in I$, with $loc_1 \in Loc_{l_1}$ and $loc_2 \in Loc_{l_2}$, we have that $i((loc_1, loc_2, m)) = (i_{l_1}(loc_1), i_{l_2}(loc_2), h(m)) \in I'$.

(ii) If $m' \in h(M)$ then for every $(loc'_1, loc'_2, m') \in I'$ there is $(loc_1, loc_2, m) \in I$ such that $(i_{l_1}(loc_1), i_{l_2}(loc_2), h(m)) = (loc'_1, loc'_2, m') \in I$.

Note that, according to this definition, all the interactions in the diagram S must be present in the diagram S' exactly in the same order (up to message renaming, but S' may include additional interactions. However, we consider that these additional

interactions should be new, i.e. should not be in $h(M)$. The intuition is that the embeddings should preservethe behavior. In particular, let us suppose that a diagram describes that, after sending the message a, an object sends a message b and then a message c. We think that this behavior would not be preserved by a diagram describing that the same object sends first the message a, then c, then b and then c. However, we could consider this behavior preserved if in between the sequence a,b,c, another message d is sent which could be considered not visible in S.

It should be obvious that the requirements for embeddings are satisfied. In particular, embeddings are closed under composition and the identity is an embedding.

Now, we will define our notion of diagram transformation. In our opinion, when considering refinement relations between sequence diagrams we may consider two different kinds of intuitions. A straightforward one is to consider a refinement relation as an implementation relation, i.e. a sequence diagram is refined by another one if the latter can be seen as an implementation of the former. In particular, if we consider that single interactions (sending messages) are refined or implemented by other sequence diagrams, then we could define that a sequence diagram D_1 is refined by another sequence diagram D_2 if D_2 is the composition of diagrams implementing the interactions in D_1. However, this is not the only intuition in our context. In particular, if D_2 is a refinement of D_1, the latter diagram may be just an abstraction of D_2, in the sense that some of the interactions described in the body are hidden in D_1 because they are considered irrelevant detail. In particular, this means just that D_1 is embedded in D_2. Putting these two intuitions together, we have that a transformation is a combination of an "implementation" and an embedding as we will see below. But, first, we need to define an operation for the composition of basic sequence diagrams. In particular, this operation will be used for defining the implementation of sequence diagram by composing the implementations of each interaction.

Definition 3.4 Given two diagrams $S = (L, Loc_L, I)$ and $S' = (L', Loc'_{L'}, I')$ over a set of messages M, we define the *composition* $S + S'$ as the diagram over M $(L \cup L', Loc''_{L\cup L'}, I \cup I')$ where for each $l \in L \cup L', Loc''_l$ is the poset $Loc_l + Loc'_l$ [1], where + denotes disjoint union and all the elements in Loc_l are considered smaller than all the elements in Loc'_l

It may be noted that this composition operation is associative, but not commutative.

Definition 3.5 Let L and L' be sets of lifelines such that $L \subseteq L'$, M and M' sets of messages and $\mathcal{S}$ a set of sequence diagrams whose sets of lifelines are included in L'. Let T be a set of *L-typed interactions over* M, i.e. a subset of $L \times L \times M$. An *implementation* $\mathcal{I}$ of T by $(L', \mathcal{S})$ is a pair of mappings $(\mathcal{I}_{Lines} : L \to 2^{L'}, \mathcal{I}_{Mess} : T \to \mathcal{S})$, such that:

[1] if l is not in L we assume Loc_l to be the empty set and, similarly, if l is not in L' we assume $Loc'_{l'}$ to be empty

(i) For every lifeline l in L, $l \in \mathcal{I}_{Lines}(l)$

(ii) If $l_0 \neq l_1$ then $\mathcal{I}_{Lines}(l_0) \cap \mathcal{I}_{Lines}(l_1) = \emptyset$.

(iii) If $\mathcal{I}_{Mess}(l_0, l_1, m) = (L_1, Loc_L^1, I_1)$ then $L1 = \mathcal{I}_{Lines}(l_0) \cup \mathcal{I}_{Lines}(l_1)$

If $\mathcal{I}$ is an implementation and $\varepsilon = (loc_0, loc_1, m)$ is an interaction in the diagram $S = (L, Loc_L, I)$, we define $\mathcal{I}(\varepsilon)$ as follows: if $loc_0 \in l_0$ and $loc_1 \in l_1$ and $\mathcal{I}_{Mess}(l_0, l_1, m) = S'$ then $\mathcal{I}(\varepsilon) = S'$; otherwise, $\mathcal{I}(\varepsilon)$ is the diagram consisting only of the interaction ε.

Let $S = (L, Loc_L, I)$ be a diagram over M and $\mathcal{I}$ an implementation by $(L', \mathcal{S})$ of a set T of L''- typed interactions over M, where $L \subseteq L''$ then the application of $\mathcal{I}$ to S, denoted $\mathcal{I}(S)$ is defined as follows. Let $\langle \varepsilon_1, \ldots, \varepsilon_n \rangle$ be a sequence of interactions, with $I = \{\varepsilon_1, \ldots, \varepsilon_n\}$, such that $(\varepsilon_j, \varepsilon_k) \in prec_I$ then $j < k$; then $\mathcal{I}(S) = \mathcal{I}(\varepsilon_1) + \ldots + \mathcal{I}(\varepsilon_n)$.

Finally, if $\mathcal{S}$ is a set of diagrams over M and $\mathcal{I}$ an implementation by $(L', \mathcal{S}')$ of T, then the application of $\mathcal{I}$ to S, denoted $\mathcal{I}(S)$, is the set of all diagrams $\mathcal{I}(S)$ such that $S \in \mathcal{S}$.

The intuition is, on one hand, that in the refinement of each lifeline other lifelines may be involved which are considered hidden at a higher abstraction level. In this sense, the first condition states that each lifeline is part of its own refinement. The second condition states that a lifeline cannot be involved in the implementation of two different lifelines. On the other hand, the third condition, states that if a (typed) interaction is implemented by a certain diagram, then this diagram includes only the lifelines which implement the lifelines occurring in the interaction. Then, applying an implementation to a diagram means replacing all the interactions by the corresponding diagrams defined by the implementation. Note that we allow to apply an implementation to diagrams whose sets of lifelines do not coincide, but are included, in the set of lifelines implemented by $\mathcal{I}$. Note also that a message renaming $h : M \to M'$, can be seen as a special case of an implementation.

It may be proved that the definition of $\mathcal{I}(S)$ is independent of the specific sequence of interactions chosen. In particular if ε and ε' are independent interactions then $\mathcal{I}(\varepsilon) + \mathcal{I}(\varepsilon') = \mathcal{I}(\varepsilon') + \mathcal{I}(\varepsilon)$. The reason is that, if the two interactions are independent then the lifelines involved in the interactions are disjoint and, as a consequence, the sets of lifelines involved in the diagrams $\mathcal{I}(\varepsilon)$ and $\mathcal{I}(\varepsilon')$ are also disjoint. In this context, we can define a notion of transformation or refinement over sequence diagrams.

Definition 3.6 Let $S = (L, Loc_L, I)$ and $S' = (L', Loc'_{L'}, I')$ be sequence diagrams over M and M', respectively. A *transformation* $t : S \Longrightarrow S'$ is a pair $(\mathcal{I}, i)$, where $\mathcal{I}$ is an implementation by $(L', \mathcal{S}')$ of a set T of L-typed interactions over M and i is an embedding of $\mathcal{I}(S)$ into S'.

It should be clear that this notion of transformation satisfies that the identity is a transformation and that embeddings are special cases of transformations. The proof that transformations are closed under composition is also easy. Given transformations $t_1 : S_1 \Longrightarrow S_2$ and $t_2 : S_2 \Longrightarrow S_3$, with $t_1 = (\mathcal{I}_1, i_1)$ and $t_2 = (\mathcal{I}_2, i_2)$,

first, we have to define, in an obvious way, the composition of implementations $\mathcal{I}_2 \circ \mathcal{I}_1$. Then, it is easy to see that there is an embedding from $\mathcal{I}_2 \circ \mathcal{I}_1(S_1)$ to S_3. Therefore, we just have to prove that extensions and parallel extensions exist for adequate notions of consistency and parallel consistency. We will, first, prove the existence of extensions in three steps. First, we will show that, under adequate conditions, if $i_1 : S_0 \to S_1$ and $i_2 : S_0 \to S_2$ are embeddings then we can define a diagram S_3 that embeds S_1 and S_2. Actually, the construction is a pushout in a category of embeddings, although we will not prove it. The second step will be to show that if $i_1 : S_0 \to S_1$ is an embedding and $\mathcal{I}_2$ is an implementation such that of $\mathcal{I}_2(S_0) = S_2$ then we can define an implementation $\mathcal{I}_1$ such that $\mathcal{I}_1(S_1)$ embeds S_2. From these two properties, we can easily conclude the extension property for basic sequence diagrams.

Definition 3.7 Two embeddings $i_1 : S_0 \to S_1$ and $i_2 : S_0 \to S_2$ are consistent if for all interactions $int, int' \in I_0$, if we have that $i_1(int) \leq i_1(int')$ and $i_2(int') \leq i_2(int)$, then $int = int'$.

Proposition 3.8 *Given two consistent embeddings $i_1 : S_0 \to S_1$ and $i_2 : S_0 \to S_2$ we can define an extension according to Fig. 8, where i'_1 and i'_2 are also embeddings.*

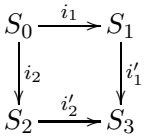

Fig. 8. Extension for embeddings

The proof is quite simple. It is enough to define the diagram $S_3 = (L_3, Loc^3_{L_3}, I_3)$ by putting together (by means of pushouts) the corresponding components of S_0, S_1 and S_2. The condition in the proposition ensures that the resulting set of interactions I_3 is a poset.

Definition 3.9 Let $h_1 : M_0 \to M_1$ be a message renaming, let L_0 and L_1 be sets of lifelines, such that $L_0 \subseteq L_1$, let T be a set of L_0-typed interactions over M_0, and let $\mathcal{I}$ be an implementation of T by $(L, \mathcal{S})$ for some given sets of lifelines L and of diagrams $\mathcal{S}$ over a set of messages M. Then, we define T' and $\mathcal{I}'$, called the extensions of T and $\mathcal{I}$ with respect to h_1 and L_1, as follows. T' is the set of L_1-typed interactions over M_1:

$$T' = \{(l_1, l'_1, h_1(m))/(l_1, l'_1, m) \in T_0$$

and $\mathcal{I}'$ is the implementation of T' by $(L' + (L_1 \setminus L_0), \mathcal{S})$ be defined as follows:

- For every $l \in L_0$, $\mathcal{I}'_{Lines}(l) = \mathcal{I}_{Lines}(l)$ and for every $l \in (L_1 \setminus L_0)$, $\mathcal{I}'_{Lines}(l) = \{l\}$
- For every $(l_1, l'_1, h_1(m)) \in T'$, $\mathcal{I}'_{Mess}(l_1, l'_1, h_1(m)) = \mathcal{I}_{Mess}(l_1, l'_1, m)$

Proposition 3.10 *Let $S_j = (L_j, Loc^j_{L_j}, I_j)$ be sequence diagrams over M_j, for $j = 0, 1$, respectively. Let $h_1 : M_0 \to M_1$ be a message renaming, $i_1 : S_0 \to S_1$ an*

h_1-based embedding, $\mathcal{I}$ an implementation of a set T of L_0-typed interactions over M_0 by $(L, \mathcal{S})$, and let T' and $\mathcal{I}'$ be the extensions of T and $\mathcal{I}$ with respect to h_1. Then, the diagram $\mathcal{I}(S_0)$ is embedded into $\mathcal{I}'(S_1)$.

Proof. [**Sketch**] According to the definition of $\mathcal{I}'$, we have that if $\langle \varepsilon'_1, \ldots, \varepsilon'_n \rangle$ is an ordered sequence of the interactions in I_1 then $\mathcal{I}'(S_1) = \mathcal{I}'(\varepsilon_1) + \ldots + \mathcal{I}'(\varepsilon_n)$.

On the other hand, by definition, we know that for every interaction $(l_0, l'_0, m_0) \in S_0$, $\mathcal{I}(l_0, l'_0, m_0) = \mathcal{I}'(i_1(l_0), i_1(l'_0), h_1(m_0))$. This means that $\mathcal{I}(S_0)$ is the sum, $\mathcal{I}'(S_1) = \mathcal{I}'(\varepsilon_{j_1}) + \ldots + \mathcal{I}'(\varepsilon j_m)$, of a subset of $\{\mathcal{I}'(\varepsilon_1), \ldots, \mathcal{I}'(\varepsilon_n)\}$. This directly implies the embedding. □

Before stating the extension property for sequence diagrams, which is a direct consequence of the previous two propositions, we must first define our notion of consistency:

Definition 3.11 An embedding $i_1 : S_0 \to S_1$ and a transformation $t_2 : S_0 \Longrightarrow S'_2$, with $t_2 = (\mathcal{I}_2, i_2)$, are consistent if i'_2 and i_2 are consistent, where i'_2 is the embedding whose existence is proved in the previous proposition.

Theorem 3.12 *(Extension for basic sequence diagrams)*

Let $S_j = (L_j, Loc^j_{L_j}, I_j)$, be sequence diagrams over M_j, for $j = 0, 1, 2$, respectively. Let $h_1 : M_0 \to M_1$ be a message renaming, $i_1 : S_0 \to S_1$ an h_1-based embedding, and $t_2 : S_0 \Longrightarrow S_2$, $t_2 = (\mathcal{I}_2, i_2)$ a transformation which is consistent with i_1.

Then, there is a diagram S_3, such that S_2 is embedded into S_3 and S_3 is a transformation of S_1, according to Fig. 9).

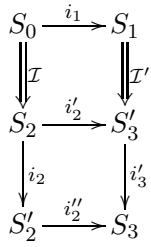

Fig. 9. Extension of diagram transformations

Then, S_3 embeds S_2 via $i''_2 \circ i'_2$ and refines S_1 via $t_1 = (\mathcal{I}', i'_3)$

Now, to define parallel extension we first have to define our notion of parallel consistency. A very simple, but restrictive notion which works in many cases would be to ask the roles involved to be disjoint (e.g. [10]). Unfortunately, this restriction would be too strong here, as shown in the example in section 5. In our case, we will require that the implementations underlying the given transformations coincide on the shared elements.

Definition 3.13 (Parallel consistency) Given a family of transformations $\{t_k \colon S_k \Longrightarrow S'_k\}_{k \in K}$, with $t_k = (\mathcal{I}_k, i_k)$ for each k, and a family of embeddings

$\{i_k\colon S_k \to S\}_{k\in K}$ for some finite set K We say that the family of transformations t_j is parallel consistent with respect to this family of embeddings if for every $k1, k2$, the following conditions hold:

(i) If $l_{k1} \in L_{k1}$ and $l_{k2} \in L_{k2}$ and $i_{k1}(l_{k1}) = i_{k2}(l_{k2})$, then $\mathcal{I}_{k1}(l_{k1}) = \mathcal{I}_{k2}(l_{k2})$.

(ii) Given interactions $int_{k1} \in I_{k1}$ and $(int_{k2} \in I_{k2}$. If $i_{k1}(int_{k1}) = i_{k2}(int_{k2})$, then $\mathcal{I}_{k1}(int_{k1}) = \mathcal{I}_{k2}(int_{k2})$.

(iii) Given interactions $int_{k1}, int1'_{k1} \in I_{k1}$ and $int_{k2}, int'_{k2} \in I_{k2}$ where $i''_{k1}(int_{k1}) = i''_{k2}(int_{k2})$ and $i''_{k1}(int'_{k1}) = i''_{k2}(int'_{k2})$. If $j_{k1}(int_{k1}) \leq j_{k1}(int'_{k1})$ and $j_{k2}(int'_{k2}) \leq j_{k2}(int_{k2})$ then $i''_{k1}(int_{k1}) = i''_{k1}(int'_{k1})$, where i''_{k1} and i''_{k2} are the embeddings defined in Fig. 10

Theorem 3.14 *(Parallel extension for basic sequence diagrams) Given a family of transformations $\{t_k\colon S_k \Longrightarrow S'_k\}_{k\in K}$, with $t_k = (\mathcal{I}_k, j_k)$ for each k, and a family of embeddings $\{i_k\colon S_k \to S\}_{k\in K}$ for some finite set K, such that they are paralled consistent and each t_k is consistent with respect to each i_k, we can define their parallel extension $t'\colon S \Rightarrow S'$ with embeddings $i'_k : S_k \to S'$ for each $k \in K$, according to Fig. 10.*

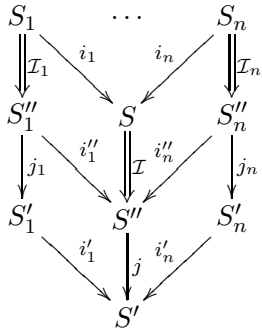

Fig. 10. Parallel extension

Proof. **[Sketch]** The construction of the parallel extension is similar to the construction of the extensions, but slightly more complicated. First, we have to build the implementation $\mathcal{I}$ as the union of the implementations $\mathcal{I}_k$. Conditions 1. and 2. of parallel consistency ensure that such a union exist. Then, we have to check that the diagram S'' built by applying the implementation $\mathcal{I}$ to S embeds all the diagrams S''_k. This is straightforward. Finally, we have to ensure that we can build a parallel extension in the special case that the transformations involved are just embeddings. In this case, the key property needed to ensure that the resulting set of interactions is partially ordered is condition 3. from parallel consistency. □

To end, we have to prove that our notion of parallel extension satisfies the vertical composition property.

Proposition 3.15 *If $t : S \Longrightarrow S'$ is the parallel extension of $t_k : S_k \Longrightarrow S'_k$, with $1 \leq k \leq n$, with respect to $i_k : S_k \to S$ and $t' : S' \Longrightarrow S''$ is the parallel extension*

of $t'_k : S'_k \Longrightarrow S''_k$, *with* $1 \leq k \leq n$, *with respect to* $i'_k : S'_k \to S'$, *as in Fig. 11, then* $t' \circ t : S \Longrightarrow S''$ *is the parallel extension of* $t'_k \circ t_k : S_k \Rightarrow S''_k$ *with respect to* $e'_k \circ e_k : S_k \to S$.

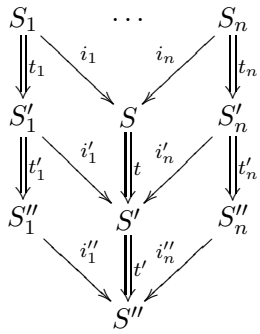

Fig. 11. Vertical composition of parallel extensions

Proof. **[Sketch]** The proof is just routine, using the construction of the parallel extensions. □

As a consequence, we have shown that basic sequence diagrams are a component framework:

Theorem 3.16 *The class of basic sequence diagrams together with the notions introduced above of embedding, transformation, consistency, and parallel consistency, and together with the extension and parallel extension constructions form a component framework.*

4 Components and connectors for sequence diagrams

In this section we will extend the previous instantiation of our generic framework to cover most of UML2 sequence diagrams. In particular, UML2 sequence diagrams are combinations of basic sequence diagrams using a given family of operators. Here we will not cover all the possible operators, but just the ones that we consider more important, for two reasons. On one hand,because of space limitations. On the other hand because we think that considering all kinds of details and variations associated to the given class of diagrams will make the paper too boring and would not add any interesting insights to the problem. Actually, as we will see, our definitions of embedding and transformation are extended in a uniform manner to all the operations considered. In this sense, the reader could easily imagine how to provide similar extensions to other operators.

More precisely,the operators that we will consider in the paper are **seq**, **alt**, **par** and **loop**. Informally, one may describe the meaning of this operators in terms of the "execution" of the diagrams involved. In particular, $\mathbf{seq}(S_1, S_2)$ specifies the (weak) sequencing of diagrams S_1 and S_2. Executing $\mathbf{alt}(S_1, S_2)$ means that either S_1 or S_2 are executed. $\mathbf{par}(S_1, S_2)$ specifies the parallel execution of S_1 and

S_2. Finally, $\mathbf{loop}(S)$ specifies that the interactions defined in S may be repeatedly executed.

UML2 provides a graphical notation for these operators that we will not use. Instead, we will use the simple textual notation presented in the paragraph above. More precisely, we can define general sequence diagrams as follows:

$$S \to B \mid \mathbf{seq}(S_1, S_2) \mid \mathbf{alt}(S_1, S_2) \mid \mathbf{par}(S_1, S_2) \mid \mathbf{loop}(S)$$

where S, S_1, S_2 are metavariables to denote sequence diagrams and B is a metavariable to denote a basic sequence diagram.

We have defined the notions of embedding and transformation for general sequence diagrams as the simplest extensions of the corresponding notions for basic sequence diagrams. In particular, embedding and transformation relations will only be defined between diagrams sharing the same structure. This means, for instance, that a diagram S_1 will be considered embedded in S_2 if, on one hand, S_1 and S_2 are defined in terms of exactly the same combination of operators and, on the other hand, each basic subdiagram in S_1 is embedded in the corresponding subdiagram of S_2. Actually, as we will see below, we will keep this structure for denoting the embeddings. The case of transformations is similar. We could have considered a more general definition. For instance, it may seem quite reasonable that S_1 is embedded in $\mathbf{seq}(S_1, S_2)$. However, according to our definitions presented below S_1 is not embedded in $\mathbf{seq}(S_1, S_2)$. There are several reasons for this. The first one is that, if we want that our definitions of embedding and transformation take into account this kind of situations, we would need to provide the formal semantics of sequence diagrams beforehand. Unfortunately, there is no such semantics for UML2 sequence diagrams and is not the aim of this paper to provide such a formal definition. Conversely, it is the aim of this paper to show that our approach can be also applied to semiformal modelling techniques. In addition, we consider that, from a methodological standpoint, it is adequate to require that the interfaces and the body of a component or a connector share the same structure.

Definition 4.1 Given two sequence diagrams S and S' we say that $i : S \to S'$ is an embedding if one of the following conditions hold:

(i) S and S' are basic sequence diagrams and $i : S \to S'$ is an embedding of basic sequence diagrams.

(ii) $S = \mathbf{op}(S_1, S_2)$, $S' = \mathbf{op}(S'_1, S'_2)$ and $i = \mathbf{op}(i_1, i_2)$, where $\mathbf{op} = \mathbf{seq}$ or $\mathbf{op} = \mathbf{alt}$ or $\mathbf{op} = \mathbf{par}$, and we have that $i_1 : S_1 \to S'_1$ and $i_2 : S_2 \to S'_2$ are embeddings.

(iii) $S = \mathbf{loop}(S_1)$, $S' = \mathbf{loop}(S'_1)$ and $i = \mathbf{loop}(i_1)$, and we have that $i_1 : S_1 \to S'_1$ is an embedding.

(iv) There are no other embeddings between sequence diagrams

The definition of transformations is similar:

Definition 4.2 Given two sequence diagrams S and S' we say that $t : S \Longrightarrow S'$ is a transformation if one of the following conditions hold:

(i) S and S' are basic sequence diagrams and $t : S \Longrightarrow S'$ is a transformation of basic sequence diagrams.

(ii) $S = \mathbf{op}(S_1, S_2)$, $S' = \mathbf{op}(S'_1, S'_2)$ and $t = \mathbf{op}(t_1, t_2)$, where $\mathbf{op} = \mathbf{seq}$ or $\mathbf{op} = \mathbf{alt}$ or $\mathbf{op} = \mathbf{par}$, and we have that $t_1 : S_1 \Longrightarrow S'_1$ and $t_2 : S_2 \Longrightarrow S'_2$ are transformations.

(iii) $S = \mathbf{loop}(S_1)$, $S' = \mathbf{loop}(S'_1)$ and $t = \mathbf{loop}(t_1)$, and we have that $t_1 : S_1 \Longrightarrow S'_1$ is a transformation.

(iv) There are no other transformations between sequence diagrams

It should be obvious that the requirements for embeddings and transformations are satisfied. In particular, both embeddings and transformations are closed under composition, embeddings are special cases of transformations, and the identity is an embedding.

We can define in a similarly the properties of consistency and parallel consistency:

Definition 4.3 An embedding $i : S \to S'$ and a transformation $t : S \Longrightarrow S''$ are consistent if and only if the following conditions hold:

(i) If S, S' and S'' are basic sequence diagrams then i and t must satisfy the consistency condition for basic sequence diagrams.

(ii) If $i = \mathbf{op}(i_1, i_2)$ and $t = \mathbf{op}(t_1, t_2)$, where $\mathbf{op} = \mathbf{seq}$ or $\mathbf{op} = \mathbf{alt}$ or $\mathbf{op} = \mathbf{par}$, then i_1 and t_1, and i_1 and t_1 must be consistent.

(iii) If $i = \mathbf{loop}(i_1)$ and $t = \mathbf{loop}(t_1)$, then i_1 and t_1 must be consistent.

Definition 4.4 A family of embeddings $\{i_k : S_k \to S\}_{k \in K}$ and a family of transformations $\{t_k : S_k \Longrightarrow S'_k\}_{k \in K}$ are parallel consistent if and only if the following conditions hold:

(i) If each S_k, S'_k and S are basic sequence diagrams then $\{i_k\}_{k \in K}$ and $\{t_k\}_{k \in K}$ must satisfy the parallel consistency condition for basic sequence diagrams.

(ii) If each $i_k = \mathbf{op}(i1_k, i2_k)$ and $t = \mathbf{op}(t1_k, t2_k)$, where $\mathbf{op} = \mathbf{seq}$ or $\mathbf{op} = \mathbf{alt}$ or $\mathbf{op} = \mathbf{par}$, then $\{i1_k\}_{k \in K}$ and $\{t1_k\}_{k \in K}$, and $\{i2_k\}_{k \in K}$ and $\{t2_k\}_{k \in K}$ must be parallel consistent.

(iii) If each $i_k = \mathbf{loop}(i1_k)$ and $t_k = \mathbf{loop}(t1_k)$, then $\{t1_k\}_{k \in K}$, and $\{i2_k\}_{k \in K}$ must be parallel consistent.

To finish proving that sequence diagrams are a component framework we have to prove that they satisfy the extension and parallel extension properties:

Theorem 4.5 *The class of sequence diagrams satisfy the extension and paralle extension properties*

Proof. **[Sketch]** Let us show the existence of extensions. We proceed by induction. Given consistent $i : S \to S'$ and $t : S \Longrightarrow S''$, the case where S, S' and S'' are

basic diagrams has already be proven in Theorem 3.12. If $i = \mathbf{seq}(i_1, i_2)$ and $t = \mathbf{seq}(t_1, t_2)$ then, by induction, we know that we can define the extensions of S_1, S_2 of i_1 with respect to t_1 and i_2 with respect to t_2, respectively. Then $S'' = \mathbf{seq}(S_1, S_2)$ is the extension of i with respect to t. The case of the other operators is similar. The construction of parallel extension is also similar. □

Moreover, the vertical composition property for parallel extensions for basic diagrams trivially implies this property for sequence diagrams. As a consequence, we have:

Theorem 4.6 *The class of sequence diagrams together with the notions introduced above of embedding, transformation, consistency, and parallel consistency, and together with the extension and parallel extension constructions form a component framework.*

5 An example

In this section we will present a small example of the use of this kind of component system. For brevity we will only use sequence diagrams, which means that the corresponding class diagrams will remain implicit. The example describes a lift system including just one lift. However, a system including several lifts would not be difficult to describe using the same components, but a more complex connector.

We consider that a lift system can be built (at a certain level of abstraction) out of three kinds of components: the elevators themselves, including the doors and the engines to move the lift; the buttons that are located inside the lift; and the set of buttons which are located in each floor (for simplicity we will consider that there is only one button per floor and not two, as it usually happens). Also for simplicity, will only describe the normal scenario describing the system, i.e. we will not consider abnormal situations. Now, let us model these components.

The body of the elevator can be described by the diagram in figure 12

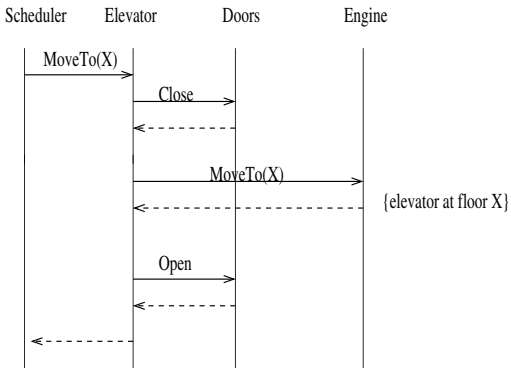

Fig. 12. Elevator Body

This diagram describes the following scenario. Someone, which we have called the scheduler, tells the elevator to move to floor X. This causes the doors to close

and when they are closed (an ack is received), the elevator sends a message to the engine to move to floor X. When the elevator is at floor X, the doors open and the scheduler is acknowledged that the operation has been completed. Now for the interface there are details that can be abstracted from this diagram. In particular, for the use (as a component) of the elevator, we do not need to know about how doors are opened or how the engine works. So, the elevator interface is just the diagram in figure 13.

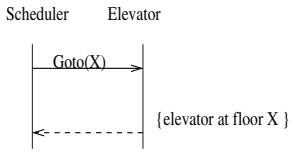

Fig. 13. Elevator Interface

Obviously, this interface is refined by the body of the component (actually the transformation is just an embedding). Now, the body of the component describing the buttons inside the lift is presented in figure 14.

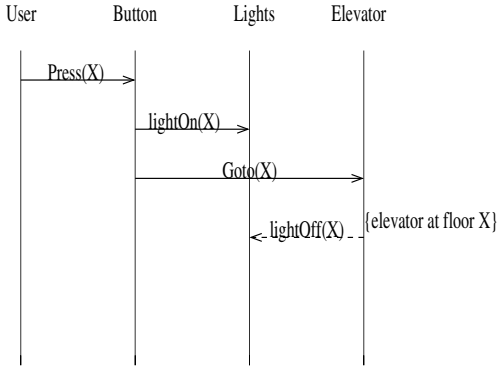

Fig. 14. Buttons Body

In particular, when a user presses the button to go to floor X, the light associated to that button is switched on and a message is sent to the elevator to move to floor X. When the elevator is at that floor the light will be switched off. We have considered that it is the elevator who sends the message to switch off the light. Instead, we may have considered that when elevator is at floor X, it will send an acknowledgement to the button object who, then, will switch off the light. Now, according to this body diagram, the interface describing the connection to the interacting components can be seen in figure 14.

Again, the refinement between this interface and the body of the component is just an embedding. The component associated to the set of buttons which are located in each floor could be described exactly in the same way as the previous one. Note that this would not have been true if there would be two buttons per floor.

Now, if we want to build a lift system, including just one lift, we need to connect these three components. The body of this connector would consist of three diagrams,

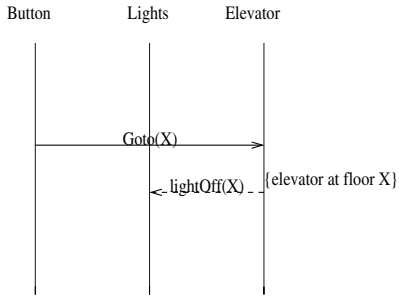

Fig. 15. Buttons Interface

where two of them would be almost identical. In particular the first diagram (see figure 16) describe that, when a request is received from some set of buttons (for instance the cabin buttons located inside the elevator, C-buttons), this request is received by a scheduler (which will probably store the request in some queue). A similar diagram would be needed to describe the situation when the request is received from the buttons located in the floors. We have not shown this diagram. The third diagram (see figure 17) describes that, when the first request to serve refers to floor X, the scheduler sends a message to the elevator to go to that floor. When the elevator acknowledges that the elevator is at floor X, then the scheduler asks the two sets of buttons to switch off the lights corresponding to that floor.

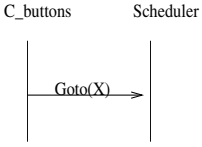

Fig. 16. Connector Body 1

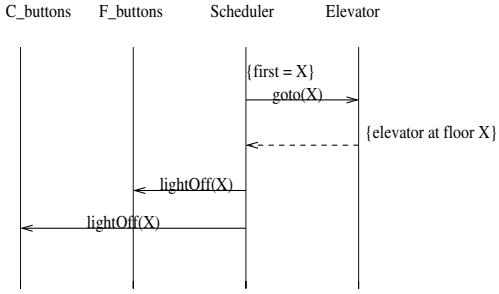

Fig. 17. Connector Body 2

Now, the connector would have three interfaces, the first two which are again almost identical would consist of two diagrams. The first one would coincide with the first body diagram (figure 16). The second one, see figure 18, describes the interaction for switching off the lights upon arrival at a given floor. The third interface describes the interaction with the elevator and would be identical to the elevator interface (figure 13).

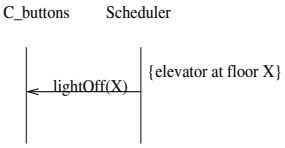

Fig. 18. Connector Interface 2

The composition of the connector with the three components would provide the expected global specification of the lift system. The connection of the elevator interface with the corresponding connector's interface is trivial, since both interfaces are equal. In the case of the buttons, the connection must be made via a transformation. In particular, we would need to say that the C-buttons lifeline in the connector's interface is implemented in terms of the lifelines Button and Lights from the button component; and, similarly, the lightOff(X) message is implemented by a diagram that includes only one interaction, consisting of sending the message lightOff(X) from the Scheduler to the Lights lifeline.

6 Conclusion

In this paper, using the generic framework presented in previous work (e.g., [3,4,5], we have shown how to define architectural connectors and components, as introduced in [1], for UML2 sequence diagrams.

In particular, first we studied how we can define connectors and components in the case of basic sequence diagrams, defining and studying the notions of embedding and transformation, which are needed for the application of our generic approach. Then we extended the constructions to the class of general sequence diagrams.

This work may be considered a first step in the definition of these concepts for full UML. In this sense, we think that the ideas presented in this paper could be useful when dealing with the rest of UML diagrams. Actually, as said in the introduction, based on our results we have defined stereotypes in UML for our components and connectors and, to provide some tool support, we have implemented a plugin for the tool Visual Paradigm. Unfortunately, due to lack of space, it was impossible to present this work in this paper, but can be found in [11]

Acknowledgement

This work is partially supported by the Spanish project GRAMMARS (TIN2004-07925-C03-01). The stay of Sonia Pérez Lovelle in Barcelona was supported by the European Alfa Net CORDIAL II (AML/B7-311-97/0666/II-0021-FA).

References

[1] R. Allen, D. Garlan. A Formal Basis for Architectural Connection. In *ACM TOSEM '97*, pp. 213–249.

[2] J. Cheesman, J. Daniels. *UML Components*. Addison-Wesley, 2001.

[3] H. Ehrig, F. Orejas, B. Braatz, M. Klein, M. Piirainen. A Generic Component Framework for System Modeling. In *Proc. FASE 2002*, Springer LNCS 2306 (2002), pp. 33–48.

[4] H. Ehrig, J.Padberg, B. Braatz, M. Klein, M. Piirainen, F. Orejas, S. Perez, E. Pino. A Generic Framework for Connector Architectures based on Components and Transformations. Proc. FESCA 2004, Barcelona.

[5] H. Ehrig, M. Klein, F. Orejas J.Padberg, S. Perez, E. Pino. A Generic Approach to Connector Architectures. Barcelona and Berlin (2007). Submitted for publication.

[6] J.L. Fiadero, A. Lopes Semantics of Architectural Connectors. *Proc TAPSOFT '97*, Springer LNCS 1214 (1997), pp. 505–519.

[7] David Garlan, Shang-Wen Cheng, Andrew Kompanek: Reconciling the needs of architectural description with object-modeling notations. Sci. Comput. Program. 44(1): 23-49 (2002)

[8] C. A. R. Hoare: Communicating Sequential Processes Prentice-Hall 1985

[9] James ivers, Paul Clements, David Garlan, Robert Nord, Bradley Schmerl, Jaime Rodrigo Oviedo Silva: Documenting Component and Connector Views with UML 2.0. Carnegie-Mellon Univ. Tech. Report CMU/SEI-2004-TR-008, 2004.

[10] Fernando Orejas, Sonia Perez: Towards Architectural Connectors for UML, in *Formal Methods in Software and Systems Modeling*, Springer Lecture Notes in Computer Science 3393 (2005) pp. 352–369

[11] Sonia Pérez Lovelle. Uso y extensin de UML para la especificacin y chequeo de consistencia en el empleo de la arquitectura de componentes genricos. PhD Thesis, La Habana 2007.

[12] J. Rumbaugh, I. Jacobson, G. Booch. The Unified Modeling Language Reference Manual. Addison Wesley (1999).

[13] Mary Shaw and David Garlan Software Architecture: Perspectives on an Emerging Discipline Prentice Hall, 1996

[14] A.M. Zaremski, J.M. Wing. Specification Matching of Software Components. In *ACM TOSEM '97*, pp. 333–369.