Towards Integrated Cognitive and Interface Analysis

Howard Bowman[1], Giorgio Faconti[2] and Mieke Massink[2]

[1] *Computing Lab, University of Kent at Canterbury, Canterbury, Kent, CT2 7NF, UK*
[2] *CNR-Istituto CNUCE, Via V. Alfieri 1, Loc. Ghezzano - 56100, Pisa, Italy*

1 Introduction

Formal methods were conceived with application in the computing system development process in mind. However, the resulting techniques can more generally be viewed as methods to write abstract (non-prescriptive) descriptions of system behaviour and then formally analyse these descriptions to determine their emergent properties, where the term system is used in a very general sense.

To some extent, this more general applicability of formal methods has been recognised and "non-standard" applications of these techniques have been made. For example, there have been applications to modelling (critical) procedures and documents, to analysing biological systems and computer interfaces [13]. The work presented here fits into the latter of these non-standard applications. However, our application of formal methods to HCI is unusual since it focuses on syndetic modelling - a particular approach to analysing the usability of computer interfaces in which the interface and the cognitive model of the user are analysised together in the same notation and in an integrated fashion. In this paper, we argue that the formal description technique LOTOS can be advantageously applied to such syndetic modelling.

Human Computer Interaction. The next generation of human-computer interfaces will be extremely complex, incorporating sophisticated interaction mechanisms, such as *gestural* and *multimodal* interaction. Furthermore, it is clear that if these interaction mechanisms are used in an unconstrained manner interfaces can be developed which are very difficult to use. As an illustration [10,9] shows how the combination of mouse-based pointing gestures and spoken phrases in the MATIS system [18] is not as effective as expected due to the demands of competing cognitive resources.

Thus, there is a clear need to assess how cognitively demanding particular interaction tasks are. The standard approach to such assessment is to construct a prototype system implementation and perform user trials. However, this is both time consuming and expensive. Thus, along with many others, we consider how cognitive models can be used in making such an assessment.

A powerful approach to such assessment is to describe both the interface and the chosen cognitive model in the same notation and then analyse the cognitive behaviour in the context of the particular interface. The term *syndetic modelling* has been used to describe such combined specification and analysis [10,9,8].

However, such an integrated approach to specification and analysis is very demanding. In particular we can highlight the following four major difficulties:-

(1) **General Specification Principles**
A description notation which is appropriate for modelling both the cognitive and interface behaviour must be identified. The key to such a quest is to locate "general" structuring and interaction paradigms (i.e. generic means to structure systems into components and mechanisms by which components can interact).

(2) **Incomplete Understanding of Cognitive Behaviour**
Firstly, cognitive behaviour is highly complex in nature and secondly, our understanding of it, as represented by existing

cognitive architectures, is far from complete. Thus, giving a complete description of cognitive behaviour is certainly not possible and appropriate abstractions have to be employed.

(3) **Scalability**

Although an obvious requirement, the need for scalability is without doubt critical. In particular, a full description of any non-trivial cognitive architecture will necessarily be very large and, in addition, interface behaviour can be extremely complex. Thus, syndetic specifications will certainly have two main (large) components, each of which will contain sub-components. In addressing this issue of scalability we seek specification structuring techniques which have two characteristics:

(a) *Compositional.* We would like to be able to build up specifications in a compositional manner by adding new components without having to break the encapsulation of existing components.

(b) *Hierachical.* A major aspect that supports scalability is the ability to build up specifications in a hierarchical manner, for example, at a particular level of decomposition, being able to wrap up a complex behaviour in a component and use the resulting component at a higher level of specification. This implies that we need to allow components to themselves be structured in terms of components. Note that some techniques fail in this respect by either being completely flat, e.g. petri nets * [19] or only allowing one level of component structure, e.g. (timed) automata approaches such as UPPAAL [3].

(4) **Interpretation of Results**

The complexity of the cognitive and interface specifications can make it difficult to interpret the combined behaviour in a user/designer friendly manner. This is especially the case if the chosen specification notation is formal in nature, which will be the case in this paper and the user/designer is not a formal methods expert. To resolve this problem, techniques

* Although hierarchical petri nets to some extent resolve this problem.

are required for systematically hiding parts of specifications. Thus, enabling only the points of behaviour that are relevant to a particular analysis to be visible.

LOTOS. This paper does not claim that all these requirements can be fully realised with the current state of research, rather it strives to make a non-trivial contribution to their realisation. Our proposal in this respect is to use a process calculus as the syndetic modelling notation. From within the process calculus canon we have selected LOTOS [4] because it has been used relatively extensively in HCI modelling. However, its use in modelling cognitive behaviour is new.

There are many reasons for selecting LOTOS (and process calculi in general), see for example [5]; here we concentrate on how it addresses the four requirements for syndetic modelling just highlighted. In fact, the body of the paper will be structured in terms of each of these requirements, each section explains how our LOTOS based approach addresses a particular requirement. However, it is important to note that the discussion here arises from a large body of work on using LOTOS to model cognitive behaviour, which is reported in [5,7].

In addition, it is beyond the scope of this paper to give a full introduction to LOTOS. Thus, a certain knowledge of the notation is assumed. Also, throughout the paper we use a reduced LOTOS notation in order to simplify presentation. For example, gate lists are not included in process definitions.

ICS. The cognitive model chosen is *Interacting Cognitive Subsystems* (ICS) [1]. [2] argue that the cognitive theories typically employed in HCI, e.g. the GOMS family of models, are directed towards the analysis of low-level, well specified, cognitive functions, such as predicting performance times for particular tasks and that consequently they have limited scope. In contrast, ICS attempts to provide a "unified" general purpose cognitive framework and this broad scope is crucial when modelling in interactionally rich settings; such as multi-modal interfaces. In addition,

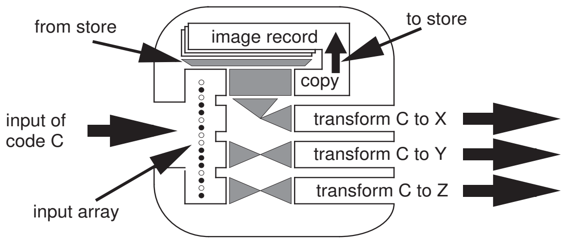

Fig. 1. General ICS Subsystem Format

there has been previous work, e.g. [10,9] on applying ICS in HCI, which we will build upon.

Structure of Paper. In the next section (2) we give a very summarised introduction to ICS. Then, in sections 3, 4, 5 and 6, we work through the four requirements for syndetic modelling just introduced, discussing in turn how LOTOS fulfills each requirement. Section 7 demonstrates methodologically how LOTOS can be used to analyse the usability of multi-modal user interfaces, and finally, section 8 presents concluding remarks.

2 Interacting Cognitive Subsystems

We now give a very brief review of ICS, for a complete presentation of the architecture the interested reader is referred to [1].

Information Flows and Representations. The basic "data" items found in ICS are *representations*. This term embraces all forms of mental codes, from "patterns of shapes and colour" as found in visual sensory systems; to "descriptions of entities and relationships in semantic space" as found in semantic subsystems [1]. We assume a set `Rep` of representations which contains a null element, denoted `null`.

These representations are past amongst the components of the architecture, being transformed from one code to another in each component. Thus, the architecture can be seen as an *information flow* model.

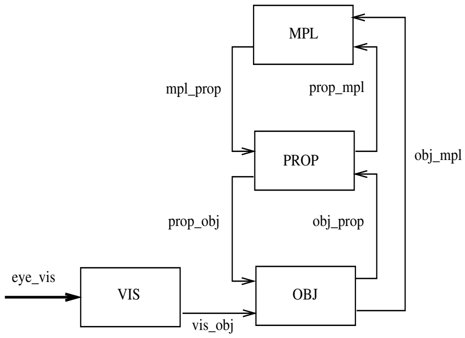

Fig. 2. A Reading Configuration

Subsystems. The components of the architecture are called *subsystems* and all subsystems have the same general format, which is shown in figure 1. Each subsystem itself contains components. For example, representations received by a subsystem are stored in the *input array*.

Each subsystem contains a set of *transformations* which take representations from the input array, apply some transformational operations to them and then relay a new (transformed) representation to a target subsystem.

We do not consider the *image record* here, see [5] for a discussion.

The Architecture. Rather than present the full ICS architecture, which would be difficult within the confines of this paper, we concentrate on a particular configuration of the architecture - a *reading* configuration, shown in figure 2.

Each subsystem is a specialization of the general subsystem format just highlighted. The roles of the subsystems shown are:-

- *Visual* (`VIS`) - receives representations from the eyes encoding "patterns of shapes and colour", i.e. light wavelength (hue) and brightness;
- *Morphonolexical* (`MPL`) - works with an abstract structural description of entities and relationships in sound space, i.e. lexical identities of words, their status and order;
- *Object* (`OBJ`) - works with an abstract structural description of entities and relationships in visual space, e.g. attributes of

objects: shape and relative position;

- *Propositional* (PROP) - works with descriptions of entities and relationships in semantic space, i.e. gives semantic meaning to entities and highlights the semantic relationships between entities;

The possible transformations between subsystems are shown in figure 2.

Multiple Flows and Blending. Sources of representation flows are typically sensory subsystems, e.g. VIS. Each representation is then relayed within the architecture by the occurrence of transformations[**]. Multiple flows can exist in the architecture at the same time.

The architecture accommodates a number of different outcomes when multiple flows are received. However, the interesting one is if an output transformation acts on a representation which is a combination of two (or more) "competing" input representations. This possibility leads to the concept of *blending*.

Representations from different flows can be blended to create a composite representation. However, the nature of the blending depends upon the cognitive task being considered. For example blending might only be possible if the two representations are, in some appropriate sense, *consistent* [1].

3 General Specification Principles

Here we consider the two issues of fully general structuring and interaction principles.

[**]There is actually a debate concerning how representations are relayed through the architecture. Here we assume *discrete* transformation firing. This is a reasonable abstraction for our purposes.

3.1 Structuring

A common approach to the disciplined construction of software systems is the use of abstract well-defined structures as a way of packaging the description of system components into units that can be used as building blocks. In the area of software development this has lead to structuring principles such as schemas in Z, processes, modules and classes.

The principle structuring construct in LOTOS is the *process*. A process is an autonomous and concurrently evolving entity.

Each process contains a number of interaction points at which it can communicate with its environment, i.e. with the other concurrently evolving processses. We view the notion of a process as a suitably general structuring paradigm to underly syndetic modelling. This is testified to by the observation that basic components of both the cognitive architecture and the interface can be modelled as LOTOS processes, see subsection 3.3.

3.2 Interaction

Clearly in a model constructed with autonomous components a mechanism needs to be provided which enables components to interact. Furthermore, if our chosen notation is going to be appropriate this interaction paradigm must be primitive enough to underly inter-component communication in both the interface and the cognitive domain. We believe that the process calculi interaction paradigm is sufficiently primitive.

Processes in process calculi interact by performing a synchronous rendez-vous/handshake. When both processes are ready, an atomic[⋆⋆⋆] synchronisation and associated transfer of data occurs. Such primitive interactions yield the process calculus concept of an *action*.

[⋆⋆⋆] This assumption of atomicity is important because it justifies the interleaving interpretation of concurrency, which is central to the process calculus approach. For example, simulation tools are predicated upon interleaving.

The primitive nature of such an interaction paradigm can be seen from the observation that more complex interaction mechanisms, such as asynchronous or shared memory communication, can be constructed from action based interaction and can thus be viewed as derived behaviour [14,17].

Furthermore, interaction in the cognitive domain can be constructed using the synchronous rendez-vous. Interaction in ICS is based on transformation occurences. Such events are modelled in the LOTOS interpretation as action executions. For example, the action instance,

```
vis_obj?r:Rep
```

models the `OBJ` subsystem receiving a representation (which will be bound to the variable `r`) from `VIS` on the transformation `vis_obj`.

3.3 Illustration

As an illustration, we offer the following examples of an interactor based interface and an ICS description in LOTOS:-

- *Interface Interactors.* For the structured description of *interactive* software, *interactor models* have been developed [12,15]. Interactor models form an abstract framework for the description of components within an interactive system. The generic interactor model can be specialised to focus attention on particular issues of system behaviour by embedding the basic interactor model into a particular language or modelling approach.

  The LOTOS Interactor Model (LIM), describes interactor behaviour in LOTOS. It organises the actions used to describe system behaviour along three dimensions: type of action (control or information), originator (application or user side), and direction (input and output). The interactor is considered as an entity that is able to mediate between the user and the application side. It gives feedback on user generated input and

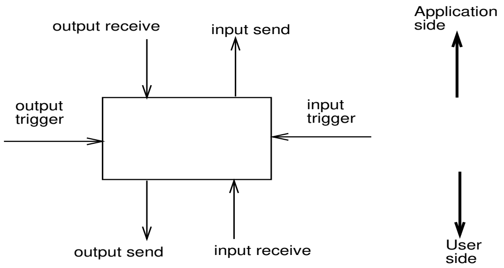

Fig. 3. External view of interactor

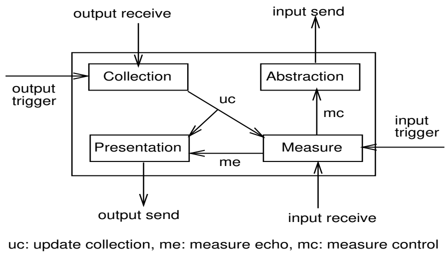

Fig. 4. Internal view of interactor

it uses trigger events to indicate further input and output. An external view of a LIM-interactor is given in figure 3.

It shows the communication to the user and the application side of the interface and the triggers for input and output.

The internal view of a LIM-interactor is shown in figure 4. The structure has been based on the Computer Graphics Reference Model, but the information processed by a LIM interactor does not need to be graphical. The interactor consists of four (sub)processes. In the *collection* an abstract representation of the information is kept that is manipulated and represented by the interactor. The *presentation* part gets the abstract representation when the *collection* is triggered. It uses this representation to make information perceivable to the user or to pass it on to a lower-level interactor. In the *measure* component, input from the user is collected. When this component is triggered it passes the input to the *abstraction* component, where it is converted into an abstract representation that can be passed on to the application or to a higher level interactor.

The following is an example of a LIM-interactor modelling the behaviour of a generic Logical Input Device (LID) [11].

```
M :=  im1; me; M  []  ...  []  imj; me; M  []
      it1; mc; M  []  ...  []  itm; mc; M
P := me ; eo ; P
```

```
A := mc ; od ; A
```

The LID is specified as the parallel composition of a Measuring (M), a Presentation (P) and an Abstraction (A) component which are all specified as LOTOS processes. The actions `im1` to `imj` model the input received by the Measure process. The actions `it1` to `itj` model the input triggers. The action `mo` is the output sent by the presentation and `od` the output generated by the Abstraction process.

A LID is then specified as the parallel composition of the above processes appropriately synchronized with `me` and `mc` hidden:

```
LID := hide me,mc in ((P ||| A) |[me,mc]| M)
```

- *ICS.* All ICS subsystems have the same general format, which is shown in figure 1. Consequently, the LOTOS subsystem descriptions also have a general format. For example, the `OBJ` subsystem would be defined as:

```
OBJ(iA:inArr,...) :=
(   vis_obj?r1:Rep; exit(...)
||| prop_obj?r2:Rep; exit(...)
(* Input Ports *)
|||
(   obj_mpl!tranOM(get(iA)); exit(..)
||| obj_prop!tranOP(get(iA)); exit(..)
||| obj_lim!tranOL(get(iA)); exit(..) )
(* Output Ports *) )
>> accept r1,r2:Rep in OBJ(#(r1,r2,0,0),...)
```

which uses a data structure `iA` to model the input array[⋆⋆⋆⋆]; `get` and `tranON` are data operations which respectively get and transform the relevant element from an input array; and `>>` is sequential composition.

Thus, the subsystem performs all its five transformations (two input, `vis_obj` and `prop_obj`, and three output, `obj_mpl`, `obj_prop` and `obj_lim`) independently and then recurses (through the sequential composition), updating the input array on the way.

Assuming we have process definitions for all subsystems we can build the top level behaviour of ICS using parallel composition. As an illustration, the reading configuration shown in figure 2 can be modelled using the following top level composition of subsystems:

```
(( VIS(...) |[vis_obj]| OBJ(...) )
   |[obj_prop,prop_obj]| PROP(...) )
   |[obj_mpl,prop_mpl,mpl_prop]| MPL(...)
```

4 Incomplete Understanding of Cognitive Behaviour

To address the problem that cognitive behaviour is only partially explained, suitable levels of abstraction to describe cognitive models must be identified. We believe that the abstraction

[⋆⋆⋆⋆] Actually there are other data structures which it is beyond the scope of this paper to discuss.

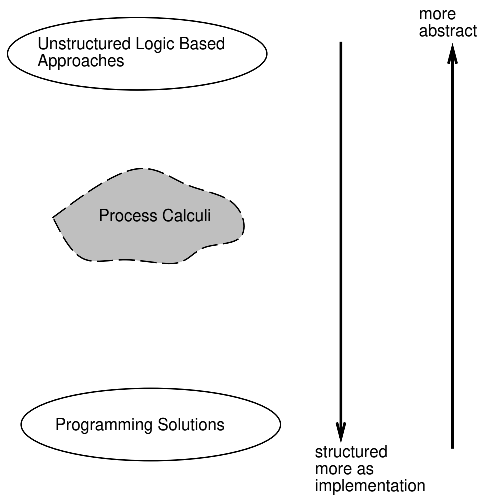

Fig. 5. The Spectrum of Available Modelling Techniques

techniques provided by process calculi facilitate such a level of specification.

There is a spectrum of available modelling techniques, see figure 5, with the two extremes being programming based approaches, such as those typically used to implement cognitive models, e.g. the LISP program's underlying SOAR, and abstract uses of mathematical logic, e.g. temporal logic[⋆⋆⋆⋆⋆]. A weakness of the former approaches is that they are often too prescriptive, forcing a particular "mechanistic" interpretation on the cognitive model, leaving it unclear which aspect of the programs behaviour results from the cognitive model and which arises from implementation decisions. In formal terms, programs only characterise a single implementation. In contrast, abstract logical techniques can characterise a set of possible implementations. Thus, enabling specification which is not prescriptive about implementation details. However, logical descriptions often express global properties across the entire system. Consequently, such approaches often fail to reflect the underlying component structure of the system being modelled. In addition, they typically fail to support execution of a specification, even in a simulated form.

[⋆⋆⋆⋆⋆] Note that here we do not mean logic programming approaches, rather we refer to pure abstract logic, which in contrast to Prolog say, does not contain framing of data.

Process calculi can be seen to sit between these two extremes, see figure 5. Firstly, the LOTOS specification we have given certainly reflects the component structure of the ICS model, e.g. we have a LOTOS process for each ICS subsystem. This makes the specification easier to understand and to maintain. Previous Modal Action Logic [10,9] based descriptions of ICS have not so directly reflected this component structure. Secondly, they enable simulated execution using tools such as LOLA and Smile [16].

Thirdly, process calculi provide techniques for avoiding overprescriptive description. In particular, they facilitate loose specification by allowing descriptions to contain non-determinism.

Non-determinism arises naturally in process calculi as a by-product of concurrency, since a process cannot look inside a concurrently evolving process, to know what it can do, it views its behaviour as non-deterministic. However in addition, non-determinism can be used to avoid prescriptive description of behaviour. Specifically, many possible behaviours can be included in the same specification, with the choice between them left unspecified.

Furthermore, non-determinism possesses very nice mathematical properties. For example,

> For a large class of properties, any such property that holds over a specification `S` will also hold over any specification that is "more deterministic" than `S` (see [5], for an exact characterization).

This means that any such property we can prove about an abstract (i.e. non-deterministic) specification will also hold over any more concrete (i.e. more deterministic) specification.

As an illustration, we can define a hierachy of interpretations of blending. For example, assume that `obj_prop` acts upon a blend of representations `r1` and `r2` (which have been placed in the `OBJ` input array from `VIS` and `PROP`), see figure 6. There are a number of possible ways of generating the new representation `r`:

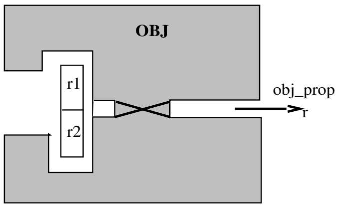

Fig. 6. Blending

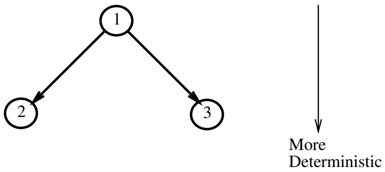

Fig. 7. Hierarchy of Non-determinism

(1) $\texttt{r}\in\texttt{Rep}$, i.e. `r` is randomly chosen from the set of all representations;
(2) $\texttt{r}=\texttt{r1} \vee \texttt{r}=\texttt{r2}$, i.e. `r` is a random choice of `r1` and `r2`;
(3) $\texttt{cons(r1,r2)} \Rightarrow \texttt{r}=\texttt{comp(r1,r2)}$ and
$\neg\texttt{cons(r1,r2)} \Rightarrow \texttt{r}=\texttt{null}$
if `r1` and `r2` are "consistent" then compose them together otherwise return `null`.

Thus, 1. is the most non-deterministic solution, as shown in figure 7. Note that although the extreme non-determinism inherent in 1. makes the solution cognitively strange, i.e. `r` has no relation to `r1` or `r2`, this is still an analytically useful interpretation. Specifically, for analysis of many cognitive properties we will only be interested (or may only need to be interested) in the blending which occurs at certain subsystems and we can leave all other blending completely unspecified, i.e. we do not care what representation is selected.

5 Scalability

LOTOS enables construction of large system specifications in a compositional and hierachical manner.

- *Compositionality.* The LOTOS parallel operator, `|[G]|`, is compositionally powerful. New behaviour can be added incrementally without breaking the encapsulation of existing processes. Furthermore, the operator can either be used *structurally* (i.e. to add components found in the target system) or *conjunctively* (i.e. to add "global" constraints, in the style of logical conjunction). The latter possibility yields the, so called, *constrainted oriented* style of specification which has been argued to be a major benefit of LOTOS.

- *Hierarchical.* Processes can themselves contain processes and thus, can contain concurrenct behaviour. As a syndetic modelling illustration of this hierarchy of concurrency. We can describe the top level behaviour of a syndetic analysis as:

  ```
  Interface(...) |[G]| ICS(...)
  ```

  where the two constituent processes could be defined in the style shown in section 3, each of which contains concurrent behaviour, and `G` is the set of common actions between the interface and ICS, e.g. control of a mouse interactor either directly (or indirectly) via the `lim_hand` ICS transformation.

6 Interpretation of Results

Process calculi come with a powerful set of tools for analysing and interpreting specifications.

- *Compaction.* Firstly, the complexity of the complete specification of both the interface and the cognitive architecture can, in some way, be hidden using the LOTOS hiding operator. This allows a set of actions to be hidden from the environment. Thus, if the set of actions that are relevant to a particular analysis can be identified, all other actions can be hidden. For example, if we are interested to observe/analyse the behaviour of ICS only at its sensory and effector ports, we can do this by hiding all other actions, here the set of actions `G`, i.e.,

  ```
  hide G in ICS(...)
  ```

  Furthermore, state spaces containing internal behaviour, can be reduced by applying equivalences, such as weak bisimulation and testing equivalence [14,17]. These equivalences relate specifications that are in some appropriate sense, indistinguishable to an external observer. Importantly, observably indistinguishable specifications may have very different internal behaviour, the level of internal complexity of which can vary dramatically.

- *Analysis.* A number of the available process calculi analysis techniques can be employed in the context of syndetic modelling. We list three techniques.

(1) *Simulated Execution.* Tools such as LOLA and Smile [16] enable specifications to be executed in a simulation environment. The approach is that the specification is run, with the user of the tool interactively resolving choices and non-determinism (automated resolution of such branches is also possible). Simulated execution can be combined with internal action compaction by just observing the behaviour of the specification at certain interaction points.

(2) *Verification.* Tools can be used, such as testing and model checking, to automatically determine whether the syndetic specification satisfies certain properties. With testing, the property is coded as a test process and then the specification is analysed to see if it will pass or fail the test. With model checking the property is coded in temporal logic and then the model checker automatically analyses whether the syndetic specification satisfies the property.

(3) *Logical Deduction.* Although powerful, simulated execution and verification techniques can not be applied in all situations. For example, when properties about infinite state space systems are considered, deductive reasoning is typically required. This can either be performed in the process calculus using axiomatizations of such calculi or in an associated logic.

7 Complete Architecture

[5] describes a specification and then analysis of ICS in the context of a number of cognitive properties. Unfortunately, it is beyond the scope of this paper to fully describe this body of work, however we summarise it here.

- *LOTOS Specification.* Using the principles highlighted in the previous sections of this paper, a LOTOS specification of ICS is given. Semantically, LOTOS specifications can be interpreted as a set of state sequences, called *intervals*. States in these intervals contain a distinguished variable which indicates the action that occurs at the state. Thus, new states are entered when actions are executed. We let $\Omega(\mathtt{P})$ denote the intervals of $\mathtt{P}$.
- *Goal Formulation Logic.* We introduce an interval temporal logic which can be used to formulate cognitive properties of ICS. This is based upon the logic Mexitl which was described in [6]. This logic is interpreted over the intervals described in the last bullet point. Thus, giving us a semantic link between LOTOS and interval temporal logic.
- *Case Study.* We analyse the capabilities of ICS to perform certain multi-modal tasks. These tasks have arisen from assessment of the MATIC system and have also been considered in [5] and in [10,9]. For example, a typical negative property that we analyse is:

  $$(\forall \mathtt{r}_1 \neq \mathtt{r}_2)$$
  $$ICS \models \neg \Diamond(\mathbf{speak}(\mathtt{r}_1) \ \wedge \ \Diamond\mathbf{located}(\mathtt{r}_2))$$

  where, ICS is the LOTOS specification of ICS; $S \models \phi$ states that the specification S satisfies the formula ϕ; $\mathtt{r}_i$ are representations and $\Diamond\psi$ holds over an interval which contains a subinterval where ψ holds. Informally, this property states that it is not possible to speak one representation and locate (i.e. point at with, say a mouse) a different representation at the "same" time[⋆⋆⋆⋆⋆⋆].

  A typical positive property would be:

  $$(\forall \mathtt{r})\ (\exists \sigma \in \Omega(ICS))$$
  $$\sigma \vdash \Diamond(\mathbf{speak}(\mathtt{r}) \ \wedge \ \Diamond\mathbf{located}(\mathtt{r}))$$

  which, informally, states that it is possible to speak and locate the same representation at the "same" time.

[⋆⋆⋆⋆⋆⋆] Actually, the use of different representations here is slightly subtle, to be more precise $\mathtt{r}_1$ and $\mathtt{r}_2$ denote representations with different psychological subjects.

- *Analysis.* Simulation and deductive reasoning are used to perform this analysis. Specifically, we verify properties of the form of the above negative property using deductive reasoning in the interval temporal logic. This reasoning uses an axiomatization of the logic. In contrast, positive properties are verified constructively using the simulation tool LOLA. Thus, a fulfilling trace is interactively constructed through simulated execution of the specification.

8 Conclusions

We have motivated the use of LOTOS in syndetic modelling. LOTOS has been used in modelling the human-computer interface before. However, our use of the notation for modelling cognitive behaviour is new. In addition, we believe that LOTOS provides an interesting alternative to Modal Action Logic which has typically been used in syndetic modelling [10,9]. Our main preference for LOTOS is that we believe it provides an appropriate level of abstraction for integrated interface and cognitive specification and analysis, since it sits between prescriptive (programmed) and very abstract (logical) modelling notations.

More broadly, we believe that such exercises in applying formal methods to description and analysis of cognitive and HCI models holds great potential. In particular, the abstraction level provided by such techniques seems intuitively to be appropriate for such analysis.

Acknowledgements

The work presented here has been performed in the context of the TMR TACIT project (funded by the EU under the TMR programme, Contract N. ERBFMRXCT970133) and thus, we must thank all the members of the project. In particular, we would like to thank David Duke, David Duce, Jon May and Phil Barnard

with whom we have had valuable discussions about ICS.

References

[1] P.J. Barnard. Interactive cognitive subsystems: Modelling working memory phenomena with a multi-processor architecture. In A. Miyake and P. Shah, editors, *Models of Working Memory*. Cambridge University Press, 1998.

[2] P.J Barnard and J. May. Representing cognitive activity in complex tasks. *Human-Computer Interaction*, 1999. to appear.

[3] Johan Bengtsson, Kim G. Larsen, Fredrik Larsson, and Paul Petters son amd Wang Yi. Uppaal - a tool suite for automatic verification of real-time system. In *Proceedings of the 4th DIMACS Workshop on Verification and Control of Hybrid Systems*, 1995.

[4] T. Bolognesi and E. Brinksma. Introduction to the ISO Specification Language LOTOS. *Comp. Networks and ISDN Systems*, 14(1):25–29, 1988.

[5] H. Bowman. An interpretation of cognitive theory in concurreny theory (long version). Technical Report 8-98, Computing Laboratory, University of Kent at Canterbury, 1998. http://www.cs.ukc.ac.uk/pubs/1998/646/index.local.

[6] H. Bowman, H. Cameron, P. King, and S. Thompson. Specification and Prototyping of Structured Multimedia Documents using Interval Temporal Logic. In *Int. Conf. on Temporal Logic*, Applied Logic Series. Kluwer, July 1997.

[7] H. Bowman and G. Faconti. Analysing cognitive behaviour using lotos and mexitl. *Formal Aspects of Computing*, 11:132–159, 1999.

[8] D. Duke, G. Faconti, and M. Massink. Synchronisation and delay in a formal model of user cognition. In S.Constantine, editor, *Proceedings of the ERCIM Workshop on User Interfaces for All*, Prague, 1996. http://www.ics.forth.gr/ercim-wg-ui4all/UI4ALL-96/proceedings.html.

[9] D.J. Duke, P.J. Barnard, D.A. Duce, and J. May. Syndetic modelling. *Human Computer Interaction*, 13(4):337–393, 1998.

[10] D.J. Duke, P.J. Barnard, J. May, and D.A. Duce. Systematic development of the human interface. In *APSEC'95, Second Asia Pacific Software Engineering Conference, Brisbane*. IEEE Computer Society Press, December 1995.

[11] G. Faconti, A. Fornari, and N. Zani. Visual representation of formal specification: an application to hierarchical logical input devices. In *Focus on Computer Graphics Series*. Springer-Verlag, 1994.

[12] G. Faconti and F. Paterno. An approach to the formal specification of the components of an interaction. In *Eurographics'90*. North Holland, 1990.

[13] FM-Elsewhere. *Web page on Formal Methods Elsewhere*, 2000. http://www.cs.ukc.ac.uk/people/staff/hb5/Elsewhere.

[14] C. A. R. Hoare. *Communicating Sequential Processes.* Prentice Hall, 1985.

[15] ISO/IEC. *Computer Graphics Reference Model.* International Standards Organization, 1992. ISO/IEC 11072.

[16] LOTOSPHERE. *LOTOS Integrated Tool Environment.* LOTOSPHERE Project, 1988. http://wwwtios.cs.utwente.nl/lotos/lite/.

[17] R. Milner. *Communication and Concurrency.* Prentice-Hall, 1989.

[18] L. Nigay and J. Coutaz. A generic platform for addressing the multimodal challenge. In *Proceedings of ACM CHI'95*, pages 98–105. ACM Press, 1995.

[19] W. Reisig. *Petri Nets, An Introduction.* Springer-Verlag, 1982.