Modelling Concurrent Cognitive Architectures using Process Calculi

Howard Bowman
Computing Laboratory,
University of Kent at Canterbury,
Canterbury, Kent, CT2 7NF, UK
+44-1227-764000
H.Bowman@ukc.ac.uk

ABSTRACT
Theories of concurrent systems have been extensively investigated in the computer science domain. However, these theories are very general in nature and hence, we would argue, are applicable to many disciplines in which concurrency arises. Furthermore, a number of existing theories of cognitive science are concurrent in nature. Thus, we investigate the application of a (process calculi based) concurrency theory to modelling Interacting Cognitive Subsystems, which is such a (concurrent) cognitive theory. Then we consider the capabilities of the cognitive system to perform combinations of speech and gesture in multi-modal human computer interaction.

Keywords
Concurrency Theory, Interactive Cognitive Subsystems, Process Calculi

INTRODUCTION

Many different notations have been used to describe cognitive models. For example, at least three different classes of notation have been used to give descriptions of the pivotal cognitive architecture SOAR:

- natural language descriptions augmented with box and arrow diagrams, e.g. [Newell, 1990];
- executable implementations, e.g. the Lisp and C programs underlying the standard Soar implementation; and
- even descriptions by [Milnes, 1992] in the formal specification notation Z, which is a combination of set theory, first order logic and constructs for structuring specifications.

It is clear that the choice of notation dramatically affects the "value" of (even the ability to complete) a description of a cognitive model. In a very general sense, selection of an appropriate modelling notation can be a major enabler to problem solving. To take a familiar illustration, the uptake of the arabic number system in the middle ages crucially enabled the progress of arithmetic, e.g. the development of arithmetic manipulation techniques, such as long division, which would have been infeasible with, for example, the roman number system.

It is thus natural to believe that the identification of appropriate modelling notations, which offer a suitable level of abstraction can aid the progress of cognitive modelling. It is with such identification in mind that the work reported here has been performed. Specifically, the underlying tenet for our work is that a set of new techniques from formal computer science can be advantageously applied to describing and analysing cognitive models. These techniques have arisen out of the field of *concurrency theory*.

Early concurrency theory work, which yielded petri nets, was followed in the 80's by the development of a wealth of techniques, e.g. communicating automata, further petri nets research, temporal logics and the techniques we will be interested in in this paper - process calculi [Milner, 1989]. Although developed with computer applications in mind the core concepts of concurrency theory are completely general and are applicable to modelling any variety of concurrent system.

The relevance of concurrency theory to cognitive modelling is that most cognitive theories are, at some level, concurrent. For example, Soar contains elements of concurrent behaviour. Furthermore, there has been recent interest in decentralized models, where cognition is modelled in terms of a collection of independently evolving (and equally statused) components, which interact. The particular such architecture that we consider is Interacting Cognitive Subsystems (ICS) [Barnard, 1998].

We have chosen ICS for a number of reasons.

Firstly, the architecture has been used successfully to analyse multi-modal human computer interaction, which is the field from which this work has arisen. Secondly, there has been previous work, e.g. [Duke and Duce, 1996] on modelling ICS with formal methods[1]. Thirdly, the concurrent nature of ICS suggests that from within the formal methods canon, concurrency theory techniques are an appropriate choice.

The particular area of cognitive science our work focuses on is analysis of combinations of speech and gesture in multi-modal human computer interaction. In fact, this extended abstract has grown out of a large body of work we have performed on describing and analysing such speech/gesture combinations using ICS and process calculi. The complete description of this work runs to 90 pages [Bowman, 1998]. This extended abstract summarises some of the main issues surrounding this work, without delving into the technical details.

We will first give a very brief outline of ICS. Then we discuss two reasons for using process calculi to model ICS - they allow concurrency to be modelled directly and they support *abstract specification*. We then consider how cognitive goals which combine speech and gesture can be analysed using process calculi and finally, we present some concluding remarks.

INTERACTIVE COGNITIVE SUBSYSTEMS

ICS adopts a "top down" approach to the design of a cognitive theory by providing a framework containing a set of core components and mechanisms that, it is argued, give a "potential design of a complete mental mechanism" [Barnard, 1998].

We give a brief review of ICS, for a complete presentation the interested reader is referred to [Barnard, 1998].

Representations and Subsystems. The basic data item in ICS is the representation[2]. These 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.

The components of the architecture are called *subsystems* and all subsystems have the same general format, which is shown in figure 1 (above).

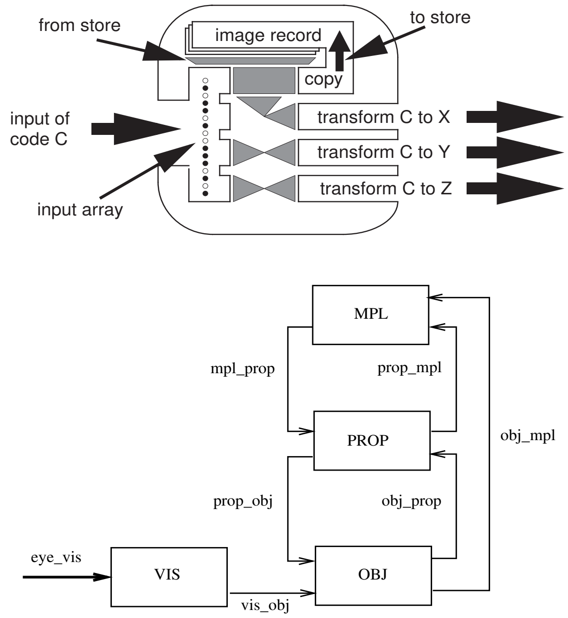

Figure 1: Subsystem Format and Reading Config.

Each subsystem itself contains components. For example, representations received by a subsystem are placed 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. Such transformations are also shown in figure 1 (below). We do not consider the *image record* here.

The Architecture. Rather than present the full ICS architecture we concentrate on a particular configuration of the architecture - a *reading* configuration, see figure 1 (below). 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

[1] This term describes the set of *mathematically based* computer science specification and analysis techniques, of which concurrency theory techniques are an example.

[2] 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 [Barnard, 1998].

space, i.e. gives semantic meaning to entities and highlights the semantic relationships between entities;

Also the concurrent nature of the architecture should be becoming clear - subsystems evolve simultaneously and independently subject to interaction between subsystems when representation converting transformations are performed.

Blending. Sensory subsystems, e.g. VIS, are a common source of representation flows. Each representation is then relayed within the architecture by the occurrence of transformations[3]. Multiple flows can exist in the architecture at the same time. The architecture accommodates a number of different outcomes in this situation. 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* [Barnard, 1998].

CONCURRENCY

The majority of work on mathematical theories of computing has focused on systems, which can be categorised as *sequential*. Such systems can typically be viewed as input to output transformers. Although perfectly adequate in the sequential setting, such *transformational* interpretations are insufficient in the concurrent setting. Concurrency theory has responded to this problem. It studies systems containing a number of components that evolve simultaneously. Such forms of concurrent behaviour can be found throughout the different bands of cognitive activity, e.g. the neuronal, neural circuit, cognitive operation and task levels [Newell, 1990].

With transformational systems the key issue is what results the computation terminates with, however with concurrent systems this is no longer the case. The interesting aspect of concurrent systems is rather their ongoing behaviour and how components respond to external stimuli throughout the system's life-time. Thus, concurrent systems are modelled in terms of the order in which they can perform external interactions.

We have interpreted ICS using a particular technique from the concurrency theory domain - the process calculus LOTOS [Bolognesi and Brinksma, 1988]. This contains operators to describe concurrent components and interaction between components. Our specification of ICS and a LOTOS introduction can be found in [Bowman, 1998].

The principle structuring construct in LOTOS is the *process*. A process is an autonomous and concurrently evolving entity, e.g. figure 2 (i) depicts three processes - the big circles. Each process contains a number of interaction points, the small squares, at which it can communicate with other concurrently evolving processses.

Clearly in a model constructed with autonomous components a mechanism needs to be provided which enables components to interact, e.g. the arrow in figure 2 (i). The synchronous rendez-vous of process calculi is such a notion of interaction. When both processes are ready, a synchronisation and associated transfer of data occurs. Such primitive interactions yield the concept of an *action*.

Interaction in the cognitive domain can be constructed using the synchronous rendez-vous. For example, 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.

As an illustration of our specification, assuming we have definitions for all subsystems, we can build the top level behaviour of ICS using *parallel composition*, i.e. the notation $\mathsf{C}_1 \,|[\mathsf{a}_1,...,\mathsf{a}_n]|\, \mathsf{C}_2$ states that components C_1 and C_2 evolve in parallel subject to interaction on $\mathsf{a}_1,...,\mathsf{a}_n$. Thus, an event a_i can only be performed when both C_1 and C_2 are ready to perform it. As an illustration, the reading sub-configurations of ICS, which we depcited in figure 1 (below), can be modelled using parallel composition as,

```
(( VISUAL(...) |[vis_obj]| OBJECT(...) )
|[obj_prop,prop_obj]|
PROPOSITIONAL(...) )
|[obj_mpl,prop_mpl,mpl_prop]|
MORPHONOLEXICAL(...)
```

This states that the VISUAL and OBJECT subsystems evolve concurrently, while exchanging representations via the transformation vis_obj; which

[3]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.

in turn evolve concurrently with the PROPOSITIONAL subsystem while exchanging representations on obj_prop and prop_obj; and so on.

ABSTRACT SPECIFICATION

Importantly, formal specifications are in nature very different to computer programs or what we will more broadly call *implementations*. A formal specification is abstract in the sense that it characterises a set of possible implementations (the implementations that satisfy it), while a program characterises a single implementation - itself.

Associated with this aspect is the desire not to overspecify (or in other terms to provide *loose specification*), i.e. that the nature of the specification language should not force the specifier to rule out acceptable implementations. We believe this feature of formal specification is very useful in the cognitive setting.

An important issue in modern cognitive science is, what has been called, the *irrelevant specification problem* [Newell, 1990]. In order to construct a working simulation program a large number of assumptions have to be made, leaving it unclear what aspect of the behaviour of the program corresponds to known cognitive behaviour and what arises from expediency. For example [Cooper et al., 1996] state,

> "Computational models conflate empirically justified mechanisms with pragmatic implementation details, and essential theoretical aspects of theories are frequently hard to identify"

In fact, [Cooper et al., 1996] have directly targeted this issue. Their approach is to use a re-engineered version of Prolog which keeps the theoretical and implementation assumptions disjoint, thus enabling one to observe the consequences of changing particular implementation assumptions.

The approach we advocate is even more radical and further from conventional implementation programming. We would argue that the irrelevant specification problem arises because cognitive theories are closer to specifications than implementations/programs. Cognitive theories typically leave much unexplained since a complete *mechanistic* interpretation of cognition is not available. Thus, a cognitive model is an abstract description of behaviour, for which the implementation details can be filled in in many ways. Using the terminology of abstract specification, a particular programming implementation of a cognitive model is an implementation which *satisfies* the cognitive model. Importantly, it certainly is *not* the cognitive model itself. Thus, our approach has been to *specify* ICS abstractly, yielding a description which characterises many possible actual implementations.

A major way in which abstract specification is supported in process calculi is through *non-determinism*. This allows many possible behaviours to be included in the same specification, with the choice between them left unspecified. Such non-determinism is used in many places in our LOTOS interpretation of ICS. For example, we use non-determinism to model the ICS concept of blending.

As an illustration, we can define a hierachy of interpretations of blending ([Bowman, 1998] actually presents a much larger and more detailed hierarchy). For example, assuming a set Rep of representations which contains a null element, denoted null and 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 2 (ii), there are a number of possible ways of generating the new representation r and these possible ways can be placed in a hierachy, see figure 2 (iii), according to their level of non-determinism. We highlight three ways here,

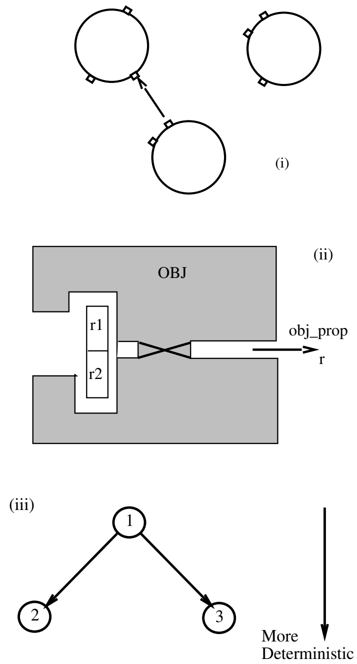

Figure 2: Assorted Figures

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

1. gives an upper bound on the level of non-determinism - it is a completely non-deterministic approach. Note that although the extreme non-determinism inherent in 1. makes the approach 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.

Approach 2. has a similar flavour to approach 1., the difference being that the set from which the representation is chosen is restricted to the two relevant representations.

In contrast, in approach 3. the two input representations are compared to determine if they are consistent, e.g. whether they are representations with the same *psychological subject*, if they are consistent, a representation which in some way combines the features of the two input representations is generated. Of course, there are many ways in which such a combined representation could be constructed and these different approaches will arise in different cognitive tasks, at different subsystems. However, the important issue is that all such approaches can be related according to their level of non-determinism.

One of the really nice aspects of how non-determinism behaves in process calculi, is that, not only does it support abstract specification, it also allows (simulated) execution and proof based verification.

Simulated Execution. A difficult problem that arises with "abstract specifications" is how to provide "executable" realizations. For example, this can be a problem if descriptions are given in pure logic, e.g. in first order or temporal logics. However, while being abstract, process caculus descriptions are still "algorithmic" and can thus, be executed using a simulation engine. The approach is that the specification is run, with the user of the simulator interactively resolving choices and non-determinism, yielding an execution trace.

Furthermore, by composing in parallel a process which plays the role of an implementation environment, i.e. resolves choices in a particular manner, the specification can be executed according to a particular implementation policy. This is equivalent to imposing particular implementation assumptions, i.e. in the terminology of [Cooper et al., 1996] enforcing "below the line" assumptions. By changing this implementation environment process we can assess the effects of different implementation assumptions (e.g. different interpretations of blending) in the same manner as [Cooper et al., 1996].

Goal Verification. By associating a logic with our process calculus we can assert properties/goals of a specification of a cognitive model (as illustrated in our case study discussion which follows shortly). Furthermore, non-determinism possesses very nice mathematical properties in this respect. For example, it can be shown that for any negative property (see [Bowman, 1998] and the next section) that,

> if the property holds over a specification S
> it will also hold over any specification that
> is "more deterministic" than S.

In terms of the irrelevant specification problem this implies that any negative property that we can deduce from our abstract specification of the cognitive model will also hold over its concrete implementations. This is a very valuable methodological device. For example, much of the reasoning we can make with our "most abstract" interpretation of blending will hold for all its instantiations.

CASE STUDY

[Bowman, 1998] describes a specification and then analysis of ICS in the context of a number of such speech/gesture goals. We summarise this work here.

Specification. A LOTOS specification of ICS is given. Semantically, LOTOS specifications can be interpreted as a set of state sequences, called *intervals*[4]. We let $\Omega(\mathsf{S})$ denote the intervals of a specification S. Then an interval temporal logic is introduced which can be used to formulate cognitive goals of ICS. It is interpreted over intervals. Thus, giving a semantic link to the LOTOS specification.

Goal Formulation. The capabilities of ICS to perform combined speech and gesture tasks is considered. Such *deictic* interaction is a good example of multi-modal human-computer interaction. Analysis of such combined speech and gesture modalities is particularly significant since it addresses a common myth in HCI, which is that since human to human communication commonly combines such modes of interaction, it should be beneficial to devise similar combinations of human-computer interactions. The analyse proceeds by first formulating the cognitive goals that are of interest. These goals come in two varieties - negative and positive goals.

A typical negative property that is analysed is:

$$(\forall \mathsf{r}_1 \neq \mathsf{r}_2)$$
$$ICS \models \neg \diamondsuit(\mathbf{speak}(\mathsf{r}_1) \ \wedge \ \diamondsuit\, \mathbf{located}(\mathsf{r}_2))$$

[4] Such an interval can be viewed as a run/execution of the specification.

where, ICS is the LOTOS specification of ICS; $S \models \phi$ states that the specification S satisfies the formula ϕ; r_i are representations and $\diamondsuit\,\psi$ 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[5].

A typical positive property which, informally, states that it is possible to speak and locate *the same* representation at the "same" time, would be:

$$(\forall \mathsf{r})\ (\exists \sigma \in \Omega(ICS))$$
$$\sigma \vdash \diamondsuit(\mathbf{speak}(\mathsf{r})\ \wedge\ \diamondsuit\,\mathbf{located}(\mathsf{r}))$$

Analysis. Simulation and deductive reasoning are used. Properties of the form of the above negative property are verified using deductive reasoning in the logic. In contrast, positive properties are verified by interactively constructing a fulfilling trace using the simulation tool LOLA.

Using these analysis techniques both the above properties can be shown to hold.

CONCLUSIONS

There is a spectrum of available modelling techniques with the two extremes being programming based approaches, such as those typically used to implement cognitive models, e.g. the LISP programs underlying SOAR, and abstract uses of mathematical logic, e.g. temporal logic[6]. 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 typically fail to support execution of a specification, even in a simulated form.

Process calculi can be seen to sit between these two extremes. Firstly, the LOTOS specification we have given enables simulated execution. Secondly, process calculi provide techniques for avoiding over-prescriptive description. In particular, they facilitate loose specification by allowing descriptions to contain non-determinism.

[5] Actually, the use of different representations here is slightly subtle, to be more precise r_1 and r_2 denote representations with different psychological subjects.

[6] 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.

In conclusion, as stated at the start of this paper, using an appropriate modelling notation can be a great enabler to "problem solving". This extended abstract has argued that a number of aspects of process caculi suggest they may be an appropriate modelling notation in the domain of concurrent cognitive architectures.

References

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

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

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

[Cooper et al., 1996] Cooper, R., Fox, J., Farringdon, J., and Shallice, T. (1996). A systematic methodology for cognitive modelling. *Artificial Intelligence*, 85:3–44.

[Duke and Duce, 1996] Duke, D. and Duce, D. (1996). Syndetic modelling: A new opportunity for formal methods. Technical Report ID/WP57, Amodeus-2 Project Report, WWW: http://www.mrc-cbu.cam.ac.uk/amodeus/abstracts/id/id_wp57.html.

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

[Milnes, 1992] Milnes, B. (1992). A specification of the Soar cognitive architecture in Z. Technical Report CMU-CSS-92-169, Carnegie Mellon University.

[Newell, 1990] Newell, A. (1990). *Unified Theories of Cognition*. Harvard University Press.