A coordination model for interactive components

Marco A. Barbosa, L. S. Barbosa, José C. Campos

Department of Informatics & Computer Science and Technology Center (CCTC)
Minho University, Portugal
{marco,lsb,jfc}@di.uminho.pt

Abstract. Although presented with a variety of 'flavours', the notion of an *interactor*, as an abstract characterisation of an interactive component, is well-known in the area of formal modelling techniques for interactive systems. This paper replaces traditional, hierarchical, 'tree-like' composition of interactors in the specification of complex interactive systems, by their *exogenous coordination* through general-purpose software *connectors* which assure the flow of data and the meet of synchronisation constraints. The paper's technical contribution is twofold. First a modal logic is defined to express behavioural properties of both interactors and connectors. The logic is new in the sense that its modalities are indexed by fragments of *sets* of actions to cater for action co-occurrence. Then, this logic is used in the specification of both interactors and coordination layers which orchestrate their interconnection.

Keywords: Interactors, coordination models.

1 Introduction

Modern interactive systems resort to increasingly complex architectures of user interface components. With the generalisation of ubiquitous computing, the notion of interactive system itself changed. Single interactive devices have been replaced by frameworks where devices are combined to provide services to a number of different, often anonymous, users accessing them in a competing way. This may explain the increasing interest on rigorous methodologies to develop useful, workable models of such systems. In such a setting, the concept of an *interactor* was originally proposed by Faconti and Paternò [13], as an abstraction for a graphical object capable of both input and output, typically specified in a process algebra. This was further generalised by Duke and Harrison [12] for modelling interactive systems. Interactors become able not only to communicate through i/o ports, but also to convey information about their state through a rendering relation that maps the latter to some presentation medium.

The framework outlined in [12], however, does not prescribe a specification notation for the description of interactor state and behaviour. Several possibilities have been considered. One of them, which directly inspired this piece of research, was developed by the third author in [8] and resorts to Modal Action

Logic (MAL) [19] to specify behavioural constraints. Another one [18] uses LOTOS to express a relation between input and output ports. Actually, the notion of an interactor as a structuring mechanism for formal models of interactive systems, has been an influential one. It has been used, for example, with LOTOS [13,17], Lustre [10], Petri nets [6], Higher Order Processes [11], or Modal Action Logic [9].

Whatever the approach, modelling complex interactive systems entails creating architectures of interconnected interactors. In [8] such models are built hierarchically through 'tree-like' aggregation. Composition is typically achieved by the introduction of additional axioms and/or dedicated interactors to express the control logic for communication. This, in turn, adds dramatically to the complexity of the proposed models. Moreover, it does not promote a clear separation of concerns between modelling interactors and the specification of how they interact with each other.

This is exactly the point where the contribution of this paper may be placed. We adopt an *exogenous coordination* approach to the composition of interactors which entails an effective separation of concerns between the latter and the specification of how they are organised into specific architectures and interact to achieve common goals. Exogenous coordination draws a clear distinction between the *loci* of computational effort and that of interaction control, the latter being blind with respect to the structure of values and procedures which typically depend on the application domains.

Our approach is based on previous work on formal calculi for component coordination published in [4,2] and closely inspired by Arbab's Reo model [1]. In this paper we propose a particular 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 interactors in different ways, but also that the difference between two systems composed out of the same set of interactors must arise out of their composition shemes, *i.e.*, their glue code.

Research reported here is a follow-up of a previous attempt to use the coordination paradigm to express the logic governing the composition of interactors, reported in [3], where a process algebra framework was used to specify connector's behavioural constraints. This, however, proved difficult to smoothly combine with interactors whose evolution is typically given by modal assertions. In this paper an extension to Hennessy-Milner logic [14] is proposed to express behavioural properties of both interactors and connectors. The novelty in the logic is the fact that its modalities are indexed by *sets* of actions to cater for action co-occurrence. Moreover, modalities are interpreted as asserting the existence of transitions which are indexed by a set of actions of which only a subset may be known. Both co-occurrence and such a sort of partial information about transitions seem to be essential for software coordination.

The rest of the paper is organised as follows. Section 2 introduces modal language $\mathbb{M}$, which is used to specify interactors in section 3, and software connectors in section 4. Section 5 brings interactors and the coordination layer together through the notion of a *configuration*. A few examples are discussed to assess the merits of proposed approach. Finally, a few topics for future work are discussed in section 6.

2 A logic for behaviour

2.1 A modal language

Like many other computing artefacts, both interactors and connectors exhibit *reactive* behaviour. They evolve through reaction, either to internal events (*e.g.*, an alarm timeout) or to the accomplishment of interactions with environment (*e.g.*, the exchange of a datum in a channel end). Following a well established convention in formal modelling, we refer to all such reaction points simply as *actions*, collected on a denumerable set *Act*. Then we define modal operators which qualify the validity of logical formaluæ with respect to action occurrence, or, more generally, to action *co-occurrence*.

Having mechanisms to express *co-occurrence* becomes crucial in modelling coordination code. For example, what characterises a *synchronous channel*, the most elementary form of software glue to connect two running interactors, is precisely the fact that any interaction in its input end is simultaneous with another interaction in the output end. Note that temporal simultaneity is understood here as *atomicity*: simultaneous actions cannot be interrupted.

The modal language introduced in the sequel is similar to the well-known Hennessy-Milner logic [14], but for a detail which makes it possible to express (and reason about) action *co-occurrence*. The basic idea is that a formula like $\langle a \rangle \phi$, for $a \in Act$, which in [14] asserts the existence of a transition indexed by a leading to a state which verifies assertion ϕ, is re-interpreted by replacing 'indexed by a' by 'indexed by a set of actions of which a is part of'. Therefore, modalities are relative to sets of actions, whose elements are represented by juxtaposition, regarded as *factors* of a (eventually larger) compound action.

In detail, modalities are indexed by either *positive* or *negative* action *factors*, denoted by K and $\sim K$, for $K \subseteq Act$, respectively. Intuitively, a *positive* (respectively, *negative*) factor refers to transitions whose labels include (respectively, exclude) all actions in it. Annotation $\sim$ may be regarded as an involution over $\mathcal{P}(Act)$ (therefore, $\sim\sim K = K$).

Formally $\mathbb{M}$ has the following syntax, where W is a positive or negative action factor and Ψ ranges over elementary propositions,

$$\phi \ ::= \Psi \ | \ \mathsf{true} \ | \ \mathsf{false} \ | \ \phi_1 \wedge \phi_2 \ | \ \phi_1 \vee \phi_2 \ | \ \phi_1 \rightarrow \ \phi_2 \ | \ \langle W \rangle \phi \ | \ [W]\phi$$

Its semantics is given by a satisfiability relation wrt to system's states. For the non modal part this is as one would expect: for example $s \models \mathsf{true}$, $s \not\models \mathsf{false}$ and $s \models \phi_1 \wedge \phi_2 \Leftrightarrow s \models \phi_1 \ \wedge \ s \models \phi_2$. For the modal connectives, we define

$$s \models \langle W \rangle \phi \ \Leftrightarrow \ \langle \exists \ s' \ : \ \langle \exists \ \theta \ : \ s \xrightarrow{\theta} s' : \ W \prec \theta \rangle : \ s' \models \phi \rangle$$

$$s \models [W] \phi \ \Leftrightarrow \ \langle \forall \ s' \ : \ \langle \exists \ \theta \ : \ s \xrightarrow{\theta} s' : \ W \prec \theta \rangle : \ s' \models \phi \rangle$$

where

$$W \prec X \triangleq \begin{cases} W = K, \text{for } K \subseteq Act \;\;\Rightarrow\;\; K \subseteq X \\ W = \sim K, \text{for } K \subseteq Act \;\;\Rightarrow\;\; K \not\subseteq X \end{cases}$$

For example, if there exists a state s' such that $s \xrightarrow{abcd} s'$ and s' verifies some formula ϕ, then $s \models \langle bd \rangle \phi$. Dually, assertion $[\sim abc]\mathsf{false}$ states that all transitions whose labels do not involve, at least and simultaneously, actions in set $\{a, b, c\}$ lead to states which validate false and their occurrence is, therefore, impossible.

Modal connectives can be extended to *families* of both 'positive' or 'negative' action factors as follows:

$$s \models \langle F \rangle \phi \Leftrightarrow \langle \exists\, W \;:\; W \in F : \; \langle W \rangle \phi \rangle$$
$$s \models [F] \phi \Leftrightarrow \langle \forall\, W \;:\; W \in F : \; [W] \phi \rangle$$

where $F \subseteq (\mathcal{P}(Act) \cup \sim\mathcal{P}(Act))$. Just as actions in an action factor are represented by juxtaposition, as in $\langle abc \rangle$, action factors in a family thereof are separated by commas, as in $\langle J, K, L \rangle$. Set complement to $\mathcal{P}(Act) \cup \sim\mathcal{P}(Act)$ is denoted by symbol $-$ as in $[-K]\mathsf{false}$ or $\langle - \rangle\mathsf{true}$, the latter abbreviating $-\emptyset$. The first assertion states that only transitions exactly labelled by factor K can occur. The second one that there exists, from the current state, at least a possible transition (of which no particular assumption is made).

Most results on Hennessy-Milner logic carry over $\mathbb{M}$. In particular, it can be shown that modal equivalence in $\mathbb{M}$ entails bisimulation equivalence for processes in CCS-like calculus extended with action co-occurrence. Although this is not the place to explore the structure of $\mathbb{M}$, the following *extension* laws are needed in the sequel: for all $a, a' \in Act$, $K, K' \subseteq Act$,

$$[a]\phi \;\Leftarrow\; [aa']\phi \quad \text{and} \quad \langle a \rangle \phi \;\Leftarrow\; \langle aa' \rangle \phi \tag{1}$$
$$[K]\phi \;\Leftarrow\; [K, K']\phi \quad \text{and} \quad \langle K \rangle \phi \;\Rightarrow\; \langle K, K' \rangle \phi \tag{2}$$
$$[K]\phi \wedge [K']\phi \;\Leftrightarrow\; [K, K']\phi \tag{3}$$
$$\langle K \rangle \phi \vee \langle K' \rangle \phi \;\Leftrightarrow\; \langle K, K' \rangle \phi \tag{4}$$

Proofs proceed by unfolding definitions. For example, the first part of (1) is proved as follows:

$$
\begin{array}{ll}
 & s \;\models\; [a]\phi \\
\Leftrightarrow & \quad \{ \text{ definition } \} \\
 & \langle \forall\; s' \;:\; \langle \exists\; \theta \;:\; s \xrightarrow{\theta} s' : \; \{a\} \subseteq \theta \rangle : \; s' \models \phi \rangle \\
\Leftarrow & \quad \{ \text{ set inclusion } \} \\
 & \langle \forall\; s' \;:\; \langle \exists\; \theta \;:\; s \xrightarrow{\theta} s' : \; \{a, a'\} \subseteq \theta \rangle : \; s' \models \phi \rangle \\
\Leftrightarrow & \quad \{ \text{ definition } \} \\
 & s \;\models\; [aa']\phi
\end{array}
$$

It is also easy to see that, for K and K', both positive or both negative,

$$[K, K']\phi \;\Rightarrow\; [K \cup K']\phi \tag{5}$$

$$\langle K, K'\rangle\phi \;\Leftarrow\; \langle K \cup K'\rangle\phi \tag{6}$$

2.2 Typical properties

To exemplify the use of the logic and introduce some notation to be used in the sequel, let us consider a number of properties useful for the specification of both interactors and coordination schemes. Most of the latter are designed to preclude interactions in which some action factor K is absent. This leads to the following property schemes

$$\mathsf{only}\, K \;\triangleq\; [\sim K]\mathsf{false} \quad \text{and} \quad \mathsf{forbid}\, K \;\triangleq\; \mathsf{only}\; \sim K$$

Properties above entails conciseness in expression. For example, assertion $\mathsf{only}\, K \wedge \mathsf{only}\, L \wedge \mathsf{forbid}\, M$ abbreviates, by (3), to $\mathsf{only}\, K, L, \sim M$. A dual property asserts the existence of at least a transition of which a particular action pattern is a factor, *i.e.*, $\mathsf{perm}\, K \;\triangleq\; \langle K\rangle\mathsf{true}$. Or, not only possible, but also mandatory, $\mathsf{mandatory}\, K \;\triangleq\; \langle -\rangle\mathsf{true} \,\wedge\, \mathsf{only}\, K$.

More complex patterns of behaviour are expressed by nesting modalities, as in $[K]\langle L\rangle\phi$, which expresses a sort of invariant: after every occurrence of an action with factor K, there is, at least, a transition labelled by actions in L which validates ϕ. The complement of $\langle -\rangle\mathsf{true}$ is $[-]\mathsf{false}$ which asserts no transition is possible. Notice that their duals — $\langle -\rangle\mathsf{false}$ and $[-]\mathsf{true}$ — are just abbreviations of constants false and true, respectively.

3 $\mathbb{M}$-interactors

3.1 A language for $\mathbb{M}$-interactors

As stated in the Introduction, our aim is to use a single specification notation for both *interactors*, which, in this setting, correspond to the computational entities, and *connectors*, which cater for the coordination of the former. Modal language $\mathbb{M}$ is, of course, our candidate for this double job — this section focuses on its first part.

The definition of a $\mathbb{M}$-interactor is adapted from [12], but for the choice of the behaviour specification language. Formally,

Definition 1. *An interactor signature is a triple (S, α, Γ), where S is a set of sorts, α a S-indexed family of attribute symbols, and Γ a set of action symbols. An $\mathbb{M}$-interactor is a tuple $(\Delta, \rho, \gamma, Ax_\Delta)$ where Δ is an interactor signature, $\rho : \mathbb{P} \longleftarrow \alpha$ and $\gamma : \mathbb{P} \longleftarrow \Gamma$ are rendering relations, from attributes and actions, respectively, to some presentation medium $\mathbb{P}$, and Ax_Δ a set of axioms over Δ expressed in the $\mathbb{M}$ language.*

The set of ports provided by an interactor is defined by ρ, γ, and Γ. Ports induced by ρ are output ports used to read the value of attributes and are always available. This condition is expressed by $\langle \forall\ p\ :\ p \in \mathsf{range}\,\rho :\ \langle p \rangle \mathsf{true} \rangle$. Ports in Γ are input/output ports and their availability is governed by axioms in Ax_{Δ}.

Syntactically, the definition of an interactor has three main declarations: of attributes, actions and axioms. The first two define the signature. The rendering relation is given by annotations on the attributes. Actions can also be annotated to assert whether or not that they are available to the user. Fig. 1 shows a very simple example of an interactor modelling an application window. Two attributes are declared, indicating whether the window is visible or displays new information.

interactor *window*

attributes

$\boxed{\text{vis}}$ *visible*, *newinfo* : *bool*

actions

hide show update invalidate

axioms

$[hide]\ \neg visible$

$[show]\ visible$

$[update]\ newinfo$

$[invalidate]\ \neg newinfo$

forbid *hide show*

forbid *update invalidate*

Fig. 1. A window interactor

Available actions model the change of visibility and information displayed in the window. Their effect in the state of the interactor is defined by the axioms in the figure. In this example, the rendering relation is defined by the $\boxed{\text{vis}}$ annotation, which indicates that all attributes are (visually) perceivable.

Although the behavioural properties specified in this example are rather simple, in general, it is necessary to specify when actions are permitted or required to happen. This is achieved with the perm and mandatory assertions, typically stated in a guarded context. Thus,

- perm $K \rightarrow \Phi$, where Φ is a non modal proposition over the state space of the interactor, as perceived by the values of its attributes. The assertion means that *if actions containing action factor K are permitted then Φ evaluates to* true.
- $\Phi \rightarrow$ mandatory K, meaning *actions containing action factor K are inevitable whenever Φ evaluates to* true.

A useful convention establishes that permissions, but not obligations, are asserted by default. I.e., by default anything can happen, but nothing must happen. This facilitates makes adding or removing permissions and obligations incrementally when writing specifications.

3.2 Composing interactors

In the literature, and specifically in [8], interactors are composed in the 'classical' way, *i.e.*, by a specification *import* mechanism, illustrated below by means

```
interactor $space$
  attributes
    $\boxed{\text{vis}}$ $state : \{open, closed\}$
  actions
    $open\ close$
  axioms
    perm $open \rightarrow state = closed$
    $[open]\ state = open$
    perm $close \rightarrow state = open$
    $[close]\ state = closed$
```

Fig. 2. The space interactor

```
interactor $spaceSign$
  aggregates
    $window$ via $oI$
    $window$ via $cI$
  attributes
    $\boxed{\text{vis}}$ $state : \{open, closed\}$
  actions
    $\boxed{\text{vis}}$ $open\ close$
  axioms
    perm $open \rightarrow state = closed$
    $[open]\ state' = open$
    perm $close \rightarrow state = open$
    $[close]\ state' = closed$
    only $open\ oI.update\ oI.show$ $\vee$ only $close\ cI.update\ cI.show$
```

Fig. 3. A classical solution

of a small example. In the literature, and specifically in [8], interactors are composed in the 'classical' way, *i.e.*, by a specification *import* mechanism, illustrated below by means of a small example. This will be contrasted in section 5 to a coordination-based solution. Consider a system that controls access to a specific space (e.g, an elevator), modelled by the interactor in Fig. 2. Now suppose two indicators have to be added to this model, one to announce *open* events, the other to signal *close* events. We will use instances of the window interactor from Fig. 1 to act as indicators. The 'classical' aggregation strategy, as in [8], requires that two instances of the *window* interactor be imported into one instance of *space* to build the new interactor. The rules that govern their incorporation are as follows:

– the *open* (respectively, *close*) indicator must be made visible and have its information updated whenever the system is opened (respectively, closed).

Additionally, it should be noted that whenever a window is made visible, it might overlap (and hide) another one. The resulting interactor is presented in figure

3, where a new axiom expresses the coordination logic. The fact that $\mathbb{M}$ allows for action co-occurrence means that constraints on actions become simpler and more concise than their MAL counterparts, as used in [8]: in our example only an additional axiom is needed. Nevertheless, this solution still mixes concerns by expressing the coordination of interactors cI and oI at the same level than the internal properties of the underlying *space* interactor. How such two levels can be disentangled is the topic of the following sections.

4 The coordination layer

Actually, coordination entails a different perspective. As in [1] this is achieved through specific *connectors* which abstract the idea of an intermediate *glue code* to handle interaction. Connectors have *ports*, thought of as *interface points* through which messages flow. Each port has an *interaction polarity* (either *input* or *output*), but, in general, connectors are blind with respect to the data values flowing through them. The set of elementary interactions of a connector $\mathbb{C}$ forms its *sort*, denoted by $\mathsf{sort}.[\![\mathbb{C}]\!]$. By default the sort of $\mathbb{C}$ is the set of its ports, but often such is not the case. For example, a synchronous channel with ports a and a' has a unique possible interaction: the simultaneous activation of both a and a', represented by aa'.

Connectors are specified at two levels: the *data* level, which records the flow of data, and the *behavioural* one which prescribes all the activation patterns for ports. Formally, let $\mathbb{C}$ be a connector with m input and n output ports. Assume $\mathbb{D}$ as a generic type of data values and $\mathbb{P}$ as a set of (unique) *port identifiers.* Then,

Definition 2. *The specification of a connector* $\mathbb{C}$ *is given by a relation* $\mathsf{data}.[\![\mathbb{C}]\!] : \mathbb{D}^n \longleftarrow \mathbb{D}^m$*, which relates data present at its* m *input ports with data at its* n *output ports, and an* $\mathbb{M}$ *assertion,* $\mathsf{port}.[\![\mathbb{C}]\!]$*, over its sort,* $\mathsf{sort}.[\![\mathbb{C}]\!]$*, which specifies the relevant properties of its port activation pattern.*

4.1 Elementary connectors.

The most basic connector is the *synchronous channel* which exhibits two ports, a and a', of opposite polarity. This connector forces input and output to become mutually blocking. Formally, $\mathsf{data}.[\![\ a \longrightarrow a'\ ]\!] \ =\ \mathsf{Id}_{\mathbb{D}}$, i.e., the identity relation in $\mathbb{D}$, and

$$\mathsf{sort}.[\![\ a \longrightarrow a'\ ]\!] \ =\ \{aa'\} \qquad \mathsf{port}.[\![\ a \longrightarrow a'\ ]\!] \ =\ \mathsf{only}\, aa'$$

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

Any coreflexive relation provides channels which can loose information, thus modelling unreliable communications. Therefore, we define, an *unreliable channel* as $\mathsf{data}.[\![\ a \xrightarrow{\diamond} a'\ ]\!] \ \subseteq\ \mathsf{Id}_{\mathbb{D}}$ and

$$\mathsf{sort}.[\![\ a \xrightarrow{\diamond} a'\ ]\!] \ =\ \{a, aa'\} \qquad \mathsf{port}.[\![\ a \xrightarrow{\diamond} a'\ ]\!] \ = \mathsf{only}\, a$$

The behaviour expression states that all valid transitions involve input port a, although not necessarily a'. This corresponds either to a successful communication, represented by the simultaneous activation of both ports, or to a failure, represented by the single activation of the input port.

As an example of a connector which is not stateless consider *fifo*$_1$, a channel with a buffer of a single position. Formally, $\mathsf{data}.[\![\ a \mathrel{-\!\square\!\!\rightarrow} a'\ ]\!] \ = \ \mathsf{Id}_{\mathbb{D}}$ and

$$\mathsf{sort}.[\![\ a \mathrel{-\!\square\!\!\rightarrow} a'\ ]\!] \ = \ \{a, a'\} \quad \mathsf{port}.[\![\ a \mathrel{-\!\square\!\!\rightarrow} a'\ ]\!] \ = \ [a]\mathsf{only}\, a', {\sim}a$$

Notice that its port specification equivales to $[a](\mathsf{only}\, a' \wedge \mathsf{forbid}\, a)$, formalising the intuition of a strict alternation between the activation of ports a and a'.

If channels forward information, *drains* absorb it. However they play a fundamental role in controlling the flow of data along the coordination code. A *drain* has two input, but no output, ports. Therefore, it looses 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 data part coincides in both connectors: $\mathbb{D} \times \mathbb{D}$. Then

$$\mathsf{sort}.[\![\ a \overset{\blacktriangledown}{\vdash\!\!\!\longrightarrow\!\!\!\dashv} a'\ ]\!] \ = \ \{aa'\} \quad \mathsf{port}.[\![\ a \overset{\blacktriangledown}{\vdash\!\!\!\longrightarrow\!\!\!\dashv} a'\ ]\!] = \mathsf{only}\, aa'$$

$$\mathsf{sort}.[\![\ a \overset{\triangledown}{\vdash\!\!\!\longrightarrow\!\!\!\dashv} a'\ ]\!] \ = \ \{a, a'\} \quad \mathsf{port}.[\![\ a \overset{\triangledown}{\vdash\!\!\!\longrightarrow\!\!\!\dashv} a'\ ]\!] = \mathsf{only}\, a, a' \ \wedge \ \mathsf{forbid}\, aa'$$

4.2 New connectors from old

Connectors can be combined in three different ways: by placing them side-by-side, by sharing ports or introducing feedback wires to connect output to input ports. In the sequel, note that behaviour annotations in the specification of connectors can always be presented in a *disjunctive* form

$$\mathsf{port}.[\![\mathbb{C}]\!] \ = \ \phi_1 \vee \phi_2 \vee \cdots \vee \phi_n \tag{7}$$

where each ϕ_i is a conjunction of

$$\underbrace{[K] \cdots [K]}_{n} \ \mathsf{only}\, F$$

Also let $t_{|a}$ and $t_{\#a}$, for $t \in \mathbb{D}^n$ and $a \in \mathcal{P}$, represent, respectively, a tuple of data values t from which the data corresponding to port a has been deleted, and the tuple component corresponding to such data. Then,

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

$$\begin{aligned}
\mathsf{data}.[\![\mathbb{C}_1 \ \boxtimes \ \mathbb{C}_2]\!] \ &= \mathsf{data}.[\![\mathbb{C}_1]\!] \ \times \ \mathsf{data}.[\![\mathbb{C}_2]\!] \\
\mathsf{sort}.[\![\mathbb{C}_1 \ \boxtimes \ \mathbb{C}_2]\!] \ &= \mathsf{sort}.[\![\mathbb{C}_1]\!] \ \cup \ \mathsf{sort}.[\![\mathbb{C}_2]\!] \\
\mathsf{port}.[\![\mathbb{C}_1 \ \boxtimes \ \mathbb{C}_2]\!] \ &= \mathsf{port}.[\![\mathbb{C}_1]\!] \ \vee \ \mathsf{port}.[\![\mathbb{C}_2]\!]
\end{aligned}$$

The relevance of sorts becomes now clear. Take, for example, the aggregation of two synchronous channels Their joint behaviour is

$$\mathsf{port}.[\![(\ a \longrightarrow a' \ \boxtimes \ c \longrightarrow c' \ )]\!] \ = \ \mathsf{only}\, aa' \ \vee \ \mathsf{only}\, cc'$$

A transition labelled by, say, $aa'c$ does not violate the behaviour prescribed above, but it is made invalid by the sort specification, which is $\{aa', cc'\}$.

Share. The effect of *share* is to plug ports with identical polarity. The aggregation of *output* ports is done by a *right share* $(\mathbb{C}\ {}^{i}_{j} > z)$, where $\mathbb{C}$ is a connector, 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 input from both ports is received at same time the combinator chooses one of them in a non-deterministic way. Let $\mathsf{data}.[\![\mathbb{C}]\!] : \mathbb{D}^n \longleftarrow \mathbb{D}^m$. Then, the data flow relation $\mathsf{data}.[\![\mathbb{C}\ {}^{i}_{j} > z]\!] : \mathbb{D}^{n-1} \longleftarrow \mathbb{D}^m$ for this operator is given by

$$r\,(\mathsf{data}.[\![\mathbb{C}\ {}^{i}_{j} > z]\!])\,t \ \Leftrightarrow \ t'\,(\mathsf{data}.[\![\mathbb{C}]\!])\,t \ \ \wedge \ \ r_{|z} = t'_{|i,j} \ \wedge \ (r_{\#z} = t'_{\#i} \ \vee \ r_{\#z} = t'_{\#j})$$

At the behavioural level, its effect is that of a renaming applied to the $\mathbb{M}$-formula capturing the behavioural patterns of $\mathbb{C}$, *i.e.*,

$$\mathsf{port}.[\![(\mathbb{C}\ {}^{i}_{j} > z)]\!] \ = \ \{z \leftarrow i, z \leftarrow j\}\ \mathsf{port}.[\![\mathbb{C}]\!]$$

over

$$\mathsf{sort}.[\![(\mathbb{C}\ {}^{i}_{j} > z)]\!] \ = \ \{z \leftarrow i, z \leftarrow j\}\ \mathsf{sort}.[\![\mathbb{C}]\!]$$

Figure 4 represents a *merger* formed by sharing the output ports of a synchronous channel and a 1-place buffer.

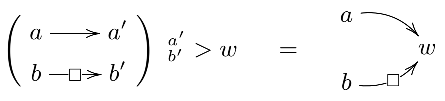

Fig. 4. A *merger*: $\mathsf{only}\, aw \vee [b]\mathsf{only}\, w, {\sim}b$.

On the other hand, aggregation of *input* ports is achieved by a *left share* mechanism $(z <^{i}_{j} \mathbb{C})$. This behaves like a *broadcaster* sending synchronously messages from z to both i and j. This case is slightly more complex: before renaming, all computations of $\mathbb{C}$ in which ports i and j are activated independently of each other must be synchronised. Therefore, we take all disjuncts in $\mathsf{port}.[\![\mathbb{C}]\!]$ in which ports i and j are involved, form their conjunction to force co-occurrence, and apply renaming. Formally, let ϕ_θ be a disjunct in $\mathsf{port}.[\![\mathbb{C}]\!]$

(recall (7)) involving only ports in θ. Define $\phi_i = \langle \bigvee \ \phi_\theta \in \mathsf{port}.[\![\mathbb{C}]\!] \ : \ i \in \theta : \ \phi_\theta \rangle$ and, similarly, ϕ_j. Therefore, for $\sigma = \{z \leftarrow i, z \leftarrow j\}$,

$$\mathsf{port}.[\![(z <^i_j \mathbb{C})]\!] \ = \ \sigma(\phi_i \wedge \phi_j) \ \vee \ \langle \bigvee \ \phi_{\theta'} \in \mathsf{port}.[\![\mathbb{C}]\!] \ : \ i \notin \theta' \wedge j \notin \theta' : \ \phi_{\theta'} \rangle$$

and, again,

$$\mathsf{sort}.[\![(z <^i_j \mathbb{C})]\!] \ = \ \{z \leftarrow i, z \leftarrow j\} \, \mathsf{sort}.[\![\mathbb{C}]\!]$$

On the other hand, relation $\mathsf{data}.[\![z <^i_j \mathbb{C}]\!] : \mathbb{D}^n \longleftarrow \mathbb{D}^{m-1}$ is given by

$$t' \, (\mathsf{data}.[\![z <^i_j \mathbb{C}]\!]) \, r \ \leftrightarrow \ t' \, (\mathsf{data}.[\![\mathbb{C}]\!]) \, t \ \wedge \ r_{|z} = t_{|i,j} \ \wedge \ r_{\# z} = t_{\# i} = t_{\# j}$$

As an example let us calculate the sharing of input ports a and b in a connector composed by three, otherwise non interfering, synchronous channels,

$$\begin{aligned}
& \mathsf{port}.[\![z <^a_b \ (\ a \longrightarrow a' \ \boxtimes \ b \longrightarrow b' \ \boxtimes \ c \longrightarrow c' \ )]\!] \\
\Leftrightarrow \quad & \quad \{ \ \text{definition} \ \} \\
& \{z \leftarrow a, z \leftarrow b\}(\mathsf{only}\, aa' \wedge \mathsf{only}\, bb') \ \vee \ \mathsf{only}\, cc' \\
\Leftrightarrow \quad & \quad \{ \ \text{renaming and (3)} \ \} \\
& \mathsf{only}\, za', zb' \ \vee \ \mathsf{only}\, cc' \\
\Leftrightarrow \quad & \quad \{ \ (5) \ \} \\
& \mathsf{only}\, za'b' \ \vee \ \mathsf{only}\, cc'
\end{aligned}$$

which asserts that input on z co-occurs with output at both a' and b'. Replacing $b \longrightarrow b'$ by a one-place buffer leads to the connector depicted in Fig. 5 which is calculated as follows

$$\begin{aligned}
& \mathsf{port}.[\![z <^a_b \ (\ a \longrightarrow a' \ \boxtimes \ b \ -\!\square\!\!\rightarrow b' \ \boxtimes \ c \longrightarrow c' \ )]\!] \\
\equiv \quad & \quad \{ \ \text{definition} \ \} \\
& \{z \leftarrow a, z \leftarrow b\}(\mathsf{only}\, aa' \wedge [b]\mathsf{only}\, b', {\sim}b) \ \vee \ \mathsf{only}\, cc' \\
\equiv \quad & \quad \{ \ \text{renaming} \ \} \\
& (\mathsf{only}\, za' \wedge [z]\mathsf{only}\, b', {\sim}b) \ \vee \ \mathsf{only}\, cc'
\end{aligned}$$

Hook. This combinator encodes a *feedback* mechanism, drawing a direct connection between an output and an input port. This has a double consequence: the connected ports must be activated simultaneously and become externally non observable. The formal definition is omitted here (but see [5]) because this combinator is not used in the examples to follow. For the sake of curiosity, note the following 'extreme' situations arising from hooking a synchronous channel and a 1-place buffer, respectively,

$$(\mathsf{only}\, aa') \ \Lsh^a_{a'} \ = \ \mathsf{only}\, \emptyset \ = \ \mathsf{true}$$

$$[a](\mathsf{only}\, a', {\sim}a) \ \Lsh^a_{a'} \ = \ [\emptyset](\mathsf{true} \wedge \mathsf{false}) \ = \ \mathsf{false}$$

as one may have expected given the buffer strict alternation activation discipline.

$$z <^a_b \begin{pmatrix} a \longrightarrow a' \\ b \xrightarrow{\square} b' \\ c \longrightarrow c' \end{pmatrix} \quad = \quad \begin{tikzcd} & a' \\ z \arrow[ur, bend left] \arrow[dr, "\square"] & \\ & b' \\ c \arrow[r] & c' \end{tikzcd}$$

Fig. 5. A *broadcaster* and a detached channel.

5 Configurations of 𝕄-interactors

Having introduced 𝕄-interactors and the coordination layer on top of the same modal language, we may now complete the whole picture. The key notion is that of a *configuration*, *i.e.*, a collection of *interactors* interconnected through a *connector* built from elementary connectors, combined trough the combinators defined above. Formally,

Definition 3. *A configuration is a tuple $\langle I, \mathbb{C}, \sigma \rangle$, where $I = \{I_i |\ i \in n\}$ is a collection of interactors, $\mathbb{C}$ is a connector and σ a mapping of ports in I to ports in $\mathbb{C}$. The behaviour of a configuration is given by the conjunction of the modal theories for each $I_n \in I$, as specified by their axioms, and the port specification* $\mathsf{port}.[\![\mathbb{C}]\!]$ *of connector $\mathbb{C}$, after renaming by σ.*

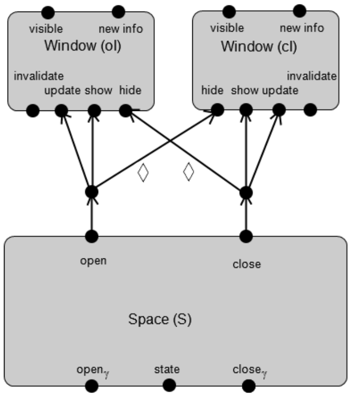

Fig. 6. A coordination-based solution.

To illustrate the envisaged approach, consider again the example discussed in section 3. A coordination-based solution, depicted in Fig. 6, replaces the hierarchical import of *window* into *spaceSign* interactor, by a configuration in which the two instances of the former and one instance of the original *space* interactor are connected by

$$\mathbb{BC} \triangleq \mathbb{B} \boxtimes \mathbb{B}$$

a connector which joins together two broadcasters $\mathbb{B}$. Each $\mathbb{B}$ is formed by two synchronous channels and a lossy channel, sharing their input ports, *i.e.*

$$\mathbb{B} \triangleq z <^w_c (w <^a_b (\ a \longrightarrow a' \boxtimes b \longrightarrow b') \boxtimes c \xrightarrow{\diamond} c')$$

An easy calculation yields $\mathsf{port}.[\![\mathbb{B}]\!] = \mathsf{only}\, za', zb', z$, which, by (5), equivales to $\mathsf{only}\, za'b'$. In a configuration in which, through a renaming σ, port z is linked to $S.open$, a' to $oI.update$, b' to $oI.show$ and c' to $cI.hide$, one may prove (*i.e.*, discover, rather than assert) a number of desirable properties of the configuration. For example, from axiom $\mathsf{perm}\, S.open$, a default axiom of interactor *space* in section 3, and $\sigma\, \mathsf{only}\, za'b'$, one concludes that

$$\mathsf{perm}\, S.open\ oI.update\ oI.show$$

i.e., there are transitions in which all the three ports are activated at the same time. But, because the connector does not allow actions not including the simultaneous activation of such three ports, the joint behaviour of the configuration asserts not only possibility but also necessity of this transition, *i.e.*,

$$\mathsf{perm}\, S.open\ oI.update\ oI.show\ \wedge\ \mathsf{only}\, S.open\ oI.update\ oI.show$$

This is stronger than the corresponding axiom added to interactor *spaceSign* in Fig. 3, although it can be deduced from the modal theory of this interactor (which, of course, includes $\mathsf{perm}\, open$). Note we are focussing only on one of the two $\mathbb{B}$ connectors in $\mathbb{BC}$, thus this conclusion does not interfere with a similar possibility for the other connection of interactor instances S and cI (recall the behavioural effect of $\boxtimes$ is disjunction).

On the other hand, one also has $\mathsf{perm}\, S.open\, cI.hide$ because action zc' is in $\mathsf{sort}.[\![\mathbb{B}]\!]$, but, now only as a possibility, because an unreliable channel was used to connect these ports. From this property and $\mathsf{only}\, S.open\, oI.update\, oI.show$ above, we can easily conclude that $cI.hide$ cannot occur independently of $S.open$, $oI.update$ and $oI.show$. Again, this is stronger than the interactor model in Fig. 3, where the hide action was left unrestricted.

As a final example, consider an interactor which has to receive the location coordinates supplied by two different input devices but in strict alternation. The connector to plug these three interactors is the *alternate merger* depicted in Fig. 7, formally, defined as

$$b <^{d'}_{f} a <^{d}_{c} (\ c \longrightarrow c'\ \boxtimes\ d \xrightarrow{\blacktriangledown} d'\ \ \boxtimes\ f \;{-}\!\square\!{\rightarrow}\; f')^{c'}_{f'} > w$$

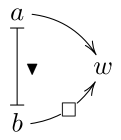

Fig. 7. An *alternate merger.*

Its behavioural pattern is

$$\mathsf{port}.[\![\mathbb{AM}]\!]\ =\ \mathsf{only}\, awb\ \wedge\ [b]\mathsf{only}\, w, {\sim}b$$

Clearly, each activation of port a is synchronous with b and w. But then data received in b (say, the coordinates of the one of the devices) is stored in the buffer. Next action is necessarily w, whose completion empties the buffer.

6 Conclusions and Future Work

It was our intention to set the foundations for an approach to modelling interactive systems entailing a true separation of concerns between modelling of individual components (interactors) and their architectural organisation. For this a new modal logic (the $\mathbb{M}$ language) was introduced, which is similar to the Hennessy-Milner logic [14] but for the fact that its modal connectives are indexed by sets of actions (*actions factors*). These action factors are interpreted over the compound actions (themselves represented by sets) that label transitions using set inclusion. This makes it possible to express properties over co-occurring actions in the logic.

Although the main drive behind the development of $\mathbb{M}$ was the need for a modal logic expressive enough to define the coordination layer, the language was also used to specify interactors, thus providing a single language for expressing the behaviour of both interactors and connectors that bind them.

The approach presents two major benefits over [13] or [8]. First of all, it promotes a clear separation of concerns between the specification of the individual interactors and the specification of how they interact with each other. Furthermore, it frees us from the rigid structure imposed by hierarchical organisation.

At this point, it is worthwhile pointing out that when composing interactors into different configurations, the resulting behaviour becomes an emergent feature of the model. Hence, we discover, rather than assert, what the system will be like. This is particularly relevant in a context were one is interested in exploring the impact of different design decisions at the architectural level. Recent related work on the use of (some type of) logic to specify component behaviour include [7] and [15], the latter with an emphasis on property verification.

A number of lines of research have been opened by the current endeavour. A main one concerns temporal extension. Actually, language $\mathbb{M}$ seems expressive enough to express connector's behaviour, but not so when facing more elaborate interactor's specifications. A typical case relates to expressing *obligation* requirements. We are currently studying how $\mathbb{M}$ can be extended in a way similar to D. Kozen's μ-calculus [16] in order to address these temporal issues.

Acknowledgements. The authors wish to thank Michael D. Harrison for useful comments on a preliminary version of this paper. This work is funded by ERDF - European Regional Development Fund through the COMPETE Programme (operational programme for competitiveness) and by National Funds through the FCT - the Portuguese Foundation for Science and Technology, within project FCOMP-01-0124-FEDER-007254.

References

1. F. Arbab. Reo: a channel–based coordination model for component composition. *Mathematical Structures in Comp. Sci.*, 14(3):329–366, 2004.
2. M. A. Barbosa and L. S. Barbosa. A relational model for component interconnection. *Journal of Universal Computer Science*, 10(7):808–823, 2004.

3. M. A. Barbosa, L. S. Barbosa, and J. C. Campos. Towards a coordination model for interactive systems. In A. Cerone and P. Curzon, editors, *FMIS 2007: Proc. 1st Inter. Workshop in Formal Methods for Interactive Systems*, volume 347 of *Electronic Notes in Theoretical Computer Science*, pages 89–103. Elsevier, 2007.
4. M.A. Barbosa and L.S. 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.
5. Marco António Barbosa. *Specification and Refinement of Software Connectors (to appear)*. PhD thesis, DI, Universidade do Minho, 2009.
6. Rémi Bastide, David Navarre, and Philippe A. Palanque. A tool-supported design framework for safety critical interactive systems. *Interacting with Computers*, 15(3):309–328, 2003.
7. J. K. F. Bowles and S. Moschoyiannis. Concurrent logic and automata combined: A semantics for components. In C. Canal and M. Viroli, editors, *Proc. of FOCLASA'06*, volume 175 (2), pages 135–151. Elsevier, 2007.
8. J. C. Campos and M. D. Harrison. Model checking interactor specifications. *Automated Software Engineering*, 8(3/4):275–310, August 2001. ISSN: 0928-8910.
9. J. C. Campos and M. D. Harrison. Systematic analysis of control panel interfaces using formal tools. In *XVth International Workshop on the Design, Verification and Specification of Interactive Systems (DSV-IS 2008)*, pages 72–85. Springer Lect Notes in Comp. Sci. (5136), July 2008.
10. Bruno d'Ausbourg, Christel Seguin, Guy Durrieu, and Pierre Roché. Helping the automated validation process of user interfaces systems. In *ICSE '98: Proc. 20th Inter. Conf. on Software Engineering*, pages 219–228. IEEE Computer Society, 1998.
11. Anke Dittmar and Peter Forbrig. A unified description formalism for complex hci-systems. In *SEFM '05: Proc. 3rd IEEE Inter. Conf. on Software Engineering and Formal Methods*, pages 342–351. IEEE Computer Society, 2005.
12. David J. Duke and Michael D. Harrison. Abstract interaction objects. *Computer Graphics Forum*, 12(3):25–36, 1993.
13. G. Faconti and F. Paternò. An approach to the formal specification of the components of an interaction. In C. Vandoni and D. Duce, editors, *Eurographics '90*, pages 481–494. North-Holland, 1990.
14. M. C. Hennessy and A. J. R. G. Milner. Algebraic laws for non-determinism and concurrency. *Journal of ACM*, 32(1):137–161, 1985.
15. E. B. Johnsen, O. Owe, and A. B. Torjusen. Validating behavioural component interfaces in rewriting logic. volume 159, pages 187–204. Elsevier, 2006.
16. D. Kozen. Results on the propositional μ-calculus. *Theor. Comp. Sci.*, (27):333–354, 1983.
17. P. Markopoulos. On the expression of interaction properties within an interactor model. In *In P. Palanque and R. Bastide (eds.), Design, Specification and Verification of Interactive Systems'95*, 1995.
18. Fabio 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.
19. Mark Ryan, José Fiadeiro, and Tom Maibaum. Sharing actions and attributes in Modal Action Logic. In T. Ito and A. R. Meyer, editors, *Theoretical Aspects of Computer Software*, pages 569–593. Springer Lect. Notes Comp. Sci. (526), 1991.