ELECTRONIC WORKSHOPS IN COMPUTING

Series edited by Professor C.J. van Rijsbergen

C.R. Roast and J.I. Siddiqi, Sheffield Hallam University, UK (Eds)

BCS-FACS Workshop on Formal Aspects of the Human Computer Interface

Proceedings of the BCS-FACS Workshop on Formal Aspects of the Human Computer Interface, Sheffield Hallam University, 10-12 September 1996

On the Composition of Interactor Specifications

P. Markopoulos, J. Rowson and P. Johnson

Published in Collaboration with the
British Computer Society

On the Composition of Interactor Specifications

Panos Markopoulos, Jon Rowson and Peter Johnson

Department of Computer Science

Queen Mary and Westfield College

London E1 4NS

markop@dcs.qmw.ac.uk

Abstract

This paper discusses the composition of interface objects, a concept of direct relevance to user interface software architectures and interface design practice. Composition is discussed formally in the framework of the Abstraction-Display-Controller (ADC) interactor model. The ADC model has been defined as a LOTOS specification template tailored for user interface specification. A transformation of the behaviour expressions describing interactor compositions is defined. The transformation, termed synthesis, maintains the generic structure of the ADC interactor while preserving the meaning of the specification. The correctness of the transformation is proved for several types of compositions. The notion of abstract views of interactors is introduced; it is shown how abstract views may themselves be used as components in composition expressions. Example specification extracts are taken from a case study which was developed to test and improve the ADC interactor model

1 Introduction

Research into the formal specification of interactive systems is confronted with the problem of finding appropriate abstractions for their representation. Such abstractions stem from an understanding of interface software architectures and principles for user interface design. Proposed models need to be tested to establish their validity and to demonstrate that they can be scaled up.

This paper attempts to address these questions with reference to the ADC interactor model [1], which is based on the interactors of Paternó [2]. Interactors are primitive abstractions used in modelling interactive software. They are conceptually akin to objects in object oriented programming or to agents of multi-agent user interface architectures e.g. PAC[3]. Most interactors model a local state component along with some representation of its display and, explicitly or implicitly, encapsulate a description of a dialogue local to the modelled object.

A case study in the specification of an existing application, was undertaken to assess the utility and feasibility of modelling interface software using ADC interactors. Extracts from this example presented below provide the context for the discussion of the compositionality of the ADC model. Compositionality refers to the ability to treat compositions of elementary instantiations of the model in the same way as their components. This is an important requirement for scaling up the use of a model and is unique to this formulation of interactors.

The ADC model was described in [1] as a process definition template in the formal specification language LOTOS. Some features of the LOTOS language are summarised in the appendix. A proper introduction to LOTOS can be found in [4]. A brief informal overview of the ADC model is presented in section 2 but the full exposition of the ADC model is not repeated for economy of space. Some aspects of the model are formally defined in section 3. Brief extracts from the case study motivate the discussion on the practical implications of the compositionality of the model, in section 4. Compositionality is discussed more formally, in section 5, in terms of a syntactic transformation of LOTOS behaviour expressions that compose specifications of ADC interactors. This transformation amounts to a synthesis of an ADC interactor specification from a composition expression by a reshuffling of its components. In section 6, the transformation is proven to preserve the observable behaviour specified. Section 7 discusses the specification of abstract views of interactors using the LOTOS operator *hide*. Finally, section 8 links this paper to its research context and to related research.

2 An Informal Overview of the ADC Model

An ADC interactor encapsulates two state components: the abstraction and the display. Its dynamic behaviour is described by its third component the controller unit. The initials ADC refer to these components. Interactors can interact with each other, and possibly with the user or the application. Interactions take place over its gates, have no duration but may be structured to model data communication into or out of the interactor. The interface can therefore be thought of as a graph, whose nodes are the interactors and whose edges correspond to connections between them over a pair of gates or are notionally connected to agents representing the application or the user. The display models the appearance of an interactor on the screen. The display state is conceived to be output directly to the screen or to some other interactor which will process and output this information. The abstraction holds state information resulting from the interpretation of input already received by the interactor.

Figure 1: The internal structure of an interactor.

This in turn, may be interpreted by the interactor to provide input to the application or to other interactors it is connected to.

Communication of information between interactors is directed. In describing an interactor the abstraction and the display sides are distinguished. Interactors are represented in diagrams as barrel shaped nodes, with the top or bottom arch dedicated to connections to the abstraction or display side respectively (figure 1). The vertical sides are dedicated to controller connections.

Input received from the top or the bottom sides may update the abstraction and display state. When both the display state and the abstraction state are needed to interpret received input, this is considered to be arriving at the display side. Intuitively, this corresponds to user input that the interactor needs to de-reference with respect to the current display. In practice this is the only difference between input arriving to one side or the other. Choosing which side an interactor gate belongs to does not depend on what it connects to but on how the information received should be interpreted. Note that the interpretation of input based on the display contents characterises graphical interaction.

An important characteristic of the abstraction and the display components is that they are 'neutral' with respect to the dialogue structure of the interaction. The abstraction-display unit (ADU) that encapsulates them is always ready to receive input and send output on all its gates. Dialogue structure is modelled in the Controller Unit (CU). The CU is a description of the allowed sequences of actions and of how the interactor starts, restarts, suspends, resumes or aborts. It is possible that an interactor has a controller component only. The purpose of such a component will be to coordinate the operations of other interactors, and to act as a 'glue' for specification components. In [5] some special case 'glue' components employed by this model are described.

The description so far has been independent of a particular specification notation. Indeed the model is meant to convey general ideas about the structure of an interface specification. It advocates the separation of dialogue and data handling into distinct components of the interactor and is intended to support a high level description, at least partly represented in a visual form that could be used by interface designers to construct architectural models of an interface. Current work has identified reusable sub-components of these interactors. This paves the way for tool support for the partly automated generation of formal specifications from more economical visual representations and libraries of specification templates. From the discussion so far it is clear that the formal specification language which would support such a scheme needs to provide both for the specification of dialogue and data handling. This work has been developed using the LOTOS specification language but it is hoped that the concepts above could be implemented using other languages with similar capabilities.

3 A Formal Model of ADC Interactors

ADC interactors are modelled as LOTOS processes with some standard behaviours and syntax. In [1] a generic interactor component was defined as a LOTOS process and its use was exemplified with the specification of a scrolling list. Below, a less detailed but more rigorous approach is taken in defining an ADC interactor. The specification of the data type that models the display and abstraction components is omitted as it is not relevant in formulating the compositions of interactor specifications that follows.

3.1 Topology of Interactor Gates

The set of gates of the interactor is partitioned into the set of input-output gates G_{io} and the control gates G_c .

$$G = G_c \cup G_{io} \text{ and } G_c \cap G_{io} = \emptyset$$

Controller gates may either effect standardised control behaviours or dialogue events carrying no data that are handled strictly by the controller component.

$$G_c = G_{control} \cup G_{dialogue} \text{ and } G_{control} \cap G_{dialogue} = \emptyset$$

$$G_{control} = G_{start} \cup G_{suspend} \cup G_{resume} \cup G_{restart} \cup G_{abort}$$

The set of input and output gates of the interactor is also partitioned to G_a and G_d which are the gates on the abstraction and display sides of the interactor. This and their further distinction to input and output gates are not important for the discussion that follows.

3.2 The Syntactic Structure of ADC Interactors

An ADC interactor is a non-terminating process with functionality `no-exit`, which in general is formed by the parallel composition of two processes, the ADU and the CU.

```
process ADC[Gc ∪ Gio](State): noexit :=
  ADU[Gio](State) || [Gio] || CU[Gc ∪ Gio]
endproc
```

where *State* stands for the state parameters of the ADU. In the general case this corresponds to the current and future display and the abstraction state.

CUs are themselves defined in a fixed structure that supports a set of standard dialogue behaviours. These are instantiated in practice by correctly connecting the corresponding gates of the interactor. Interactor-specific dialogue is encoded in a sub-process of the controller unit called *constraints*. In following sections, where the synthesis of interactors is described as a syntactic transformation, the internal structure of the ADU and the CU components are not considered.

3.3 Behaviour of the ADU

The ADU component supports input and output without applying any constraints on the dialogue of the interactor. It is a recursive, non terminating process that always offers events on all its gates. Input gates always accept data of the appropriate sort and output gates always offer a non-empty choice of values. In general, it is required that events on the input gates of an ADU are specified by variable declarations which may not be associated with a selection predicate.

For a process expression to be an ADU, it is necessary and sufficient that:

1. event specifications have no selection predicates
2. $P(ADU) \approx Q$, where $Q = \text{choice } g \text{ in } [G_{io}] [] g; Q$

$\approx$ denotes weak observational equivalence. P is the naive transformation (cf. [6, pp 136]) from full LOTOS to basic LOTOS that simply ignores data values of events. The naive transformation P maps each event $g\langle v \rangle$ specified by its operand to an event specified only by the gate identifier g . It supports the concept of ϕ -bisimulation equivalence (see appendix) between its input and its output behaviour expressions.

Lemma 1. Given two abstraction display units ADU^A and ADU^B , their parallel composition over a set of gates $G \subseteq G_{io}^A \cup G_{io}^B$ is also an ADU. The gate sets refer to the identifiers used in the instantiation of the processes.

Proof. Condition 1 above holds trivially since no selection predicates are introduced. By the distribution of the naive transformation over the parallel composition operator, if condition 2 holds for the two composed units, then it will hold also for their synchronous composition, where process Q is defined as

$$Q = \text{choice } g \text{ in } [G_{io}^A \cup G_{io}^B] [] g; Q$$

3.4 Standard Control Behaviours

The standard built-in control behaviour of an ADC interactor is:

1. it starts with an event from the set G_{start} .
2. its operation suspends with an event from the set $G_{suspend}$ and resumes where it stopped with an event from the set G_{resume} .
3. it terminates with an abort event from the set G_{abort} .
4. at any moment its dialogue may return to its initial state - the state following a start event- with an event from the set $G_{restart}$.

Figure 2: The composition of three interactors of Simple Player™.

In [1,2] these standard control behaviours are supported by an intensional description of the CU. For the definition of the synthesis transformation it will not be required that the CU have any particular structure. It is interesting though to examine how such behaviours may be preserved by the synthesis transformation.

4 Compositionality in the Context of an Example Specification

The ADC model has been tested by developing a specification of the graphical interface of the Simple Player™ application program for the Macintosh computer. This provides access to play-back and editing facilities for QuickTime™ movies. The specification was elaborated to an abstraction level much lower than would be normally intended in a formal specification of an interface design. This provides added confidence in the expressive power of the model: the comparison with the observed behaviour is straight forward, and the contingencies of the represented interactions are not brushed over or abstracted away from. A full exposition of the case study [5] and its results is not in the scope of this paper.

Each of the visible components of the Simple Player™ interface was modelled as an ADC interactor. The interface was modelled as a composition graph of these interactors, plus some auxiliary interactors used as connectives. A small segment of this graph is shown in figure 2. This includes the three objects indicated on the snapshot of Simple Player™ in operation, represented as barrel shaped nodes. The half-ellipses are abstraction only interactors used as connectives that communicate data to two interactors asynchronously.

The top side of the player bar interactor is connected to the functional core, not shown in figure 2. This core sends to the player bar the time coordinate of the currently shown frame of the movie played. The player bar interactor outputs its display state through gate *outPLR*. There are two recipients for this data. The thumb interactor receives the time information at gate *iaTHMB* through the connective shown as an ellipse. The thumb interactor uses this information to update its display, offered as a display state value over gate *outTHMB*. This provides feedback to the user, who interacts with the display side of the leaf interactor nodes. Similar information can flow in the opposite direction. Mouse events are input to the thumb from the display side, and communicated to the other two interactors from gate *oaTHMB*, once more with the help of a connective. The structure shown in figure 2 is not suggested to be the structure of the actual software which is defined as a layered set of function libraries. Interactors are proposed as useful abstractions for structuring a formal specification and the particular composition above was found helpful for this purpose in the case study.

It is clear, that the three interactors and the two connectives are very closely related, in fact, they can be thought of as a single interactor. This would model the behaviour of a movie slider that combines the functions of its components, i.e. it allows selections to be made, and movie positions to be input or output. In the context of the graph representing the global interface architecture, the graph segment of figure 2, could be replaced by a single barrel node. In this process of synthesis a set of interactors is grouped and is substituted by a single interactor that models the same observable behaviour. Some specification extracts modelling the interactive components of figure 2 show the behaviour expression corresponding to the structure of figure 2 (the distributed form), and then it is re-written as a single composite interactor (the compound form).

4.1 Distributed Form (DF)

The DF is the synchronous composition of the *playerBar*, *thumb* and *selectionBand* interactors, plus the two connectives *dm1* and *dm2*. For economy of presentation, where the gate set is obvious it is substituted by ‘...’ and the following shorthand is used:

$$G_c = \{\text{start, restart, suspend, resume, abort}\}$$

The LOTOS behaviour expression defining the DF is:

```
((playerBar[Gc, inpPLR, oaSB, outPLR, setMovieBox, data, gotoTime, setSelection]
|[Gc, outPLR]|
dm1[Gc, outPLR, iaTHMB, iaSB])
|[Gc, iaTHMB, inpPLR]|
(thumb[Gc, press, move, release, outTHMB, iaTHMB, oaTHMB]
|[Gc, oaTHMB]|
dm2[Gc, oaTHMB, inpPLR, inpSB]))
|[Gc, inpSB, oaSB, iaSB]|
selectionBand[Gc, inpSB, outSB, iaSB, oaSB, erase, enable, send]
```

For economy of space, it is only possible to show here a bit of the internal structure of one interactor. The interested reader is referred to [5], for a full exposition of the specification. Consider for example the *selectionBand* interactor. It is itself the composition of an ADU and a CU as follows:

```
process selectionBand[Gc, inpSB, outSB, iaSB, oaSB, erase, enable, send]:noexit:=
adu_SB[...] (noSelection, thePlayerBar, thePlayerBar)
|[inpSB, out, ia, oa, erase]|
cu_SB [Gc, inpSB, outSB, iaSB, oaSB, erase, enable, send]
endproc
```

Process *adu_SB* is initialised with an abstraction value *noSelection*, (defined in the data typing component of the specification), and an initial display state *thePlayerBar*, which is presumed to be the initial presentation of the player bar (with no selection band showing).

```
process adu_SB[...] (a:line, pc,ps: playBar):noexit :=
oaSB!a;          adu_SB[...] (a,pc,ps) []
outSB!pc;        adu_SB[...] (a,pc,pc) []
iaSB?x:playBar;  adu_SB[...] (receivePB(a,x),renderPB(pc,x),ps) []
inpSB?x:pnt;     adu_SB[...] (inputPnt(x,a),echo(x,ps,a),ps) []
erase;          adu_SB[...] (inputEr(a),echoEr(pc, a),ps)
endproc
```

Process *cu_SB* supports the standard control behaviours plus the particular to *selectionBand* dialogue constraints. For brevity only the process *constraints* that describes them is shown.

```
process constraints[inpSB, outSB, iaSB, oaSB, erase, enable, send]:noexit:=
(enable; (operation[inpSB, outSB, iaSB, oaSB, erase, send]
[> send; oaSB?x:line; constraints[...]))
[]      iaSB?x:playBar;constraints[...]
[]      outSB?x:playBar; constraints[...]
[]      erase; outSB?x:playBar; oaSB?x:line; constraints[...]
where
  process operation[inpSB, outSB, iaSB, oaSB, erase, send] : noexit :=
    inpSB?x:pnt; outSB?x:playBar;      operation[...]
    []      outSB?x:playBar;           operation[...]
  endproc
endproc
```

4.2 Compound Form (CF)

Process *compound*, the CF, has the general ADC structure:

```
process compound[Gc, Gio]:noexit :=
  adu[Gio] |[Gio]| cu[Gc ∪ Gio ∪ {enable, send}]
endproc

where
```

$G_{io} = \{inpPLR, outPLR, setMovieBox, data, gotoTime, setSelection, press, move, release, outTHMB, iaTHMB, oaTHMB, inpSB, outSB, iaSB, oaSB, erase\}$

The ADU and the CU of the compound form are defined as follows:

```
process adu[Gio]: noexit :=
  ((adu_playerBar[inpPLR, oaSB, outPLR, setMovieBox, data, gotoTime,
    setSelection] (initPlayBarData, thePlayerBar, thePlayerBar)
    |[outPLR]|
    adu_dm1[outPLR, iaTHMB, iaSB](thePlayerBar))
    |[iaTHMB, inpPLR]|
    (adu_thumb[press, move, release, outTHMB, iaTHMB, oaTHMB](indicator, thumb, thumb)
    |[oaTHMB]|
    adu_dm2[oaTHMB, inpPLR, inpSB](aPoint)))
    |[inpSB, oaSB, iaSB]|
    adu_SB [inpSB, outSB, iaSB, oaSB, erase] (noSelection, thePlayerBar, thePlayerBar)
endproc

process cu[Gc, Gio ∪ {enable, send}]: exit :=
  ((cu_playerBar[Gc, inpPLR, oaSB, outPLR, setMovieBox, data, gotoTime,
    setSelection]
    |[Gc, outPLR]|
    cu_dm1[Gc, outPLR, iaTHMB, iaSB])
    |[Gc, iaTHMB, inpPLR]|
    (cu_thumb[Gc, press, move, release, outTHMB, iaTHMB, oaTHMB]
    |[Gc, oaTHMB]|
    cu_dm2[Gc, oaTHMB, inpPLR, inpSB]))
    |[Gc, inpSB, oaSB, iaSB]|
    cu_SB[Gc, inpSB, outSB, iaSB, oaSB, erase, enable, send]
endproc
```

This conceptually simple idea, of grouping and reshuffling of interactor components, has several implications for the modelling scheme. First it raises the question of what it means to group interactors or for a group and a single higher level interactor to be the same etc. It also has practical repercussions. The synthesis of interactors by such groupings enables the bottom-up construction of interactors from library components, or conversely the top down refinement of abstract specifications to structures closer to the implementation architecture. Supporting this concept of composition, by appropriate abstractions, notations and tools should get us a step closer to supporting the use of formal methods in the interface design process.

From an interface design perspective, compositions transform an implementation view of the design as a composition of individual interactors, to a design oriented view which separates a centralised dialogue description from data handling for the global interface specification. This is easier to inspect and easier to manipulate in order to design interaction and induces a rational separation of concerns for the designer. In terms of LOTOS specifications it amounts to a transition from a resource oriented specification to a constraint oriented specification. In contrast, the conventional specification practice involves the transformation from a constraint oriented view to a resource oriented view [7]. This unorthodox transition between specification styles reflects a view that the application of formal methods to the design, specification and verification of user interfaces, could benefit by aligning with user interface design activity. This has been argued to be ‘opportunistic’ [8] requiring the designer to work both bottom-up and top-down.

5 Compositionality of the ADC Model

When building a specification of an interface we may want to compose ADC interactors, specified as LOTOS processes, using any LOTOS process algebra operator that combines two or more processes. What we mean by compositionality is that it should be possible to re-write this expression into a single compound ADC interactor, i.e. to a single LOTOS process that satisfies the conditions described in the previous section. It is required that the rewritten form is weakly observationally congruent (see appendix) to the original specification. This rewriting transformation is termed *synthesis* of interactors in the remaining of this paper.

The synthesis transformation is described by:

1. the *distributed form* $DF = ADC^A \otimes ADC^B$ where $\otimes$ is a binary composition expression for non-terminating LOTOS behaviour expressions (one of $\|$, $[[G]]$, $\|$, $[>$, $[]$).
2. the *compound form* $CF = ADU^{AB}[G_{io}^{AB}] \| CU^{AB}[G_c^{AB} \cup G_{io}^{AB}]$.
where $ADU^{AB} = ADU^A [[G_s]] ADU^B$ and $CU^{AB} = CU^A \otimes CU^B$

The compound form is sufficiently defined by $\otimes$, G_{io}^{AB} , G_s .

Figure 3: Synthesis applied to the synchronous composition of two interactors

3. the *Transformation Requirement* is that the compound form is itself an ADC interactor. From lemma 1, ADU^{AB} is an abstraction display unit for any G_s . No special requirements are set to the syntax or the behaviours of the CU, so the CF is an ADC interactor as the parallel composition of an ADU and a CU.
4. the *Correctness Preservation Requirement*, describes the invariant of the transformation, i.e. it expresses the desired relationship between the semantics of the distributed form and the compound form. Generally it is required that $DF = CF$ where the equals sign denotes weak observational congruence (the definition of this relationship can be found in the appendix). The proof of the correctness of the transformation is presented below for all applicable process composition operators.

6 Correctness of the Synthesis Transformation

Static composition operators are examined first. Full synchronisation $\parallel$, and pure interleaving $\|$, can be thought of as boundary cases of the (partial) synchronisation of behaviour expressions, so only the latter need be examined. Consider the synchronous composition of two interactors, with respective gate sets G^A and G^B over a synchronisation gate set $G \subseteq G^A \cap G^B$. For a gate $g \in G$, several types of connections are distinguished, depending on what type of gate it is for the connected interactors. A composition of two interactors may use none (in the case of pure interleaving) or many connections of types (a) - (g) below.

- a) Data is sent from the abstraction side of interactor A to the display side of B. The effect of interactor A is to raise the abstraction level of the input B receives.
- b) Data is sent from the display side of B to the abstraction side of A. Interactor A processes the display state of interactor B e.g. as part of a graphics pipeline.
- c) Data is sent from the abstraction side of A to the abstraction side of B. Similar to case (a), but here the interpretation of the received data is independent of the display of B.
- d) Data is sent from the display side of B to the display side of A. This case may be ruled out as non-intuitive. The need to de-reference input with respect to the display is a concept introduced to model the interpretation of graphical input so it is not clear how this type of connection could be used.
- e) Synchronisation of control gates. This can be thought of as one interactor controlling another, by triggering, or by constraining dialogue elements of the other. However, since a LOTOS action is undirected this case can be thought as a mutual constraint of their dialogue structures.

Figure 4: The range of static compositions of ADC interactors. Arrows indicate data flows while simple lines represent simple synchronisation.

- f) Connection of a control gate of A to the display side of B. An event carrying no data will evoke the interpretation operations that update the abstraction and display states respectively. The event is constrained by the controller component of both interactors.
- g) Connection of a control gate of A to the abstraction side of B. It is similar to the previous case, with the difference that the abstraction of B is updated without reference to its display content.

6.1 Correctness of the Synthesis Transformation for Synchronous Composition

The distributed and the compound forms of the composition are:

$$\begin{aligned} DF &= \left(ADU_A \mid [G_{io}^A] \mid CU_A \right) \mid [G] \mid \left(ADU_B \mid [G_{io}^B] \mid CU_B \right) \\ CF &= \left(ADU_A \mid [G_1] \mid ADU_B \right) \mid [G_{io}^A \cup G_{io}^B] \mid \left(CU_A \mid [G_2] \mid CU_B \right) \end{aligned}$$

Strong observational equivalence ($\sim$) is proved below, that entails the required congruence relation. The proof results in constraints for the gate sets G_1 and G_2 so that $DF \sim CF$.

Proof.

The proof technique used here is adapted from [7] and is a shorthand version of the proof by bisimulation.

Strong observational equivalence is defined as follows:

$DF \sim CF$ iff $(DF, CF) \in R$ and R is a bisimulation relation.

By definition the following must hold for R to be a bisimulation relation for all events $g \langle v \rangle$ for which there is a transition for DF or CF .

$$\begin{aligned} DF &\xrightarrow{g \langle v \rangle} DF' \Rightarrow \exists CF' \mid (DF', CF') \in R \bullet CF \xrightarrow{g \langle v \rangle} CF' \\ CF &\xrightarrow{g \langle v \rangle} CF' \Rightarrow \exists DF' \mid (DF', CF') \in R \bullet DF \xrightarrow{g \langle v \rangle} DF' \end{aligned} \quad (T)$$

The proof consists in proposing a relation R that can play the role of a bisimulation relation and finding the conditions that result for the component processes if (T) should hold.

Let $R = \{ \langle A \mid [S_1] \mid B \mid [S_2] \mid C \mid [S_3] \mid D, A \mid [S_4] \mid C \mid [S_5] \mid B \mid [S_6] \mid D \rangle \}$

where A-D are behaviour expressions and S_1 - S_6 are gate sets. Clearly, $(DF, CF) \in R$ and by the definition of the semantics of the synchronisation operator, any transition of DF or CF , will involve the transition of at least one of its components, and will always result in the *same* static structure for DF' and CF' , with only the component processes that perform the transition changing their state. Thus, $(DF', CF') \in R$.

Condition (T), requires that DF and CF have the same sets of transitions. A transition $g \langle v \rangle$ is defined by gate identifier g and value $\langle v \rangle$. The gate identifier g must belong to the label set of the component process or processes of each of the two behaviour expressions compared that participate in the transition. The gate sets do not grow with the recursive instantiation of the processes. Transitions are categorised by the component processes involved. Each category is described by a combination of transitions for the component processes. The domain of g for each type of transition is the intersection of the label sets of the component processes, constrained by the behaviour expression DF or CF so that the transition is possible. An empty gate set means that the transition is impossible. By equating the domains of g for DF and CF , for each type of transition, conditions result between the label sets of the two expressions.

To consider all possible transitions for DF or CF , all possible combinations of transitions of their components are examined (table 1). Note that some combinations are impossible by the definition of the model, as for example an ADU firing without synchronising with the CU component that controls it. The rightmost column presents the condition resulting by equating the domain of g for the DF and the CF form, for each possible transition $g \langle v \rangle$.

The combination of the resulting conditions allows many possible constructions for G_1 and G_2 . An important condition that results from their combination is that

$$G_{io}^A \cap G_{io}^B \subseteq G, G_1, G_2 \quad \wedge \quad G_1 \cap ((G_{io}^A \cap G_c^B) \cup (G_{io}^B \cap G_c^A)) = \emptyset$$

Condition CPR below is stronger than the conditions above, but it allows an economical construction of the compound form.

$$G_1 = G_{io}^A \cap G_{io}^B \cap G - ((G_{io}^A \cap G_c^B) \cup (G_{io}^B \cap G_c^A)) \quad \wedge \quad G_2 = G \quad (CPR)$$

Thus, the controllers in the CF are composed over exactly the synchronisation gate set used in the DF , while the ADUs can not synchronise over the gates implementing connections of types (f) and (g) of figure 4.

The discussion so far has ignored the data values passed. It is argued that they do not discredit the proof presented. Without loss of generality it is assumed that A produces data values and B consumes them. By the definition of the model the gates of the ADU are typed. Let $ValueSet(t)$ denote the possibly infinite set of values of data type t , and $range_A(g) \subseteq ValueSet(t)$ (res. $range_B(g)$) denote the set of values possibly offered by ADU_A

ADU _A	CU _A	ADU _B	CU _B	Condition
•				impossible for both DF and CF
	•			$G_C^A \cap G = G_C^A \cap (G_2 \cup G_{io}^B)$
•	•			$G_{io}^A \cap G_s = G_{io}^A \cap (G_1 \cup G_2)$
		•		impossible for both DF and CF
•		•		impossible for both DF and CF
	•	•		$(G_{io}^B - G_1) \cap ((G_C^A \cup G_{io}^A) - G_2) = \emptyset$
•	•	•		No resulting condition ($\emptyset = \emptyset$)
			•	$G_C^B \cap G = G_C^B \cap (G_2 \cup G_{io}^A)$
•			•	$(G_{io}^A - G_1) \cap ((G_C^B \cup G_{io}^B) - G_2) = \emptyset$
	•		•	$G_C^A \cap G_C^B \cap G = G_C^A \cap G_C^B \cap G_2$
•	•		•	$G_{io}^A \cap G_C^B \cap G = (G_2 - G_1) \cap ((G_{io}^A \cap G_C^B) \cup (G_{io}^A \cap G_{io}^B))$
		•	•	$G_{io}^B \cap G = G_{io}^B \cap (G_1 \cup G_2)$
•		•	•	No resulting condition ($\emptyset = \emptyset$).
	•	•	•	$G_{io}^B \cap G_C^A \cap G = (G_2 - G_1) \cap ((G_{io}^B \cap G_C^A) \cup (G_{io}^B \cap G_{io}^A))$
•	•	•	•	$G_{io}^A \cap G_{io}^B \cap G = G_{io}^A \cap G_{io}^B \cap G_1 \cap G_2$

Table 1: Each row shows a combination of transitions and the condition put on the label sets of the components and G , G_1 , G_2 so that the transition be possible for both DF and CF.

(res. read by ADU_B) over gate g . As predicate selection was disallowed in the definition of the interactors it follows that: $\text{range}_B(g) = \text{ValueSet}(t)$. Also, CU_A and CU_B by their definition allow all values $v \in \text{ValueSet}(t)$. Data communication takes place only in the transitions described by the last row of the table. It is easy to see by inspection of the DF and the CF that in both cases the set of possible values is $\text{range}_A(g)$. Thus, for any gate label g , DF and CF offer the same transitions $g < v >$, with $< v > \in \text{range}_A(g)$.

The case where the two interactors are composed with the pure interleaving operator in the distributed form, follows directly by substituting $G = \emptyset$ into the CPR:

$$(\text{ADU}_A \parallel [G_{io}^A] \parallel \text{CU}_A) \parallel (\text{ADU}_B \parallel [G_{io}^B] \parallel \text{CU}_B) = (\text{ADU}_A \parallel \text{ADU}_B) \parallel [G_{io}^A \cup G_{io}^B] \parallel (\text{CU}_A \parallel \text{CU}_B)$$

6.2 Synchronous Composition and the Standard Control Behaviours

In the example of section 4, and in the whole graphical interface modelled for the case study, the composed interactors come to life, suspend, resume and abort simultaneously. This is true both in the interface and in the formal model constructed. In general this is the case for a group of interactors controlling closely related aspects of application functionality. However, if the interactive application requires navigation through some screens, or distinguishing modes as with a dialogue box, or the dynamic creation and deletion of interactive objects as in drawing packages, then the commonality in such dialogue elements ceases to hold. Depending on the interactive behaviour modelled the gate sets corresponding to standard control behaviours have to be defined for the compound form. Below, the weakest requirements for these gate sets are proposed, for these logical behaviours to be preserved in the CF, for synchronous composition. The static configurations supported are constrained so that the two interactors synchronise at least on the following gate sets:

$$G_x^{AB} \subseteq G_x^A \cap G_x^B \text{ for all } x \in \{\text{start, suspend, resume, restart, abort}\}$$

So for the gate set G over which two interactors synchronise it is required that

$$G_{\text{start}}^{AB} \cup G_{\text{suspend}}^{AB} \cup G_{\text{resume}}^{AB} \cup G_{\text{restart}}^{AB} \cup G_{\text{abort}}^{AB} \subseteq G$$

6.3 Correctness of the Composition with $[]$ (Choice)

The proof for choice and disable is much simpler, although given the dynamic nature of the operators, relation R has to be extended.

$$DF = (ADU_A \parallel [G_{io}^A] \parallel CU_A) [] (ADU_B \parallel [G_{io}^B] \parallel CU_B)$$

$$CF = (ADU_A \parallel ADU_B) \parallel [G_{io}^A \cup G_{io}^B] \parallel (CU_A \parallel CU_B)$$

The bisimulation relation is defined to be

$$R = \{ \langle A \parallel [S_1] \parallel B \parallel [C \parallel [S_2] \parallel D], A \parallel [C \parallel [S_3] \parallel (B \parallel D)] \rangle, \langle A \parallel [S_1] \parallel B, (A \parallel [C \parallel [S_2] \parallel B]) \rangle \}$$

With similar investigation of all possible transitions for DF and CF the following condition results:

$$G_c^A \cap G_{io}^B = \emptyset \wedge G_c^B \cap G_{io}^A = \emptyset \wedge G_{io}^A \cap G_{io}^B = \emptyset$$

6.4 Correctness of the Composition with $[>]$ (Disable)

The DF and CF have the following form:

$$DF = (ADU_A \parallel [G_{io}^A] \parallel CU_A) [>] (ADU_B \parallel [G_{io}^B] \parallel CU_B)$$

$$CF = (ADU_A \parallel ADU_B) \parallel [G_{io}^A \cup G_{io}^B] \parallel (CU_A [>] CU_B)$$

The bisimulation relation is defined to be

$$R = \{ \langle A \parallel [S_1] \parallel B [>] C \parallel [S_2] \parallel D, A \parallel [C \parallel [S_3] \parallel (B [>] D)] \rangle, \langle A \parallel [S_1] \parallel B, (A \parallel [C \parallel [S_2] \parallel B]) \rangle \}$$

The resulting condition on the labels of the gate sets is:

$$G_c^A \cap G_{io}^B = \emptyset \wedge G_c^B \cap G_{io}^A = \emptyset \wedge G_{io}^A \cap G_{io}^B = \emptyset$$

The disable and choice operators, can easily be recognised as useful structure in dialogue design, but the implications of their use to combine ADC interactor specifications have not yet been explored. A similar discussion about the standard control behaviours as for the case of the synchronous composition is straight forward, but reveals some of the difficulties of applying them in practice. This shows the necessity of deriving practical intuitions of their use through further example specifications.

7 Composition and Hiding

Hiding may be applied to an interactor, or to a group of interactors to abstract from internal detail. In the discussion that follows, the term *abstract view* refers to an interactor some of whose gates are hidden. Abstract views are illustrated as interactors enclosed in black frames as in figure 5. In general when a group of interactors is composed, it will be required to observe the behaviour of the whole without knowledge of its internal organisation. To result in an abstract view of the specification of the interface it is necessary to be able to compose such abstract views as if they were interactors. A stronger requirement is to be able to transform the abstract view into an equivalent interactor expression. These two possibilities are examined below.

(a) It would be convenient if applying the hide operator to an interactor specification would result in an interactor structure with the hiding distributed among its sub-components. Unfortunately hiding of a synchronisation gate does not preserve the observable behaviour. If though the data values associated with events are ignored, as for example by applying the naive transformation mentioned in section 3, then the behaviour modelled by the distribution of hide to the components of the ADC is weakly observationally equivalent to the original expression. More formally it can be said that the resulting interactor structure is Φ -bisimilar to the original one, for a coding function ϕ , that maps an event specification to one that is defined only by gate identifier and data type (not the value of the data).

$$\text{hide } g \text{ in } (ADU \parallel [G_{io}] \parallel CU) \sim_{\phi} (\text{hide } g \text{ in } ADU \parallel [G_{io}-g] \parallel \text{hide } g \text{ in } CU)$$

and following the discussion on the nature of the ADU it can easily be seen that

$$\text{hide } g \text{ in } (ADU \parallel [G_{io}] \parallel CU) \sim_{\phi} \text{hide } g \text{ in } CU$$

The definition of Φ -bisimulation is included in the appendix. It formalises the intuition behind transformations from full LOTOS to basic LOTOS, as for example the naive transformation used in section 3. Intuitively, the above expression means that if data values are ignored, then the dialogue structure of the abstract view is sufficiently observed by experimenting with the abstract view of the control unit only.

Figure 5: Transforming an abstract view to a Φ -bisimilar ADC interactor.

Figure 6: Composition of an abstract view with an ADC interactor

(b) an abstract view of an interactor with some of its gates hidden can be composed with other interactors using a composition operator $\otimes$. Consider the case of synchronous composition over a possibly empty set of gate labels G ; expressions for $[\cdot]$, $[\cdot]_>$, $[\cdot]_{||}$ are very similar. The result will be an abstract view of the composition of the two interactors.

$$\begin{aligned} & \forall g \in (G_{io}^A \cup G_c^A) \mid g \notin (G_{io}^B \cup G_c^B) \cup G \bullet \\ & (\text{hide } g \text{ in } (ADU_A \mid [G_{io}^A] \mid CU_A)) \mid [G] \mid (ADU_B \mid [G_{io}^B] \mid CU_B) \sim \\ & \text{hide } g \text{ in } ((ADU_A \mid [G_1] \mid ADU_B) \mid [G_{io}^A \cup G_{io}^B] \mid (CU_A \mid [G] \mid CU_B)) \end{aligned}$$

This is a straight forward but important extension to the composition transformation described in the previous section. Composition extends to abstract views of interactors defined.

The cumulative effect of applying the transformation (b) is that an interface specification written as the composition of ADC interactors and/or abstract views of ADC interactors, which themselves are compositions of lower level interactors, can be rewritten as an abstract view of a composition of interactors, which is an expression of the form:

$$\text{hide } [\dots \text{all hidden gates} \dots] (ADU_{\text{top level}} \mid [G_{\text{top level}}] \mid CU_{\text{top level}})$$

8 Discussion

The ADC interactor model was introduced in the context of current research in interactor models in [1], where it was hypothesised that interface specifications could be constructed by the composition of ADC interactor specifications. This hypothesis was tested with a reverse engineering case study reported in [5]. While in principle it is possible to use the ADC interactor to model all architectural elements of the interface, practice showed that much was to be gained in simplicity and conciseness by supporting special case interactors. These included abstraction-only interactors, display-only interactors, and controller-only interactors. Some abstraction-only interactors have been standardised, playing the role of a ‘glue’ between interactive components, as for example the special purpose connectives of figure 2.

The idea of composing interactors has been further developed. This paper focused on the use of two LOTOS operators: synchronisation and hiding. Synchronisation is useful in forming static compositions of interactors, which intuitively corresponds to interactors appearing together on the screen as one object. It may model data communication between interactors and mutual constraints on each others’ dialogue. The synthesis transformation of behaviour expressions composing ADC interactor specifications was formally defined and it was proved that it preserves strong observational equivalence.

Hiding is important in scaling-up the use of interactors, in that it provides a more abstract view of a given interactor. Two types of transformation were discussed. The first consists in moving the hide operator inside the interactor specification. It was discussed how this transformation does not preserve the observable behaviour, but may be useful in the observation of the dialogue of an abstract view of an interactor independently of its data handling component (the ADU). The second demonstrates that abstract views of interactors can be composed with interactors, provided gate identifiers do not collide, resulting in an abstract view of the composition. This is very important as it enables abstract views to be used as design constructs.

The synthesis transformation discussed does not apply generally to LOTOS specifications; the proof of its correctness was contingent upon the specific form of the ADC model. Regarding the synchronous composition of interactors, synthesis is effectively a re-shuffling of the operands and synchronisation gates in a parallel composition of processes. A similar re-shuffling was introduced as a congruence law for Basic LOTOS processes in [7], but the conditions it requires do not hold for the ADC model which uses full LOTOS anyway. The reshuffling of a synchronous composition can be seen as a special case of the graphical composition theorem with hiding (GCTH) of [9]. The maximal cooperation condition that is required by the GCTH is stronger than the conditions of table 1, although it is not too constraining in practice. However the proof presented in section 6 is interesting as it extends to the disable and choice operators.

Relevant work reported by Faconti et.al. in [10] is concerned with the synchronous composition of logical input devices for graphics systems to form hierarchies (the approach could easily apply to the interactor

model of [2]). All possible textual interpretations of a diagrammatic representation of a LOTOS specification are established to be equivalent on the basis of the GCTH of [9]. This validates the graphical representation as a more accurate and precise model of the specification and motivates the support of a visual editor. In the terms of the previous sections the equivalence discussed in [10] is between a set of distributed forms. In contrast, the approach presented here focuses on the synthesis of a compound form that maintains the general ADC structure.

Interactors are a direct analogue of multi-agent conceptual software architectures like PAC [3] and MVC [11] in the context of the formal specification of user interfaces. Duke et.al. [12] make a comparison between formal interactor models and such informal conceptual architectures. The comparison reveals a strong similarity in structure and purpose of the components. However, contrary to such informal architectures, formal interactor models have to be associated with precisely defined semantics of their compositions. The research reported is a contribution in this direction. The synthesis transformation allows the general structure of the interactor to be maintained throughout the composition of lower level entities. Thus the higher level composition will have the general ADC interactor structure. The consistent structure that is preserved through the composition of interactors should enhance the comprehensibility of the specifications. Further it enables the use of generic expressions of properties of interfaces capitalising on the standardised syntactic structure of the interactors as was suggested in [1]. Such analytical use of interactor specifications is reported by Paternó in [13].

The practical impact of the synthesis transformation in the design life-cycle of interactive systems has not yet been investigated. At a first glance it seems to support the bottom-up construction of user interface specifications. It is not the intention of this research and it should not be the intention of a representation technique, formal or not, to prescribe a particular approach to interface design. What is required from such a scheme, is that it provides the appropriate flexibility for its use in realistic settings. Empirical investigations of interface design practice suggest that it is mostly a bottom-up exercise. The synthesis transformation is in line with such an approach. The use of ADC interactor specifications is also consistent, and ought to be, with the standard software engineering approach to the refinement of abstract specifications. Standard techniques for the refinement of LOTOS specifications can be used for this purpose. Refinement is also supported via the decomposition transformation, the dual to the synthesis transformation discussed in this paper. In LOTOS terms the decomposition is a transition from the constraint oriented to the resource oriented specification style [7]. An early report on the nature and the theoretical foundation of the decomposition of ADC interactor specifications can be found in [14].

The definition of a formal framework for the modular development of software by the composition of lower level entities or equally well by the refinement and specialisation of higher level entities is a goal of current research in object oriented formal specification techniques [15]. The ADC interactor model is an object based formalism. This means that it supports encapsulation and information hiding. An object can only be accessed by its environment through well a defined interface, which in this case is its set of gates. Object-orientation proper, requires the introduction of the concept of a class and a mechanism to support inheritance, for the incremental definition of classes. Cussack et.al. [16] have presented a simple set theoretic framework that may guide the object oriented interpretation of a formal specification language and among other languages they have applied it to LOTOS. An object oriented enhancement of the ADC model is one of the aims set for future work.

It has been argued throughout this paper that the synthesis of interactors to form higher level entities has direct relevance in the context of interface design. The work reported here has addressed, at a theoretical level, some of the issues arising in scaling up and in supporting abstract views of interactor specifications. This is on going work and the theoretical and practical implications of the various compositions are still under investigation.

9 References

- [1] Markopoulos P. On the Expression of Interaction Properties within an Interactor Model. In: Palanque P, Bastide R (eds) *Design, Specification, Verification of Interactive Systems '95*, Springer, Wien - New York, 1995, 294-311.
- [2] Paternó F, Faconti G. On the use of LOTOS to describe graphical interaction. In: Monk A, Diaper D, Harrison M (eds), *Proceedings of Human Computer Interaction 1992*, Cambridge University Press, 1992, 155-173 (People and Computers VII).
- [3] Coutaz J. PAC, an Object Oriented Model for Dialog Design. In: Bullinger H J, Shakiel B (eds) *Proceedings INTERACT-'87*, Elsevier, North-Holland, 1987, 431-436.
- [4] Bolognesi T, Brinksma E. Introduction to the ISO specification language Lotos. In: van Eijk P, Vissers C, Diaz M (eds) *The Formal Description Technique Lotos*, Elsevier, North-Holland, 1989, 23-73.
- [5] Markopoulos P. Case study in the formal specification of the SimplePlayer™ graphical interface for playing QuickTime™ movies, using the ADC interactor model. Technical Report 712, Department of Computer Science, Queen Mary and Westfield College, University of London, 1996.

- [6] LOTOSPHERE. Catalogue of LOTOS Correctness Preserving Transformation. In: Bolognesi T (ed) Final Deliverable of the LOTOSPHERE ESPRIT project, Lo/WP1/T1.2/N0045/Vo3, 1991.
- [7] Vissers C A, Scollo G, van Sinderen M, Brinksma E. Specification styles in distributed systems design and verification. *Theoretical Computer Science*, 1991; 89:179-206.
- [8] Guindon R. Designing the Design Process: Exploiting Opportunistic Thoughts. *Human-Computer Interaction*, 1990; 5:305-344.
- [9] Bolognesi T, Frutos-Escrig D & Ortega-Mallen Y. Graphical Composition Theorems for Parallel and Hiding Operators. In: Quemada J, Manas J, Vasquez E (eds) *Formal Description Techniques III*, Elsevier, North-Holland, 1991, 459-470.
- [10] Faconti G, Fornari A, Zani N. Visual Representation of Formal Specification: An Application to Hierarchical Input Devices. In: Paternó F (ed) *Interactive Systems: Design, Specification and Verification*. Springer, 1994, 349-368.
- [11] Krasner G E, Pope S T. A Cookbook For Using the Model-View-Controller User Interface Paradigm in The Smalltalk-80 System. *Journal of Object Oriented Programming*; 1988, 1:3:26-49.
- [12] Duke D, Faconti F, Harrison M, Paternó F. Unifying Views of Interactors. In: *Advanced Visual Interfaces '94 Conference Proceedings*, ACM Press, 1994, 143-152.
- [13] Paternó F. Definition of properties of user interfaces using action based temporal logic. In: *Proceedings of the 5th Conference in Software Engineering and Knowledge Engineering*, 1993, 314-318.
- [14] Markopoulos P, Rowson J, Johnson P. Dialogue modelling in the framework of an interactor model. In: *Pre-conference Proceedings. Design Specification and Verification of Interactive Systems*. Namur, Belgium, 1996.
- [15] Ruid-Delgado A, Pitt D, Smythe C. A review of object oriented approaches in formal methods. *The Computer Journal*, 1995; 38:10: 777-784.
- [16] Cussack E, Rudkin S, Smith C. An object oriented interpretation of LOTOS. In: Vuong S T (ed) *Formal Description Techniques II*, Elsevier, North Holland, 1990, 211-226.

10 Appendix

This appendix is a brief introduction to some of the elements of the LOTOS language. The reader is referred to [4] for a proper tutorial to the language. LOTOS is a hybrid specification language that consists in two component languages: a process algebra for the specification of the temporal ordering of the behaviour of systems and the ACT-ONE abstract data typing language.

A system is described in LOTOS as a set of interacting processes. Processes interact over gates. Sequencing information about such interaction events is found in behaviour expressions. The simplest behaviour expression is just a process instantiation. Two built-in processes are the stop and exit processes. Stop implies that no further events are offered; exit implies successful termination. More complex behaviour expressions are built up by the composition of process instantiations with LOTOS operators.

<action>; B	perform <action> then perform process B (action prefix)
[b] -> B	if the Boolean expression b is true then perform B (guard)
A [] B	perform A or B (choice)
A B	processes A and B are independent of one another, but can be performed simultaneously: they are multi-threaded (interleave)
A >> B	completing the performance of A enables performance of process B (enable)
A [> B	perform A until its termination, unless at some point B has to be performed interrupting A (disable)
A [a,b,...] B	processes A and B communicate synchronously through the gates a, b,... in the brackets.

LOTOS supports the multi-way synchronisation of processes. This means that any number of processes can synchronise over a gate, participating in a single synchronisation on each interaction on this gate. This allows for the incremental specification of constraints on the occurrence of events at a particular gate. This constitutes the constraint oriented style of specification, discussed in [7] and used throughout this paper. In this style each process sets its own constraints upon the sequencing of interactions on a particular gate or set of gates.

A process may specify its participation in an interaction with an action declaration. This may simply consist in the gate identifier if no data communication is involved. It may involve an output of some value e.g. g!v where v is a value. A LOTOS action declaration may sometimes be of the form g?x:t, where x is a name of

a variable and t is a sort identifier indicating the domain of values over which x ranges. This corresponds to a set of possible actions for the behaviour expression. For example, $g?x:\text{integer}$ specifies a set of actions $g\langle v \rangle$ where $\langle v \rangle$ is in the domain of the integers. Suppose processes A and B are composed in parallel over a gate g as in $A[g]B$. If A offers a value over that gate e.g. $g!\text{true}$ and B offers any event $g?x:\text{bool}$, then the value true is passed from A to B .

The semantics of LOTOS operators are defined in terms of labelled transition systems. Such a definition is beyond the scope of this paper, and the reader is referred to [4]. However some definitions of the equivalences used to compare LOTOS behaviour expressions in this paper should be helpful. In the following the notation $s \xRightarrow{t} r$, means that a string of observable actions t , effects a transition from a state s of the LTS to a state r . Actions are qualified as ‘observable’ to differentiate from the silent actions τ (a hidden action) or δ (successful termination). Specifications are compared with respect to various equivalences. Strong equivalence is defined in the main text. Some weaker types of equivalences are defined below:

10.1 Weak Bisimulation Equivalence

Let S be a set of states, and A the set of actions (labels) of the LTS. A relation $R \subseteq S \times S$ is a weak bisimulation relation iff:

- If $(s_1, s_2) \in R$ then for all $t \in (A - \{\tau\})^*$
- If $\exists s_1' \in S \mid s_1 \xRightarrow{t} s_1'$ then $\exists s_2' \in S \mid s_2 \xRightarrow{t} s_2'$ and $(s_1', s_2') \in R$ and
- If $\exists s_2' \in S \mid s_2 \xRightarrow{t} s_2'$ then $\exists s_1' \in S \mid s_1 \xRightarrow{t} s_1'$ and $(s_1', s_2') \in R$

Let Sys_1 and Sys_2 be labelled transition systems, with state sets S_1 and S_2 , and initial states s_{01} and s_{02} . Sys_1 and Sys_2 are weak bisimulation equivalent iff there exists a weak bisimulation relation R such that

$$S = S_1 \cup S_2 \text{ and } (s_{01}, s_{02}) \in R.$$

Two LOTOS behaviour expressions B_1 and B_2 , with all their free value-identifiers contained $\{x_1, \dots, x_n\}$ are weak bisimulation equivalent, if all instances $[E_1/x_1, \dots, E_n/x_n]B_1$ and $[E_1/x_1, \dots, E_n/x_n]B_2$ are weak bisimulation equivalent where $E_1, \dots, E_n$, are closed value expressions of the same sort as $x_1, \dots, x_n$ respectively.

Two LOTOS behaviour expressions B_1 and B_2 are weak bisimulation congruent, written as $B_1 = B_2$, if B_1 may be replaced by B_2 and vice versa in all LOTOS contexts, without changing the meaning, with respect to weak bisimulation equivalence.

10.2 Φ -Bisimulation Equivalence

This definition is adapted from [6, page 136]. Let $\text{Sys}_1 = (S_1, A_1, T_1, s_{01})$ and $\text{Sys}_2 = (S_2, A_2, T_2, s_{02})$ be two labelled transition systems and $\phi: A_1 \rightarrow A_2$ be a coding function from Sys_1 to Sys_2 such that

$$\forall g_1, g_2 \in \text{Gates} \cup \{\tau, \delta\}, \forall v_1, v_2 \in \text{Value}^* \mid g_1 \neq g_2 \Rightarrow \phi(g_1) \neq \phi(g_2) \text{ and } \phi(\tau) = \tau \text{ and } \phi(\delta) = \delta.$$

R_ϕ is called a ϕ -bisimulation relation for Sys_1 and Sys_2 iff ϕ is a bijection and $\forall (s_1, s_2) \in R_\phi$:

$$\text{If } \exists s_1' \in S_1 \mid s_1 \xrightarrow{\alpha_1} s_1' \text{ then } \exists s_2' \in S_2 \mid s_2 \xrightarrow{\phi(\alpha_1)} s_2' \text{ and } (s_1', s_2') \in R_\phi \text{ and}$$

$$\text{If } \exists s_2' \in S_2 \mid s_2 \xrightarrow{\alpha_2} s_2' \text{ then } \exists s_1' \in S_1 \mid s_1 \xrightarrow{\phi^{-1}(\alpha_2)} s_1' \text{ and } (s_1', s_2') \in R_\phi$$

Two LOTOS behaviour expressions B_1 and B_2 are ϕ -bisimilar iff for some coding function ϕ from Sys_1 into Sys_2 there is a ϕ -bisimulation relation for Sys_1 and Sys_2 .