Towards a Coordination Model for Interactive Systems

Marco Antonio Barbosa [1,4] Luís Soares Barbosa [2,4]
José Creissac Campos [3,5]

DI-CCTC – Universidade do Minho
Braga, Portugal

Abstract

When modelling complex interactive systems, traditional interactor-based approaches suffer from lack of expressiveness regarding the composition of the different interactors present in the user interface model into a coherent system. In this paper we investigate an alternative approach to the composition of interactors for the specification of complex interactive systems which is based on the coordination paradigm. We layout the fundations for the work and present an illustrative example. Lines for future work are identified.

Key words: Interactors, Coordination models, Configuration.

1 Introduction

Interactive systems can be seen as a special case of the more general class of reactive systems. However, interactive systems have specificities that present new challenges when considering modelling and reasoning about them. One major aspect is the need to consider interaction with the user, and not only between components of the user interface.

The notion of interactor has long been proposed as an approach to structuring and organizing models of interactive systems. Different authors use different flavours of interactors. A common trait being the view of interactors

[1] Email: marco.antonio@di.uminho.pt
[2] Email: lsb@di.uminho.pt
[3] Email: jose.campos@di.uminho.pt
[4] Research carried out in the context of the PURe Project supported by Fct, the Portuguese Foundation for Science and Technology, under contract POSI/ICHS/44304/2002.
[5] Research carried out in the context of the IVY Project supported by Fct, the Portuguese Foundation for Science and Technology, and FEDER, the European regional development fund, under contract POSC/EIA/56646/2004.

This paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

as components capable, not only of communicating between themselves, but also of conveying information to the user(s).

Two main flavours of interactors are York [11] and CNUCE [19] interactors. York interactors are basically objects equipped with a rendering relation that maps their internal state into some presentation medium. More than a concrete specification formalism, they offer a framework for structuring the user interface specifications, whatever formalism is being used. CNUCE interactors (see Fig. 1) can be seen as blackbox components that communicate, with each other, and with the user(s), through input and output ports (for more on CNUCE interactors see section 3).

One main distinction between the two approaches is that with York interactors state can be specified explicitly (cf. MAL interactors [9]), while with CNUCE interactors state is only referred two indirectly through the interactor's ports. Whatever the approach, modelling complex interactive systems entails creating architectures of interconnected interactors. In the case of York interactors, there is no prescription about how that should be accomplished (it will depend on the particular specification approach being used). In the case of CNUCE interactors, specifications are built by connecting the different ports into an adequate architecture by means of synchronous channels. Interactor behaviour is modelled in LOTOS by expressing the relation between input and output ports.

Managing the coordination between the different interactors is typically achieved by the introduction of additional interactors to express the control logic for their communication. This, in turn, adds to the complexity of the models. Ideally we should be able to express the logic of the coordination between the different interactors in an as natural and simple way as possible. In this paper we explore the application of the coordination paradigm to model architectures of interactors. The approach is based on previous work of some of the authors (see, [5,6]).

2 Coordination

The *coordination paradigm* [13,18] offers a promising way to address issues related to the development of complex systems. Since the coordination component is separate from the computational one, the former views the processes comprising the latter as black boxes, whose internal implementation is hidden from the outside world. Instead, the composition of components is defined in terms of their (logical) interfaces which describe their externally observable behavior. By hiding all system computation in the components, a system can be described in terms of the observable behavior of its components and their interactions. As such, component-based software modelling provides a high-level abstract description of a system that allows for a clear separation of concerns between the coordination and the computational aspects.

Closely related to the concept of coordination is that of configuration and

architectural description. They view a system as comprising components and interconnections, and aim at separating structural description of components from component behaviour. Furthermore, they support the formation of complex components as compositions of more elementary components. Finally, they understand changing the state of some system as an activity performed at the level of interconnecting components rather than within the internal purely computational functionality of some component.

Our approach is based on the coordination model Reo [2], a subset of Reo, to be more exact. A more formal treatment of the semantics of our approach is shown in [5,6] where we describe an exogenous coordination model wherein complex coordinators, called "connectors" are compositionally built out of simpler ones. This implies that not only should it be generally possible to produce different systems by composing the same set of components in different ways (creating different configurations), but also that the difference between two systems composed out of the same set of components must arise out of the actual rules that comprise their two different compositions, *i.e.*, their glue code. In such a context we may specify different configurations for a given scenario only by constructing different connectors and patterns of interactions.

Another feature of this work is that our approach takes advantage of the authors' previous work on named *generic process algebra* [3,21]. Such work provides a more general and adaptable approach to the design of complex systems using process algebras. For example, some applications may require similar constructs coexisting with different interaction disciplines (see section 4.2).

Using process algebra to model interactors is not new, and we may refer to the usage of LOTOS in [19] and CSP in [10] (just to name a few). However, our generic approach provides a more flexible way to represent interactors by proposing a clear separation between structural aspects and interaction disciplines.

3 CNUCE Interactors

Paternò views interactors (CNUCE interactors — see Fig. 1) as blackbox entities which communicate through a public interface identified by ports with opposite polarity (*i.e.*, either input or output). Ports are divided into different categories. There are ports to communicate with the users (somewhat equivalent to the rendering relation in York interactors) and ports to communicate with the underlying application functional core. There are also triggers, needed to synchronize the flow of information from input to output ports.

Specifications are built by connecting the ports of different interactors into an adequate architecture by means of synchronous channels. Interactor behaviour is modelled in LOTOS by expressing the relation between input and output ports.

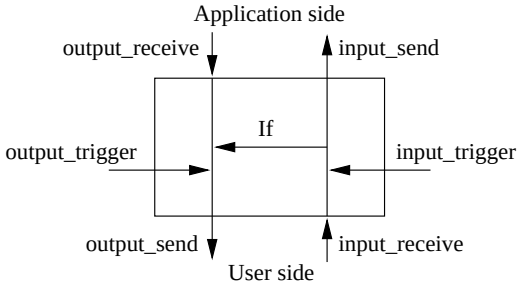

Fig. 1. CNUCE Interactors

An interactor can generate data in two directions: towards the user, and towards the application. This means that interactor behaviour is divided into two distinct parts: the *external* one, which contributes to the definition of the appearance, and the *internal* one, which consists of sending data to other interactors or application processes. Hence, an interactor is defined by a couple of functions: FI is associated with the internal behaviour (the information flow from the user towards the application side); FO is associated with the external behaviour (the information flow from the application towards the user side).

An interactor I, with input_receive ports I_{m_1} to I_{m_n}, input_send ports Inp_1 to Inp_t, output_receive ports I_{c_1} to I_{c_s} and output_send ports Out_1 to Out_z, is defined as (with Φ representing the absence of information)

$I = (FI, FO)$, where:
$FI : (I_m \times Bool \times T) \rightarrow (Inp \cup \Phi) \times If$
with $I_m = I_{m_1} \times \ldots \times I_{m_n}, If = If_1 \times \ldots \times If_k, Inp = Inp_1 \times \ldots \times Inp_t$
$FO : (I_c \times If \times Bool \times T) \rightarrow (Out \cup \Phi)$
with $I_c = I_{c_1} \times \ldots \times I_{c_s}$, and $Out = Out_1 \times \ldots \times Out_z$

As can be seen I_m, If, Inp, I_c, and Out are domains defined by Cartesian products of subdomains. This is mainly because an interactor can receive (and generate) different data types from (to) collections of channels.

In the definition of FO, I_c represents the domain describing the output entities which it receives from the outside. Out is the type which describes the external appearance which can be generated, and If is the data type which the input part of the interactor passes to the output part for echoing. T, in both the FI and FO definitions, is the time that can be considered as a one-dimensional quantity, made up of points, where each point is associated with a value. At moment t, FO is applied to data from I_c and to an element in domain If produced by FI at moment $t-1$.

For interactors without explicit triggers (interactors that generate meaningful results whenever they receive any input), the Boolean in the above can be ignored.

Generally speaking the main difference between the two functions describing one interactor is that the external function receives input data from the

input part of the interactor (in order to echo the current measure value) as well as from the outside. This indicates that the presentation of an interactor is defined by the information it receives from higher levels (levels closer to the application) and the feedback information generated by the users' input.

4 Interactors and coordination

The main aspects of the CNUCE model of interactors can be summarized as follows: interactors are seen as black-box entities communicating through identified ports (input/output), a notion of discrete time and synchronization constraints (involving a notion of trigger) are included in the model, and composition is used in order to construct complex interactive systems from simple components. Such features resemble previous work on coordination models by some of the authors [6,5].

The goal of this paper is to provide an alternative model for expressing the composition of interactors. Central to our approach to the rigorous representation of *interactors* is the notion of *configuration*. This captures the intuition that interactors may be seen as components which cooperate through their specific *interfaces* using *connectors* as the unique communication mechanism, *i.e.*, interactors do not directly interact among themselves. Such idea of connector abstracts the idea of an intermediate *glue code* to handle interaction.

In order to represent a configuration we need a notion of *a)* interactor's *interface*, *b)* what *connectors* are and how they compose, and *c)* how interactors' *interfaces* and *connectors* interact in a *configuration*. These points are tackled in the following sub-sections.

4.1 Interfaces

In exogenous coordination models, like [2] or [5], components are black box entities accessed by purely syntactic interfaces. The role of an interface is restricted to keeping track of port names and, possibly, of admissible types for data items flowing through them [6]. Such a notion of components interface is perfectly extensible with the notion of CNUCE interactors. So, let us define an interface as

Definition 4.1 An interface for a component C is specified by a *port signature*, $sig(C)$ over $\mathbb{D}$, given by a port name and a polarity annotation (either in(put) or out(put)), and a *use pattern*, $use(C)$, given by a process term over port names.

Typically the behaviour of a component's interface can be expressed using transition systems [16], regular-expressions [20] or process algebras [1]. Process algebra, in particular, provides an expressive setting for representing

[6] In the sequel, however, we assume a unique, general data domain, denoted by $\mathbb{D}$, as the type of all data values flowing in an application.

behavioural patterns and establish/verify their properties in a compositional way. Some flexibility, however, is required with respect to the underlying interaction discipline (captured in this work by θ). Actually, different such disciplines have to be used, at the same time, to capture different aspects of component coordination. For example the discipline governing the composition of *software connectors* (to build the overall *glue code*) differs from the one used to capture the interaction between the connectors and the relevant components' interfaces. Meeting this goal entails the need for a *generic* way to *design* process algebras.

The model proposed in this work resorts to the rigorous discipline of process calculi, namely the calculational style presented in [3] to express both component and connectors behaviour.

4.2 Generic Process Algebra

References [3,4] introduced a denotational approach to the *design* of process algebras in which processes are identified with inhabitants of a final coalgebra [15] and their combinators defined by coinductive extension (of 'one-step' behaviour generator functions). The *universality* of such constructions entails both definitional and proof principles on top of which the development of the whole calculus is based.

As we shall see in the following our generic approach to process algebras maintains the basic combinators present in classical processes algebras as CCS, CSP or LOTOS. The fundamental point to be noted is the presence of a more flexible way to represent an *interaction discipline* which parametric on θ. Technically, an *interaction discipline* is modeled as an Abelian positive monoid $\langle Act; \theta, 1\rangle$ with a zero element 0. The intuition is that θ determines the interaction discipline whereas 0 represents the absence of interaction: for all $a \in Act$, $a\theta 0 = 0$. On the other hand, being a positive monoid entails $a\theta a' = 1$ iff $a = a' = 1$. A typical example of an interaction structure captures action co-occurrence as in CSP, in which case θ is defined as $a\theta b = \langle a, b\rangle$, for all $a, b \in Act$. Another example is provided by the action complement match used in CCS [17], *i.e.*, $a\theta\bar{a} = \tau$.

Definition 4.2 Let P be the set of port identifiers and S stand for (the specification of) a component. Its use pattern, denoted by $use(S)$ is given by a process expression over $Act \triangleq \mathcal{P}P$, given by the following grammar:

$$\begin{aligned} P \ ::= \ & \mathbf{0} \mid \alpha.P \mid P + P \mid P \otimes P \mid P \,|||\, P \mid P; P \mid P \mid P \mid \\ & \sigma\, P \mid \mathsf{fix}\ (x \ = \ P) \end{aligned}$$

where α is an element of Act (*i.e.*, a set of port identifiers) and σ is a substitution.

Notice that choosing Act as a *set* of port identifiers allows for the synchronous activation of several ports in a single computational step.

Combinators $\mathbf{0}$, ., $+$, $\otimes$, $|||$, and $|$, represent inactive process, prefix, nondeterministic choice, synchronous product, interleaving, and parallel composition, respectively. *Renaming* is given by term substitution. The $\mathsf{fix}\ (X = P)$ is a fixed point construction, which, as usual, can be abbreviated in an explicit recursive definition. Sequential composition, as in CSP [14], is given by ';' and requires its first argument to be a terminating process.

The semantics of such expressions is fairly standard, but for the parametrization of all forms of parallel composition (*i.e.*, $\otimes$ and $|$) by an interaction discipline as discussed above. The reader is referred to [21] for the full details.

Definition 4.3 The joint behaviour of a collection $\{S_i |\ i \in n\}$ of components is given by

$$use(S_1)\ |\ \ldots\ |\ use(S_n)$$

where the interaction discipline is fixed by $\theta = \cup$, *i.e.*, the synchronisation of actions in α and β corresponds to the simultaneous realization of all of them.

This joint behaviour is computed by the application of Milner's *expansion law* [7] , while obeying the interaction discipline given by θ. The following example illustrates this construction.

Example 4.4 Consider a component C_1 with two ports a and b whose use pattern is restricted to the activation of either a or b, forbidding their simultaneous occurrence. The expected behaviour is captured by

$$use(C_1)\ =\ \mathsf{fix}\ (x\ =\ a.x\ +\ b.x)$$

Now consider another component, C_2, with ports c and d whose behaviour is given by the co-occurrence of actions in both ports. Therefore,

$$use(C_2)\ =\ \mathsf{fix}\ (x' = cd.x'), \quad \text{where, } cd \stackrel{\text{abv}}{=} \{c, d\}$$

According to definition 4.3, the joint behaviour of C_1 and C_2 is

$$use(C_1)\ |\ use(C_2)\ =\ \mathsf{fix}\ (x\ =\ acd.x\ +\ bcd.x\ +\ a.x\ +\ b.x\ +\ cd.x)$$

As a final example, consider still another component C_3, with ports e and f activated in strict order, *e.g.*, first input e and then output f

$$use(C_3)\ =\ \mathsf{fix}\ (y\ =\ e.f.y)$$

Clearly, expansion leads to

$$use(S_2)\ |\ use(S_3)$$
$$=\ \mathsf{fix}\ (x\ =\ cd.x\ +\ e.f.x\ +\ cde.f.x\ +\ cde.cdf.x\ +\ e.cdf + \cdots + cdf.x)$$

[7] This law, which states that a process is always equivalent to the non deterministic choice of its derivatives, is a fundamental result in interleaving models for concurrency.

4.3 Connectors

Our approach resorts to connectors as the only inter-component communication mechanism. This allows a clean, flexible, and expressive model for construction of the glue code for component composition which also supports exogenous coordination.

Connectors are *glueing devices* between services which ensure the flow of data and the meet of synchronization constraints. Their specification builds on top of our previous work on component interconnection [5], extended with an explicit annotation of activation, or *use*, patterns for their *ports*.

Ports are *interface points* through which messages flow. Each port has an *interaction polarity* (either *input* or *output*). Another particular characteristic is the capability to construct complex connectors out of simpler ones using a set of *combinators*.

Let $\mathbb{C}$ be a connector with m input and n output ports. Assume, again, $\mathbb{D}$ as a generic type of data values and $\mathbb{P}$ as a set of (unique) *port identifiers* Formally, the behaviour of a connector may be given by

Definition 4.5 The specification of a connector $\mathbb{C}$ is given by a relation $\mathsf{data}.[\![\mathbb{C}]\!] : \mathbb{D}^m \longrightarrow\!\!\!\!\!\rightarrow \mathbb{D}^n$ which records the flow of data, and a process expression $\mathsf{port}.[\![\mathbb{C}]\!]$ which gives the pattern of port activation.

The model provides a set of basic connectors and combinators which allow us to construct more elaborated connectors and define more complex patterns of coordination and interaction. In the following let us consider some of these basic connectors. For more connectors and a more formal treatment of them we refer to [5,6].

4.3.1 Synchronous channel.

The *synchronous channel* has two ports of opposite polarity. This connector forces input and output to become mutually blocking, in the sense that any of them must wait for the other to be completed.

$$\mathsf{data}.[\![\, \bullet \longmapsto\!\!\!\!\!\rightarrow \bullet \,]\!] \;=\; \mathsf{Id}_{\mathbb{D}} \quad \text{and} \quad \mathsf{port}.[\![\, \bullet \longmapsto\!\!\!\!\!\rightarrow \bullet \,]\!] \;=\; \mathsf{fix}\,(x \;=\; ab.x)$$

Its semantics is simply the identity relation on data domain $\mathbb{D}$ and its behaviour is captured by the simultaneous activation of its two ports.

4.3.2 Drain.

A drain has two input, but no output, ports. Therefore, it loses any data item crossing its boundaries. A drain is *synchronous* if both write operations are requested to succeed at the same time (which implies that each write attempt remains pending until another write occurs in the other end-point). It is *asynchronous* if, on the other hand, write operations in the two ports do

not coincide. The formal definitions are, respectively,

$$\mathsf{data}.[\![\bullet \vdash\!\!\blacktriangledown\!\!\dashv \bullet]\!] = \mathbb{D} \times \mathbb{D} \quad \text{and} \quad \mathsf{port}.[\![\bullet \vdash\!\!\blacktriangledown\!\!\dashv \bullet]\!] = \mathsf{fix}\ (x = ab.x)$$

and,

$$\mathsf{data}.[\![\bullet \vdash\!\!\triangledown\!\!\dashv \bullet]\!] = \mathbb{D} \times \mathbb{D} \quad \text{and} \quad \mathsf{port}.[\![\bullet \vdash\!\!\triangledown\!\!\dashv \bullet]\!] = \mathsf{fix}\ (x = a.x + b.x)$$

4.3.3 $Fifo_1$.
This is a channel with a buffer of a single position.

$$\mathsf{data}.[\![\bullet \longmapsto \bullet]\!] = \mathsf{Id}_{\mathbb{D}} \quad \text{and} \quad \mathsf{port}.[\![\bullet \longmapsto \bullet]\!] = \mathsf{fix}\ (x = a.b.x)$$

4.4 Combinators

Connectors can be combined to build more complex *glueing code*. The following are the required combinators.

4.4.1 Aggregation.
This combinator places its arguments side-by-side, with no direct interaction between them.

$$\mathsf{port}.[\![\mathbb{C}_1 \boxtimes \mathbb{C}_2]\!] = \mathsf{port}.[\![\mathbb{C}_1]\!] \mid \mathsf{port}.[\![\mathbb{C}_2]\!], \quad \text{with } \theta = \cup \tag{1}$$

4.4.2 Hook.
This combinator encodes a *feedback* mechanism, drawing a direct connection between an output and an input port. Formally, $\mathsf{port}.[\![\mathbb{C} \curvearrowleft_i^j]\!]$ is obtained from $\mathsf{port}.[\![\mathbb{C}]\!]$, by deleting references to ports i and j. To be well-formed it is required that i and j appear in different factors of some form of parallel composition ($|||$, $\otimes$, or $|$).

4.4.3 Join.
Its effect is to plug ports with same polarity. The aggregation of *output ports* is done by a *right join* ($\mathbb{C} \overset{i}{j} > z$), where $\mathbb{C}$ is a connector, and i and j are ports and z is a fresh name used to identify the new port. Port z receives asynchronously messages sent by either i or j. When messages are sent at the same time the combinator chooses one of them in a nondeterministic way. On the other hand, aggregation of *input ports* resorts to a *left join* ($z <_j^i \mathbb{C}$). This behaves like a *broadcaster* sending synchronously messages from z to both i and j. Formally, at a behavioural level, both operators effect is that of a renaming operation

$$\mathsf{port}.[\![(\mathbb{C} \overset{i}{j} > n)]\!] = \mathsf{port}.[\![(n <_j^i \mathbb{C})]\!] = \{n \leftarrow i, n \leftarrow j\}\mathsf{port}.[\![\mathbb{C}]\!]$$

4.5 Configurations

Finally, let us complete the whole picture providing a notion of configuration. A *configuration* is simply a collection of *components*, characterized by their *interfaces*, interconnected through a *connector* network built from elementary connectors using the combinators mentioned above. Formally,

Definition 4.6 A configuration involving a collection $C = \{C_i |\ i \in n\}$ of components is a tuple

$$\langle U, \mathbb{C}, \sigma \rangle \tag{2}$$

where $U = use(C_1) \mid use(C_2) \mid \cdots \mid use(C_n)$ is the (joint) use pattern for C, $\mathbb{C}$ is a connector and σ a mapping of ports in C to ports in $\mathbb{C}$.

The relevant point concerning configurations is the semantics of the interaction between the *connector's port behaviour* and the *joint use patterns* of the involved components. This is captured by a synchronous product $\otimes$ for a quite peculiar θ, which is expected to capture the following requirements:

- Interaction is achieved by the simultaneous activation of identically named ports.
- There is no interaction if the connector intends to activate ports which are not linked to the ones offered by the interactors' side. For example if a port a of an interactor S is connected to the input end of a synchronous channel whose output end is disconnected, no information can flow and port a will never be activated.
- The dual situation is allowed, *i.e.*, if the interactors' side offers activation of all ports plugged to the ones offered by the connectors' side, their intersection is the resulting interaction.
- Moreover, and finally, activation of unplugged interactors' ports is always possible.

Formally, this is captured in the following definition.

Definition 4.7 The behaviour $bh(\Gamma)$ of a configuration $\Gamma = \langle U, \mathbb{C}, \sigma \rangle$ is given by

$$bh(\Gamma) \ = \ \sigma\, U \ \otimes \ \mathsf{port}.[\![\mathbb{C}]\!] \tag{3}$$

where θ underlying the $\otimes$ connective is given by

$$c\ \theta\ c' \ = \ \begin{cases} c \cap (c' \cup \mathsf{free}) \Leftarrow \ c' \subseteq c \\ \emptyset \qquad\qquad\qquad\quad \Leftarrow \ otherwise \end{cases} \tag{4}$$

and free denotes the set of unplugged ports in U, *i.e.*, not in the domain of mapping σ.

5 An Example

As an example let us consider a variation of an air traffic control system presented in [12]. Our example (see Fig. 2) is centred in a scenario where aircrafts A_2 and A_3 are on their final approach to the runway, aircraft A_1 is

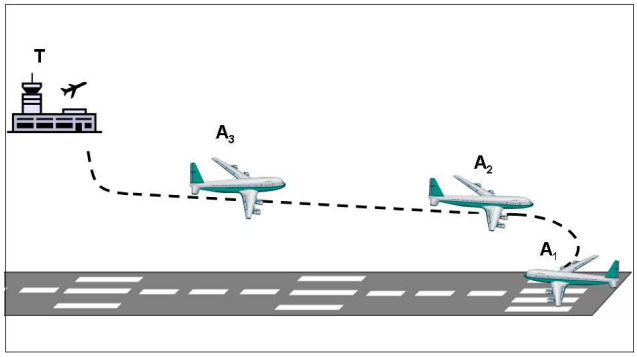

Fig. 2. Air Traffic Control Configuration

on the runway waiting the response for its 'accepted' to take off requirement, and the tower T is responsible for air traffic control. Aircraft A_2 and A_3 are on their "downwind leg" and are to be turned onto a heading towards the runway. Before A_2 can be turned it must reduce speed. This means that A_3 must reduce speed also to avoid loss of separation with A_2. Of course, A_2 will be allowed to land just after A_1 has taken off.

At this stage we are mainly interested in investigating how to combine interactors in different ways for different scenarios. Investigating the appropriateness of each configuration would be the next step in the design process.

First we express the expected behaviour of the interactors involved in this configuration.

interactor: A_i
ports: $\mathsf{slow}'_{\mathsf{i}}, \mathsf{turn}'_{\mathsf{i}}, \mathsf{accept}'_{\mathsf{i}}$
external behaviour:
$use(A_i) \;=\; \mathsf{fix}\,(x \;=\; \mathsf{slow}'_{\mathsf{i}}.x \;+\; \mathsf{turn}'_{\mathsf{i}}.x \;+\; \mathsf{accept}'_{\mathsf{i}}.x)$, where $0 < i \leq 3$.

Such a specification represents the three aircrafts involved in the scenario. Each aircraft has three input ports (distinguished by the symbol: $'$) available for communication in a non-deterministic manner. The tower is represented by interactor T.

interactor: T
ports: $\mathsf{slow}_{\mathsf{i}}, \mathsf{turn}_{\mathsf{i}}, \mathsf{accept}_{\mathsf{i}}$
external behaviour:
$use(T) \;=\; \mathsf{fix}\,(x \;=\; \mathsf{slow}_{\mathsf{i}}.x \;+\; \mathsf{turn}_{\mathsf{i}}.x \;+\; \mathsf{accept}_{\mathsf{i}}.x)$, where $0 < i \leq 3$.

Once the interactors defined, the following step is to define how they will

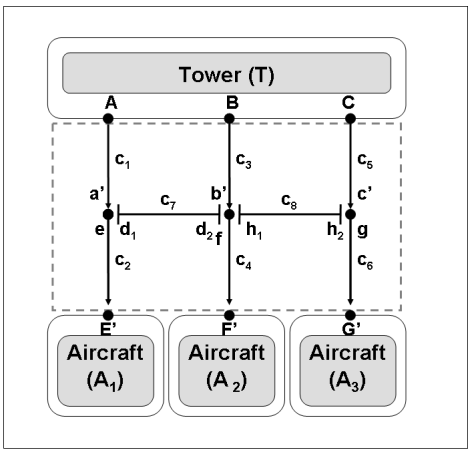

Fig. 3. Air Traffic Control Configuration

cooperate, *i.e.*, we need to represent how the whole system will behave. Such is done by creating an architecture of interactors and connectors.

The scenario captured by Fig. 2 represents a critical situation where the aircrafts must respond to actions appropriately or the safety will be dangerously compromised. So, let us consider a situation where T sends a message $\mathsf{accepted}_1$ to A_1, in order for A_1 to take off, the message slow_2 to A_2, in for A_2 to slow before turning to the runway, and the message slow_3 to A_3 in order for A_3 decrease speed maintaining a safety distance to A_2. In order to ensure that the response to these actions will happens synchronously we may consider a special connector, called *synchronization barrier* SB which enforces that all messages are delivered to their destinations in a synchronous way.

Such a connector (see Fig. 3) is an aggregation among six synchronous channels ($c_1, \ldots, c_6$) and two synchronous drains (c_7 and c_8) which are composed using hook and join combinators. This connector is computed starting from the behaviours of the elementary connectors, *e.g.*, $\mathsf{port}.[\![c_1]\!] = \mathsf{fix}\ (x = aa'.x)$, till the behaviour of the whole connector is calculated: $\mathsf{port}.[\![SB]\!] = \mathsf{fix}\ (x = abce'f'g'.x)$

The resulting behaviour of this connector means that the six ports must be activated synchronously. It should be noted that, since we are not considering timing issues at this stage, this synchronicity does not meant that the ports are activated concurrently. In the current context, what we are stating is that if one port is activated, then all the other must be activated, before the connector can engage in a new interaction.

The configuration of such a scenario is given by

$$\begin{aligned} C_{f_1} &= \langle USC, \mathbb{C}, \sigma_{SC} \rangle, \text{ where} \\ USC &= use(T) \mid use(A_1) \mid use(A_2) \mid use(A_2) \\ \mathbb{C} &= SB \\ \sigma_{cf_1} &= \{a \leftarrow A, b \leftarrow B, c \leftarrow C, e' \leftarrow E', f' \leftarrow F', g' \leftarrow G'\} \end{aligned}$$

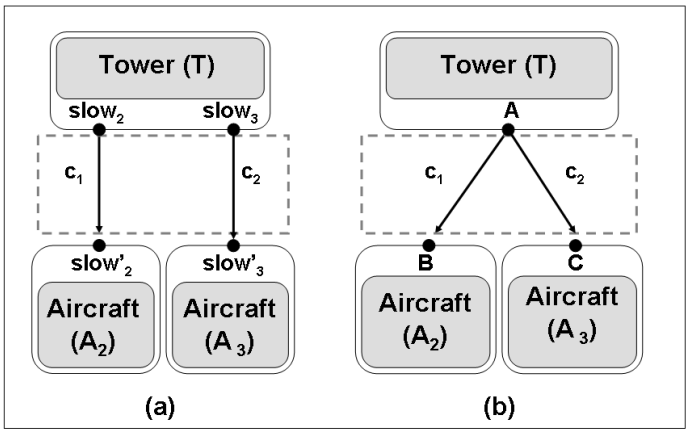

Fig. 4. Parallel and broadcaster connectors

For a cleaner notion let us consider $A = \mathsf{accept}_1$, $B = \mathsf{slow}_2$, $C = \mathsf{slow}_3$, $E' = \mathsf{accept}'_1$, $F' = \mathsf{slow}'_2$, and $G' = \mathsf{slow}'_3$.

The result of the $\otimes$ composition of USC and SB is the behaviour of configuration C_{f_1}. There is no need, however, to compute the complete expansion of the parallel composition in USC expression, which is

$$\begin{aligned}\mathsf{fix}\ (x\ =\ & a.x + \cdots + e'.x + f'.x + g'.x + \\ & ae'.x + \cdots + be'.x + \cdots + ce'.x + \cdots + abce'.x + \cdots + \\ & ae'f'.x + \cdots + be'f'.x + \cdots + ce'f'.x + \cdots + abce'f'.x + \cdots + \\ & ae'f'g'.x + \cdots + be'f'g'.x + \cdots + ce'f'g'.x + \cdots + \underline{abce'f'g'}.x + \cdots + \\ & e'f'.x + e'g'.x + f'g'.x + e'f'g'.x)\end{aligned}$$

because, according to interaction discipline (4), the only successful case of composition with $\mathsf{port}.[\![SB]\!]$ corresponds to the underlined alternative in the expression above. Clearly, the θ-composition of $abce'f'g'$ with $abce'f'g'$ (from the connector side) is $abce'f'g'$, while for all other cases it results in the empty set $\emptyset$. Therefore, and finally,

$$bh(C_{f_1}) \ =\ \mathsf{fix}\ (x\ =\ abce'f'g'.x) \tag{5}$$

Consider now the configuration in Fig. 4 (a) where T sends messages to A_2 and A_3 synchronously. We may specify a situation where T can only send a message for A_2 to slow down if it also sends a slow down message to A_3. This is captured by

interactor: T
ports: $\mathsf{slow}_2, \mathsf{slow}_3$
external behaviour: $use(T)\ =\ \mathsf{fix}\ (x\ =\ \mathsf{slow}_3.x\ +(\ \mathsf{slow}_2.\mathsf{slow}_3.x))$

Consider now a situation where T needs to send synchronously a message to both A_2 and A_3. A solution for this situation is pictured in Fig. 4 (b).

As a final remark is important to note that this work reports on the main ideas of this approach only. The full specification of the calculi involved in the development of the examples was not demonstrated in this paper. We refer to [7] for a complete view of this approach applied to another kind of application.

6 Conclusions and Future Work

When modelling complex interactive systems, traditional interactor-based approaches suffer from lack of expressiveness regarding the composition of the different interactors present in the user interface model into a coherent system. In this paper we have started exploring the application of a coordination based approach to express the *interconnection glue* between interactors.

By using the notion of a black-box component, defined only by its interface to the outside, this approach can be closely related to that of CNUCE interactors. The definition of an interactor in CNUCE as $I = (FI, FO)$, is a relation among input port to output ports, *i.e.*, $I : FI \rightarrow FO$ quite similar to our approach where an interactor is given by a relation $\mathsf{data}.[\![\mathbb{C}]\!] : \mathbb{D}^m \rightarrow \mathbb{D}^n$. Although the definition does not clearly separate ports according to categories (as is done in CNUCE interactors), this can be easily accomplished by using syntactic annotation, as hinted at in the example. Nevertheless, the rendering of information to users is one characteristic of interactors that has not been fully explored in this paper. With the formal underpinning now in place, we intend to explore this as the next step in this work.

Our approach promotes a clear separation of concerns between the specification of the individual components of the model (the interactors), and the specification of how they are organized into an architecture, and how they interact with each other. This separation of concerns was not as clear neither in CNUCE interactors, or in the York based MAL interactors [9], but it is fundamental to enable the modelling of complex systems in a more clear and concise manner.

In the CNUCE model we have an explicit representation of time and a trigger to regulate the synchronisation constraints. Although this aspect has not been addressed here, in [6] a preliminary version of our approach was presented where time was also explicitly defined by a time stamp $\mathbb{T}$ representing, in fact, not real time but a way to represent an order of data occurrence. In the current model the notion of 'time' is implicitly represented by the sequence in which the ports are activated, *i.e.*, the sequence in which the data flows thought the ports. For instance, if we have a synchronous channel, both ports are activated 'at same time' *i.e.*, ports are activated in an atomic way without being interleaved by another operation while both operations have not been well succeed. If we model an asynchronous channel between the activations of both ports, then other port activations might succeed in between the two. Another point to note is that with a parametric interaction discipline and the

rigour provided by the connectors, there is no need for triggers in our model.

The use of connectors allows for more flexibility in the design of complex systems. This constitutes an advantage not only compared with CNUCE models but with any model which uses simple channels as communication medium. The capability to define a filter in the connectors without the need to change the definition of an interactor can be a desirable feature. Or, as shown in our example, we may construct different configurations from a scenario. The theme of defining different architectures to achieve different interaction effects in the user interface is also one that deserves further research.

As a final note, it should be point out that, when modelling complex interactive systems, the need arises to express dynamic aspects of the user interface, such as user interface components being created and destroyed, or the interconnections between components being changed in runtime. This is a complex area which we have not addressed here. A very preliminary work in this direction was presented in [8]. In that work the basic connectors are enriched with a special connector called orchestrator which is responsible to handle the mobility and the dynamism of the system. We plan to explore this aspect further, as it is one of the main drives for our research in identifying alternative modelling notations for interactive systems.

References

[1] R. Allen and D. Garlan. A formal basis for architectural connection. *ACM TOSEM*, 6(3):213–249, 1997.

[2] F. Arbab. Reo: a channel–based coordination model for component composition. *Mathematical Structures in Comp. Sci.*, 14(3):329–366, 2004.

[3] L. S. Barbosa. Process calculi *à la* Bird-Meertens. In M. L. Andrea Corradini and U. Montanari, editors, *CMCS'01*, volume 44.4, pages 47–66, Genova, April 2001. Elect. Notes in Theor. Comp. Sci., Elsevier.

[4] L. S. Barbosa and J. N. Oliveira. Coinductive interpreters for process calculi. In *Proc. of FLOPS'02*, pages 183–197, Aizu, Japan, September 2002. Springer Lect. Notes Comp. Sci. (2441).

[5] M. Barbosa and L. Barbosa. Specifying software connectors. In K. Araki and Z. Liu, editors, *Proc. First International Colloquim on Theoretical Aspects of Computing (ICTAC'04), Guiyang, China*, pages 53–68. Springer Lect. Notes Comp. Sci. (3407), 2004.

[6] M. A. Barbosa and L. S. Barbosa. A relational model for component interconnection. *Journal of Universal Computer Science*, 10(7):808–823, 2004.

[7] M. A. Barbosa and L. S. Barbosa. Configurations of web services. In *Proceedings of the 5th International Workshop on the Foundations of Coordination Languages and Software Architectures (FOCLASA'06)*, Electr. Notes Theor. Comput. Sci., Bonn, Germany, August 2006. Elsevier. To appear.

[8] M. A. Barbosa and L. S. Barbosa. An orchestrator for dynamic interconnection of software components. In *Proc. 2nd International Workshop on Methods and Tools for Coordinating Concurrent, Distributed and Mobile Systems (MTCoord'06)*, Electr. Notes Theor. Comput. Sci., Bologna, Italy, June 2006. Elsevier. To appear.

[9] J. C. Campos and M. D. Harrison. Model checking interactor specifications. *Automated Software Engineering*, 8(3/4):275–310, August 2001. ISSN: 0928-8910.

[10] D. A. Duce, R. van Liere, and P. J. W. ten Hagen. An approach to hierarchical input devices. *Comput. Graph. Forum*, 9(1):15–26, 1990.

[11] D. J. Duke and M. D. Harrison. Abstract interaction objects. *Computer Graphics Forum*, 12(3):25–36, 1993.

[12] B. Fields, P. Wright, and M. Harrison. Time, tasks and errors. *SIGCHI Bull.*, 28(2):53–56, 1996.

[13] D. Gelernter and N. Carriero. Coordination languages and their significance. *Commun. ACM*, 35(2):97–107, 1992.

[14] C. A. R. Hoare. *Communicating Sequential Processes*. Series in Computer Science. Prentice-Hall International, 1985.

[15] B. Jacobs and J. Rutten. A tutorial on (co)algebras and (co)induction. *EATCS Bulletin*, 62:222–159, 1997.

[16] J. Magee, J. Kramer, and D. Giannakopoulou. Behaviour analysis of software architectures. In *WICSA1: Proc. of the TC2 First Working IFIP Conf. on Software Architecture (WICSA1)*, pages 35–50. Kluwer, B.V., 1999.

[17] R. Milner. *Communicating and Mobile Processes: the π-Calculus*. Cambridge University Press, 1999.

[18] G. Papadopoulos and F. Arbab. Coordination models and languages. In *Advances in Computers — The Engineering of Large Systems*, volume 46, pages 329–400. 1998.

[19] F. D. Paternò. *A Method for Formal Specification and Verification of Interactive Systems*. PhD thesis, Department of Computer Science, University of York, 1995. Available as Technical Report YCST 96/03.

[20] F. Plasil and S. Visnovsky. Behavior protocols for software components. *IEEE Trans. Softw. Eng.*, 28(11):1056–1076, 2002.

[21] P. R. Ribeiro, M. A. Barbosa, and L. S. Barbosa. Generic process algebra: A programming challenge. In *Proc. 10th Brazilian Symposium on Programming Languages*, Itatiaia, Brasil, 2006.