Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 160 (2006) 3–14
www.elsevier.com/locate/entcs

# Reo Based Interaction Model

Silvia Amaro[1,4]

*Dpto. de C. de la Computación*
*National University of Comahue*
*Argentina*

Ernesto Pimentel[2,5]

*Dpto. de Lenguajes y Ciencias de la Computación*
*University of Málaga*
*Spain*

Ana M. Roldan[3,5]

*Dpto. de Ing. Electrónica y Sist. Informáticos*
*University of Huelva*
*Spain*

**Abstract**

In Component-based Software Development the integration of possibly heterogeneous and distributed components together to form a single application require mechanisms for controlling and managing the interactions among the active entities. Coordination models and languages offer a solution to this problem. In this context we propose the use of Reo, a channel-based coordination model, to specify the interactive behavior of software components. In particular, we define a way to complement interface description languages for describing components, in such a way that the information about which are the services provided by a component is extended by giving details on how these services should be used. Our aim is to define an interaction description language based on Reo for component coordination.

*Keywords:* Components, formal methods, coordination, process algebra, connector.

## 1 Introduction

In Component-based Software Development (CBSD) the integration of possibly heterogeneous and distributed components together to form a single application

[1] Email: samaro@uncoma.edu.ar
[2] Email: ernesto@lcc.uma.es
[3] Email: amroldan@diesia.uhu.es
[4] The work of Silvia Amaro has been partially supported by CYTED, proyect VII-J-RITOS2
[5] The work of Ana M. Roldán and E. Pimentel has been partially supported by the Project TINC2004-07943-C04-01 funded by the Spanish Ministry of Education and Science


doi:10.1016/j.entcs.2006.05.012

require mechanisms for controlling and managing the interactions among the active entities. In spite of its relatively recent birth, a lot of activities are being devoted to CBSD both in the academic and in the industrial world. The reason of this growing interest is the need of systematically developing open systems and "plug-and-play" reusable applications, which has led to the concept of "commercial off-the-shelf" (COTS) components.

With the increasing use of distributed systems and COTS, interoperability is a major issue to consider. Commercial component models and platforms (CORBA, DCOM, EJB, .NET) attends interoperability from a syntactic point of view using Interface Description Languages (IDL). They allow the interoperation of heterogeneous components based on syntactic agreements. However the sort of interoperability attended by using IDL's is not enough in large systems, where the information given by interfaces, that is the knowledge of the services offered by components, and in some cases the services required from other components in run-time is not enough to guarantee that they will suitably interoperate. Indeed, at the protocol level, mismatches may also occur because of blocking conditions and the ordering of exchanged messages, that is, because of differences in the component behaviors. In fact, compatibility checkings at protocol level require the solution of coordination and synchronization problems, to ensure that the restrictions imposed on components interactions when communicating are preserved and their communication is deadlock free.

In general, the use of IDL descriptions during run-time is quite limited. They are mainly used to discover services and to dynamically build service calls. However, there are no mechanisms currently in place to deal with automatic compatibility checks or dynamic component adaption which are among the most commonly required facilities for building component-based applications in open and independently extensive systems. To overcome such a limitation, several proposals have been put forward in order to enhance component interfaces. In [10] Doug Lea proposes the use of a protocol specification language (PSL) to describe the protocols associated to component's methods. It is a very expressive extension of CORBA IDL based on logical and temporal rules, but does not take into account the services a component may need from other components, neither it is supported by proving tools. The approach formalized by Yellin and Strom [15] for describing component service protocols using finite state machines, although considering both services offered and required by components, does not support multi-party interactions. Moreover the simplicity that allows the easy checking also makes it too rigid and unexpressive for general usage in open and distributed environments. Bastide et al. [4] use Petri nets to describe the behavior of components in CORBA, but this approach inherits some of the limitations imposed by the Petri nets notation: the lack of the modularity and scalability of the specifications.

This paper addresses the problem of interoperability at protocol level, exploring the capability of the coordination model *Reo*, for specifying the interaction behavior of software components. Reo [1] is a channel-based coordination model which enforces the use of connectors for the coordination of concurrent processes or com-

ponent instances in a component-based system. Indeed our aim is to propose this model to enhance components interfaces with a description of an abstract component interaction protocol in a similar way as behavioral types [12] or role-based representations [7,8]. Intuitively, when using this model compatibility checkings will depend on the connector considered for the composition. Moreover, as the connector adds its own behavior to the resulting application, we are interested in analyzing how the composition of the same set of components is affected by selecting different connectors.

In Reo complex connectors are constructed compositionally, out of simpler ones, using its *join* operator, and hiding the internal topology of the resulting connector. This yields a connector with a number of input and output ports which can be used by other entities to interact with and through the connector. As our model is not concerned with the internal topology of connectors, but in the connection ends a connector offers to the environment and its observable behavior, we consider a connector defined by a set of input and output ends, the possible configurations in which it can be, and a labelled transition relation defining its behavior. With this in mind, in this paper we address the problem of generating the labelled transition relation indicated for a given connector. We present an algorithm that takes as input a coordination protocol given by a set of input and output ends and a constraint automata, and produces the labelled transition relation defining the behavior of a connector.

The rest of the paper is organized as follows. In section 2 we give an introduction to Reo. Section 3 is devoted to the interaction model, its semantics and the corresponding calculi to encapsulate the model. We also give an algorithm for the generation of the transitions giving the behavior of the connector, from its constraint automata. In section 4 an illustrative example is presented, showing the application of the model and the algorithm. Finally, we give some concluding remarks and future work.

## 2 An introduction to Reo

Reo [1] is a channel-based coordination model defined in terms of communication primitives acting on connectors which are constructed as a combination of different kinds of channels. The channel composition mechanism in addition to the great diversity of channel types with semantics different from the traditional ones allows the construction of many different connectors imposing very interesting coordination patterns. For example, the connector Exclusive Router showed in figure 1a) enables the flow of data items from its input end $a$ to one of its output ends $b$ or $c$ (when both $b$ and $c$ are willing for a data item a non deterministic decision take place). This connector is the result of composing five synchronous channels, two Lossy Synchronous channels and a synchronous Drain. A Lossy Synchronous channel is a synchronous channel with a loosing policy for items written on its input end when the output end is not waiting for it; and a Synchronous Drain channel is a synchronous channel with two input ends which has an important synchronization

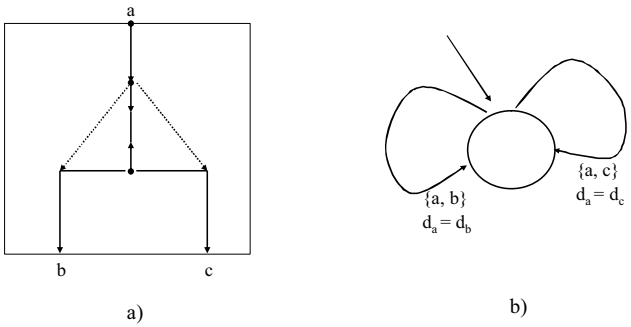


Fig. 1. Connector Exclusive Router and constraint automata

function.

The formal semantic for Reo is based on relations on timed data streams [2]. Indeed timed data streams, which are pairs consisting of a data stream and a time stream models the potential behavior of connector ends, and relations over them express which combinations of timed data streams are mutually consistent. A timed data stream is a pair $\langle \alpha, a \rangle$ consisting of a data stream $\alpha$ and a time stream $a$, in which the time stream $a$ specifies for each $n \geqq 0$ the time moment $a(n)$ at which the $n$th data element $\alpha(n)$ is being input or output. Its derivative $\langle \alpha', a' \rangle$ is used to indicate changes in time. In this context is possible to define the language induced by a Reo connector in terms of the TDSs representing its input and output ends. The coinduction reasoning principle applied to the TDS calculus is sufficiently powerfull for the proof of certain formal properties over connectors, such as expressiveness and connector equivalence.

The operational model for the behavior of Reo connectors is based on Constraint Automata and was introduced by Arbab et. al in [3]. A constraint automata describes the TDS language induced by Reo connector networks. A *constraint automata* — over *Data*, a finite set of data that can be sent and received via channels— is a tuple $\mathcal{A} = (Q, \mathcal{N}, \longrightarrow, Q_0)$ where $\mathcal{Q}$ is a finite set of states, $\mathcal{N}$ a finite set of nodes, $\longrightarrow$ is a finite subset of $\mathcal{Q} \times (2^{\mathcal{N}} \times DC) \times \mathcal{Q}$ called the transition relation —DC denotes the set of data constraints over $\mathcal{N}$—, and $\mathcal{Q}_{\prime} \subseteq \mathcal{Q}$ a nonempty set of initial states. A transition $q \xrightarrow{N,g} p \in \longrightarrow$ requires $N \neq 0$ and $g \in DC(N)$ to be satisfiable. Figure 1b) shows the constraint automata corresponding to the connector Exclusive Router introduced before. The *join* operator, and hiding operation used in Reo for the construction of complex connectors have their counterparts in the model of constraint automata. Though the constraint automata for a complex con-

nector is obtained from the constraint automata of its constituents through product and hiding of internal nodes.

# 3 The interaction model

When using channel-based coordination models the framework evolves by means of performing communication actions over input or output ends of channels to which the coordinated components are connected. In the case of Reo, the communication actions are performed over the input/output ends of a connector, then the interaction model will be parametrized with respect to the connector being considered.

## *3.1 The Calculus*

For the specification of components interaction protocols we define a process algebra $\mathcal{R}$ based on the communication primitives of Reo. We consider a set $\mathcal{I}$ of input ends, a set $\mathcal{O}$ of output ends, and the basic actions to insert an item in a connector ($write$), to remove an item from the connector ($take$) and to capture an item without removing it ($read$). Agents in $\mathcal{R}$ are constructed by means of the prefix operator, the nondeterministic choice and the parallel composition. Formally, the syntax of $\mathcal{R}$ is defined as follows:

$$P ::= 0 \mid A.P \mid P + P \mid P \parallel P \mid recX.P$$
$$A ::= wr(c, v) \mid tk(c, [v]) \mid rd(c, [v])$$

where 0 denotes the empty process and $c \in \mathcal{I} \cup \mathcal{O}$ denotes an input or output end of a connector. The prefixes $wr$, $tk$ and $rd$ are shorthand for the basic operations $write$, $take$ and $read$ respectively. Note that in output operations the variable is optional, if it is not specified the operation succeed when any data item is available for taking (or reading) and it is removed through the specified connector end.

As in Reo communication is possible only in presence of a connector, in order to define the operational semantics of $\mathcal{R}$ we must consider the semantics of the selected connector. We consider a connector C defined by a tuple $\langle \mathcal{I}_{\mathrm{C}}, \mathcal{O}_{\mathrm{C}}, \Sigma_{\mathrm{C}}, \longmapsto_C \rangle$, where $\mathcal{I}_{\mathrm{C}}, \mathcal{O}_{\mathrm{C}}$ represent the input ends set and output ends set of connector C respectively, $\Sigma_{\mathrm{C}}$ is the set of states, that is the possible configurations of the connector, and $\longmapsto_{\mathrm{C}} \subseteq (\Sigma_{\mathrm{C}} \times MAct) \times MAct \times (\Sigma_{\mathrm{C}} \times MAct)$ represents the labelled transition relation defining the connector behavior. $MAct$ denotes the multiset of communication actions. When $(\langle \mathrm{C}, act \rangle, act_1, \langle \mathrm{C}', act_2 \rangle) \in \longmapsto_C$ we will write

$$\langle \mathrm{C}, act \rangle \overset{act_1}{\longmapsto}_{\mathrm{C}} \langle \mathrm{C}', act_2 \rangle$$

with the following intuitive interpretation: $act$ denotes the set of actions which applied in parallel over the ends of the connector may produce a progress over it producing eventually a state change. The set $act_1$ denotes the actions actually applied, and $act_2$ represents pending actions in any end of the connector. Pending actions are actions $write$ or $take$ that, in presence of a synchronous behavior, remain pending when applied in parallel with $read$ actions. These multisets must respond to the relation $act = act_1 \uplus act_2$. When deduced from the context, we will omit the

| | | | |
|---|---|---|---|
| $(1)_{\mathcal{R}}$ | $act \cdot P \xrightarrow{act} P$ | $(4)_{\mathcal{R}}$ | $\dfrac{P_1 \xrightarrow{act_1} P_1' \; P_2 \xrightarrow{act_2} P_2'}{P_1 \parallel P_2 \xrightarrow{act_1 \uplus act_2} P_1' \parallel P_2'}$ |
| $(2)_{\mathcal{R}}$ | $\dfrac{P_1 \xrightarrow{act} P_1'}{P_1 + P_2 \xrightarrow{act} P_1'}$ | $(5)_{\mathcal{R}}$ | $\dfrac{P \xrightarrow{act} P' \; \langle \mathrm{C}, act\rangle \xmapsto{act}_{\mathrm{C}} \langle \mathrm{C}', \emptyset\rangle}{\langle P, \mathrm{C}\rangle \longrightarrow \langle P', \mathrm{C}'\rangle}$ |
| $(3)_{\mathcal{R}}$ | $\dfrac{P_1 \xrightarrow{act} P_1'}{P_1 \parallel P_2 \xrightarrow{act} P_1' \parallel P_2}$ | $(6)_{\mathcal{R}}$ | $\dfrac{P_1 \xrightarrow{act_1} P_1' \; P_2 \xrightarrow{act_2} P_2' \; \langle \mathrm{C}, act\rangle \xmapsto{act_1}_{\mathrm{C}} \langle \mathrm{C}', act_2\rangle}{\langle P_1 \parallel P_2, \mathrm{C}\rangle \longrightarrow \langle P_1' \parallel P_2, \mathrm{C}'\rangle}$ |

Table 1
Transition System for $\mathcal{R}$

subindex C when referring to the sets $\mathcal{I}$, $\mathcal{O}$, and $\Sigma$. The rules giving the connector behavior will be generated from its corresponding constraint automata, using the algorithm introduce in the next subsection.

The operational semantics of $\mathcal{R}$ depends on the connector considered. Formally, given a connector C with a behavior defined via a labelled transition relation $\xmapsto{\alpha}_{\mathrm{C}}$, we define the transition system $\langle \mathcal{R}, \mathrm{C}, \longrightarrow_{\mathrm{C}}\rangle$, where $\mathcal{R}$ is the set of programs described in the process algebra, C is the considered connector and $\longrightarrow_{\mathrm{C}} \subseteq (\mathcal{R} \times \mathrm{C}) \times (\mathcal{R} \times \mathrm{C})$ the transition relation defined by rules $(5)_{\mathcal{R}}$ and $(6)_{\mathcal{R}}$ of table 1. Note that the definition of $\longrightarrow_{\mathrm{C}}$ depends on the auxiliary labelled transition system $\langle \mathcal{R}, Act, \longrightarrow\rangle$ where $\longrightarrow \subseteq \mathcal{R} \times Act \times \mathcal{R}$ is the transition relation defined by rules $(1)_{\mathcal{R}}$ to $(4)_{\mathcal{R}}$. There are no rules for recursion, its semantics is defined by the structural axiom $recX.P \equiv P[recX.P/X]$. We also consider both systems close with respect to the structural axioms for choice and parallel operators.

In the process of composing components specified in $\mathcal{R}$, the connector constraints the behavior of the overall system, imposing its own behavior. This leads to a level of composition flexibility which is highly desirable in component based systems. Due to the great diversity of communication patterns possible in Reo, these model, in contrast with other models [6] make possible the production of different systems composed out of the same set of components, by the use of different connectors with a well defined semantic.

### *3.2 From Constraint Automata to $\longmapsto_C$ transitions*

Now we present the algorithm to generate the transition rules for the transition relation $\longmapsto_C$, for the connector C from its constraint automata representation.

Let C be a connector defined by the sets $\mathcal{I}$ and $\mathcal{O}$ of input ends and output ends respectively, and its constraint automata $CA_{\mathrm{C}}$ given by:

$$CA_{\mathrm{C}} \equiv (Q_{\mathrm{C}}, \mathcal{N}_{\mathrm{C}}, \rightarrow_{\mathrm{C}}, Qo_{\mathrm{C}})$$

where $\mathcal{N}_{\mathrm{C}} = \mathcal{I} \cup \mathcal{O}$. We associate a name $\mathrm{C}_q$ to every $q \in Q_{\mathrm{C}}$ to indicate the connector C is in a state $q$. As the automata transitions are labelled with the maximum number of nodes over which data can flow simultaneously, we can identify from them the input ends and output ends of the connector over which input or output operations ocurring synchronously produce a state change. The symbol $\models$ represents the satisfaction relation resulting from interpreting data constraints over data assignments.

$$\langle ExR_0, \{wr(a,t), tk(b,t)\}\rangle \overset{\{wr(a,t),tk(b,t)\}}{\longmapsto_{\text{ExR}}} \langle ExR_0, \emptyset\rangle$$
$$\langle ExR_0, \{wr(a,t), tk(c,t)\}\rangle \overset{\{wr(a,t),tk(c,t)\}}{\longmapsto_{\text{ExR}}} \langle ExR_0, \emptyset\rangle$$
$$\langle ExR_0, \{wr(a,t), rd(b,t)\}\rangle \overset{\{tk(b,t)\}}{\longmapsto_{\text{ExR}}} \langle ExR_0, \{wr(a,t)\}\rangle$$
$$\langle ExR_0, \{wr(a,t), rd(c,t)\}\rangle \overset{\{rd(c,t)\}}{\longmapsto_{\text{ExR}}} \langle ExR_0, \{wr(a,t)\}\rangle$$

Table 2
Transition Rules for the Exclusive Router

Considering the issues mentioned we propose the following algorithm:

(i) for each transition $(q \xrightarrow{N,g}_{\text{C}} p) \in \rightarrow_{\text{C}}$, a transition is generated as follows:

$$\langle \text{C}_q, act_\delta\rangle \overset{act_\delta}{\longmapsto_{\text{C}}} \langle \text{C}_p, \emptyset\rangle$$

where $\delta$ is any data assignment function such that $\delta \models g$, y $act_\delta$ is defined as $(act^{wr})_\delta \cup (act^{tk})_\delta$, where:

$$(act^{wr})_\delta = \{wr(I, \delta(I)) : I \in \mathcal{I} \cap N\}$$
$$(act^{tk})_\delta = \{tk(O, \delta(O)) : O \in \mathcal{O} \cap N\}$$

(ii) for each transition rule $\langle \text{C}_q, \overline{act}\rangle \overset{\overline{act}}{\longmapsto_{\text{C}}} \langle \text{C}_p, \emptyset\rangle$ generated in (i), suppose $\overline{act} = \overline{act^{tk}} \dot{\cup} \overline{act^{wr}}$, the disjoint union of the $tk$ actions and the $wr$ actions that can be applied over the connector ends. For each $\overline{act}' \subseteq \overline{act^{tk}}$, we construct $\overline{act^{rd}} = \{rd(O,t) : tk(O,t) \in \overline{act}'\}$ and generate a rule $\langle \text{C}_q, (\overline{act} - \overline{act}') \cup \overline{act^{rd}}\rangle \overset{\overline{act^{rd}}}{\longmapsto_{\text{C}}} \langle \text{C}_q, \overline{act} - \overline{act}'\rangle$

We need to consider the $rd$ operation particularly because of its non destructive condition. Last rule consider the situation in which at least one $rd$ operation is applied synchronously with other communication operations. In this case only $rd$ operations succeed, the other communication actions ($wr$ and $tk$) remain pending over the corresponding ends until the environment provides the necessary conditions for them to proceed, by the application of some other rule.

Consider the constraint automata $CA_{ExR}$ corresponding to the connector exclusive router (Figure 1) given by the tuple:

$$CA_{ExR} = (Q, \{a,b,c\}, \rightarrow_{\text{ExR}}, Q)$$

where $Q = \{0\}$, and $(0 \overset{\{a,b\},d_a=d_b}{\longrightarrow}_{\text{C}} 0)$ and $(0 \overset{\{a,c\},d_a=d_c}{\longrightarrow}_{\text{C}} 0) \in \rightarrow_{\text{C}}$. Applying the algorithm to the constraint automata $CA_{ExR}$ we obtain the transition rules given in table 2. Finally we represent the connector as follows:

$$ExR = \langle \{a\}, \{b,c\}, \{ExR_0\}, \longmapsto_{ExR}\rangle$$

# 4 Specifying components protocols in Reo

As we have already mentioned, intending to solve the interoperability problems at protocol level we propose the use of the model introduced in the previous section,

based on Reo, to enhance components interfaces with a description of an abstract component interaction protocol. We ilustrate our proposal by means of an example. We describe a simplified version of a real patient monitoring system that was first introduced by Papadopoulos and Arbab [13] to show the potential of control driving coordination languages for expressing dynamically reconfigurable software architectures. The basic scenario involves a number of monitors and nurses. There is a monitor, one for each patient, recording readings of the patient's health state in response to a received request. Besides a monitor can also send data in case of exceptional situations. A nurse is responsible for periodically checking the patient's health state by asking the corresponding monitor for readings; further more a nurse should respond to receiving exceptional data readings.

As we can see in the interface below, a monitor offers one method that allows the user to request the periodical readings. The nurse interface defines two methods to be invoked by the environment. Method *normal* implements the main service offered by the process, it receives readings of the patient's health state on the parameter *normalState* and processes them. On the other hand the method *signal* allows the nurse to treat emergency cases, which are captured on the *emergencyState* parameter.

```
interface Monitor {
    void request();
}

interface Nurse {
    void signal ([in]Data emergencyState);
    void normal ([in] Data normalState);
}
```

### *4.1 Interaction protocols*

From the interfaces above is very difficult to discern the way in which a monitor and a nurse will behave if they are integrated in a software application. Nothing is said concerning to their interactions and the rules governing them. In fact, it is not manifested neither the possibility for a monitor to send emergency signals nor that emergency situations have priority for being attended. Now we give the specification for both agents, a monitor and a nurse, oriented to overcome this situation.

A monitor receives a request for its data registers on the patient health state readings. Eventually the monitor may detect abnormal situations and in this case it has to send an emergency state. Emergency situations have priority for being attended. Note that the monitor only needs to receive a piece of data in the connection point *requestIn*, and this action is interpreted as a request for information.

On the other hand a nurse is responsible for checking the patients health state. He or she requests a monitor for its data registers writing a token in the connection point associated to this action. A nurse must also attend the reception of emergency states, which must be attended first. The behavior of both agents is defined bellow

```
MONITOR = tk(requestIn).(MONITOR1
```

```
                        +
                        wr(signalOut,<emergencyState>). MONITOR1
                        )
           +
           wr(signalOut,<emergencyState>). MONITOR

MONITOR1 = wr(normalOut,<normalState>). MONITOR


NURSE = wr(requetOut,token). NURSE1
        +
        tk(signalIn,<emergencyState>). NURSE

NURSE1 = tk(normalIn,<normalState>). NURSE
         +
         tk(signalIn,<emergencyState>).
         tk(normalIn,<normalState>). NURSE
```

Component interaction protocols are specified to describe the behavior of given component interfaces. In general, there are no precise guidelines about what should and should not be included in a protocol specification. It will depend, of course, on the level of abstraction or details required. Because of in Reo communication only is possible by means of input and output operations over connector ends (connection points) we must take them into account when specifying protocols. Thus, we associate an input end with each method representing a service offered by the component, and we considered an output end for each service required by the component. In case the method has no arguments we do not consider any object in the input operation. However, for the output operation a token is needed, just as a signal for the requested service.

### *4.2 Selecting the connector*

At this point we address the selection of an adequate connector for the composition of a monitor and a nurse. Consider the situation in which in the resulting application the monitor must serve first the emergency signals and the nurse has the obligation to firstly deal with emergency situation. In the specification neither the monitor, nor the nurse ensure the priority in attending emergency cases, because of the non deterministic choice among attending the periodical readings and attending the emergency readings. In this scenario it seems clear that the expected behavior of the composition of a nurse and a monitor is achieved only when selecting a connector which enforces the required priorities. With this aim in mind we selected the connector *CMN* shown in Figure 2.

The connector is defined by the tuple

$$\langle \{RIn, SIn, NIn\}, \{ROut, SOut, NOut\}, \Sigma_{CMN}, \longmapsto_{CMN} \rangle$$

In table 3 we give the transitions in the labelled transition relation which defines

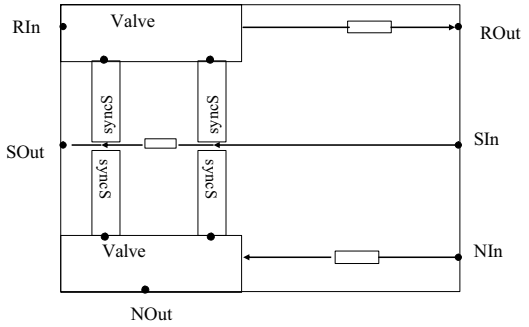


Fig. 2. CMN Connector

| | |
|---|---|
| (1) | $\langle CMN_0, \{wr(RIn,t)\}\rangle \overset{\{wr(RIn,t)\}}{\longmapsto_{\text{CMN}}} \langle CMN_1, \emptyset\rangle$ |
| (2) | $\langle CMN_0, \{wr(SIn,t)\}\rangle \overset{\{wr(SIn,t)\}}{\longmapsto_{\text{CMN}}} \langle CMN_3, \emptyset\rangle$ |
| (3) | $\langle CMN_1, \{wr(SIn,t)\}\rangle \overset{\{wr(SIn,t)\}}{\longmapsto_{\text{CMN}}} \langle CMN_2, \emptyset\rangle$ |
| (4) | $\langle CMN_6, \{wr(SIn,t)\}\rangle \overset{\{wr(SIn,t)\}}{\longmapsto_{\text{CMN}}} \langle CMN_4, \emptyset\rangle$ |
| (5) | $\langle CMN_2, \{wr(NIn,t)\}\rangle \overset{\{wr(NIn,t)\}}{\longmapsto_{\text{CMN}}} \langle CMN_4, \emptyset\rangle$ |
| (6) | $\langle CMN_7, \{wr(SIn,t)\}\rangle \overset{\{wr(SIn,t)\}}{\longmapsto_{\text{CMN}}} \langle CMN_5, \emptyset\rangle$ |
| (7) | $\langle CMN_2, \{tk(SOut,t)\}\rangle \overset{\{tk(SOut,t)\}}{\longmapsto_{\text{CMN}}} \langle CMN_1, \emptyset\rangle$ |
| (8) | $\langle CMN_2, \{tk(ROut,t)\}\rangle \overset{\{tk(ROut,t)\}}{\longmapsto_{\text{CMN}}} \langle CMN_3, \emptyset\rangle$ |
| (9) | $\langle CMN_3, \{tk(SOut,t)\}\rangle \overset{\{tk(SOut,t)\}}{\longmapsto_{\text{CMN}}} \langle CMN_0, \emptyset\rangle$ |
| (10) | $\langle CMN_4, \{tk(ROut,t)\}\rangle \overset{\{tk(ROut,t)\}}{\longmapsto_{\text{CMN}}} \langle CMN_5, \emptyset\rangle$ |
| (11) | $\langle CMN_4, \{tk(SOut,t)\}\rangle \overset{\{tk(SOut,t)\}}{\longmapsto_{\text{CMN}}} \langle CMN_6, \emptyset\rangle$ |
| (12) | $\langle CMN_5, \{tk(SOut,t)\}\rangle \overset{\{tk(SOut,t)\}}{\longmapsto_{\text{CMN}}} \langle CMN_7, \emptyset\rangle$ |

Table 3
Behavioral Transitions for *CMN*

its behavior, which have been generated following the algorithm previously introduced. Since this connector presents many possible states and so many transitions, we give only those related to the blocking situations.

This connector imposes certain restrictions over its connection points which

seems appropriate for solving our priority problems. The expected behavior is imposed by the effect of the two *valve* connectors (see [1]), and the four *syncSignal* connectors (which are connected to the valve control connection points), present in the configuration of the connector. The *syncSignal* connector is the result of the composition of a syncDrain channel with a syncSpout channel, which has almost the same behavior of a sync channel, except that it doesn't matter which is the item written over its input end. From the analysis of its transition relation, we conclude that it presents the needed behavior. In fact, it shows an asynchronous behavior, which in some cases is disable by means of an input operation over the input end *SIn* —rules (3), (5) y (9)—, leaving the connector in a state in which it remains blocked until an output operation is applied over the output end *SOut* -rules (11), (13), (15) and (16)-. The blocking effect is the result of the propagation of the input operation over *SIn* foward to the open input end of both valves connectors. Indeed, even when the connector is blocked it is possible to apply input and output operations when it is for example in state $CMN_2$ (blocked but with a data item present in the buffer associated with ROut), or in state $CMN_4$ (blocked but with a data item present in the buffer associated with ROut, and a data item present in the buffer associated with NIn). When composing a nurse and a monitor via the connector CMN, the expected effect is achieved regarding the input and output ports for emergency signals are connected to the connection points *Sin* and *Sout* of CMN respectively.

## 5 Conclusions

Although Reo was defined with a different purpose (i.e. coordination), by applying the previously explained approach it can also be used to specify the interaction behavior of software components. The information provided by this kind of protocols may be useful for analyzing a number of properties like compatibility [6](when two components can interact without deadlocking) or substitutability (when a component can be substituted by another one, preserving its "safe" behavior in the system).

Reo's capability in expressing component's protocols was manifested by the example. If we analyze the specifications we can observe that without increasing the complexity embedded in the interaction protocol is possible to ensure the priority on the treatment of emergency signals, selecting an adequate connector. Moreover, the composition of a *Monitor* and a *Nurse* in presence of other connector (for example one without blocking behavior) results in an application completely different in that the priority in attending emergency signals is not ensure. Because of the possibility of merging synchronous and asynchronous behaviors in connectors and the special semantics of some channels (for example Drain and Spout types), connectors may offered many different communication patterns, and may respond to any constraint needed by an application. In our example the complexity added by the treatment of the emergency signal is transferred to the connector, maintaining a simple and elegant specification.

The main objective of this paper was to define a framework for describing the behavior of components in terms of coordination models. In this sense, the basic idea is based on extending interface description languages with an explicit description of the interactive behavior of a component in a similar way as behavioral types [12] or role-based representations [7,8]. To do this, we consider the coordination model Reo which by means of its channel composition mechanism and the great diversity of channel types (with a well defined behavior) allows the construction of many different connectors, imposing specific coordination patterns. In contrast with other models [6], the model based on Reo make possible the production of different systems composed out of the same set of components, by the use of different connectors. We argue that this model of coordination, Reo, is mature enough to be used in the design and validation of components of large distributed systems, and the use of such methods will lead to the better design of components and component-based applications in open systems.

Our future work will be devoted to formally define compatibility and substitutability relations for the model presented in this work, oriented to their semi-automated evaluation. We are also interested in the semiautomated selection of connectors for the coordination.

## References

[1] F. Arbab. *A Channel-based Coordination Model for Component Composition.* Electronic Notes in Theoretical Computer Science, 68(3), 2003.

[2] F. Arbab, J. Rutten "A coinductive calculus of component connectors". Technical CWIReport SEN-R0216 ISSN 1386-369X, 2002.

[3] F. Arbab, C. Baier, J. Rutten, M. Sirjani "Modeling Component Connectors in Reo by Constraint Automata". CWI Technical Report SEN-R0304, ISSN 1386-371l, 2003.

[4] R. Bastide, O. Sy, P. Palanque. *Formal specification and prototyping of CORBA systems.* Lecture Notes in Computer Science, 1628, 474-494, 1999.

[5] A. Braccali, A. Brogi, F. Turini. *Coordinating interaction patterns.* Proc. $16^{th}$ ACM Sym. 2001.

[6] A. Brogi, E. Pimentel, and A. Roldán. *Compatibility of Linda-based Component Interfaces.* Electronic Notes in Theoretical Computer Science, 66(4), 2002.

[7] C. Canal "Un Lenguaje para la Especificacin y Validacin de Arquitecturas de Software". PhD thesis, Department of Languages and Computer Science, University of Málaga, 2001.

[8] C. Canal, L. Fuentes, E. Pimentel, J.M. Troya, A. Vallecillo. *Extending Corba Interfaces with Protocols.* The Computer Journal, 44(5):448–462, 2001.

[9] H. Han. *Semantic and usage packing for software components.* Proceedings of WOI'99, 25-34, 1999.

[10] D. Lea. *Interface-based protocol specification of open systems using PSL* (ECOOP'95) Lecture Notes in Computer Science, 25-34, Springer, 1995.

[11] G. Leavens, Staraman. "Foundations of Component-Based Systems". Cambridge University Press, 2000.

[12] J. Magee, J. Kramer, D. Giannakopoulou. "Behaviour analysis of software architectures" Kluwer Acad. Publ. 1999.

[13] G. Papadopoulos, F. Arbab. *Dynacmic Reconfiguration in Coordination Languages.* Advances in Computer 46, Acad. Press, 329-400,1998.

[14] A. Vallecillo, J. Hernndez, J.M. Troya. "Component Interoperability" Technical Report ITI-2000-37, Department of Languages and Computer Science, Univ. of de Mlaga, July 2000.

[15] D.M. Yellin, R.E. Strom. *Protocol Specifications and Components Adaptors.* ACM Transactions on Programming Languages and Systems, 19:1, 292-333, 1997.