PROTOCOL MODELS OF HUMAN-COMPUTER INTERACTION

Ashley McNeile, Ella Roubtsova and Gerrit van der Veer
Metamaxim Ltd, UK and Open University of the Netherlands, The Netherlands
ashley.mcneile@metamaxim.com, ella.roubtsova@ou.nl, gerritvanderveer@ou.nl

Keywords: Human-computer interaction, Protocol Models, separation of concerns, CSP composition, local reasoning.

Abstract: Current approaches to modeling human-computer interaction do not always succeed in producing behaviorally complete models of manageable size and complexity. We argue that the reason for this lies in lack of support for *parallel* composition of *partial behavioral descriptions*, and propose the recently developed *Protocol Modeling* approach as a superior alternative. The semantics of Protocol Modeling support separation and composition of concerns in models of human-computer interaction, and the production of executable models to explore and refine the desired behavior. Protocol Modeling supports a crucial property sought in modeling methods if it is to scale to complex problems, namely the ability to reason about the modeled behavior of the whole based on examination only of a part (sometimes called ”modular” or ”local” reasoning).

1 INTRODUCTION

Human Computer Interaction (HCI), perhaps more than any other aspect of a system, calls for the need to be able to capture *complex behavior*. A user interface may have *multiple simultaneous states* associated with different *concurrent aspects* of a user's task. This is the kind of situation which, if modeled in an inappropriate way, can lead to high complexity driven by combinatorial explosion, with the result that model becomes intellectually unmanageable when applied to a large problem.

Modeling approaches commonly used to model HCI behavior fail to handle complexity well because they use an unsatisfactory approach to decomposition of a large problem into smaller, simpler parts. For instance two widely promoted approaches, *User Virtual Machine* (UVM) (M.Tauber, 1988; A. Dix, G. Abowd, R. Bealle, J. Finlaj, 1998; D. Hix, H. Hartson, 1998; S. Payne, T. Green, 2000) and *Coloured Petri Nets* (P. Palanque, R. Bastide, L. Dourte, C. Sibertin-Blanc, 1993), support two basic methods of combining parts and hence to restructure a large model into parts: *linear merging* and *hierarchical structuring*. The first of these means concatenation of sub-models such that some output states of one model become the input states of another model. The second means replacing a node of a model by a sub-model which has its own structure, a technique that can be applied recursively to create a hierarchy of models.

While both of these techniques allow a problem to be broken up, both result in parts that can only be understood by reference to the rest of the model and how all the parts work together. Moreover, neither addresses the issue of multiple simultaneous states associated with different concurrent aspects of a user's task that is a common feature of HCI requirements, for which a parallel (rather than linear or hierarchical) composition technique is essential. As a result decomposition in these approaches does not generally support maintenance of intellectual control over a model as complexity grows.

Our thesis is that the right way to achieve scalability is through *parallel composition of partial behavioral descriptions* in such a way that the parts can be reasoned about independently of each other. In this paper we examine the ability of the *Protocol Modeling* (PM) approach (A. McNeile, N. Simons, 2006) to support this paradigm. Although the domain of HCI is a novel application area for Protocol Modeling, it appears well suited to it. This paper is structured as follows:

- Section 2 presents a case study and shows it as both a conventional HCI model and a PM model.
- Section 3 explains and illustrates the semantics of the PM approach using the case study.
- Section 4 concludes the paper by showing the role of the PM approach in HCI design.

2 CASE STUDY

We illustrate the UVM approach by modeling the HCI of the search subsystem of the information system BaMaS (BaMaS, 2007). BaMas collects and provides information about links and bridge programmes between Bachelor and Master degree programmes in the Netherlands. After finishing a Bachelor Programme at one university a student can continue with a Masters Programme at the same or another university, provided that his/her Bachelor Programme is a recognized qualification for the chosen Masters degree. A user of the BaMaS system can choose an Institution (University) and a Bachelor Programme and investigate which programmes this Bachelor degree qualifies him or her to pursue. Alternatively, a user can choose an Institution or a Masters Programme and find which Bachelor Programmes meet the requirements for admission into this Masters Programme, with or without a bridge program.

UVM of Search in BaMaS. The UVM HCI model of the search subsystem is shown in Figure 1.

- The states are depicted as ellipses. For example, state *AllB, AllM* indicates that no selection of an institute has been made by the user. State *IB, AllM* means that an Institution for Bachelor has been chosen.
- A transition is represented by an arc labelled by the events that cause the transition. For example, the transition from *AllB, AllM* to *IB, AllM* is caused by event *Select IB*.

The human-computer interaction is modeled as a whole, so the interaction during the selection of an institute cannot be separated from the search of information about the links and bridge programs between Bachelor and Master courses. If the UVM needs to be extended and redrawn, states and arcs can be added but there is no guarantee that the functionality of the initial UVM is preserved. After extension or change the model must be fully regression tested, and it is notoriously difficult to ensure that it has not been ”broken” by a modification.

Protocol Model of Search in BaMaS. In order to show the difference of UVM and PM models we show the Protocol Model of the search functionality in BaMaS without explanation of semantic details, which are covered in the next section. The PM model comprises (Figure 2) four small protocol machines:

- Machine *Selection for Bachelor* represents the human-computer protocol for selection of an institute and a programme for a Bachelor programme.

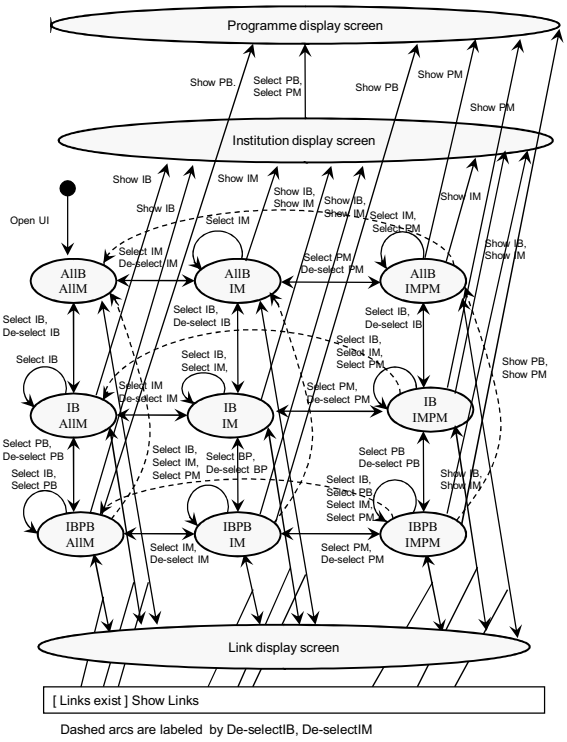

Figure 1: UVM model of the search in BaMaS.

- Machine *Selection for Master* similarly handles the selection steps for a Masters programme.
- Machine *Display Screens* presents the screens that can be seen: the choice screen, the screen about the chosen Institution, the screen about the chosen programme and the screen presenting links between programmes.
- Machine *Link Display* presents the two possible consequences of the choices of programmes: *”No links”* between programmes and *”Links exist”*.

The composition of those protocol machines, as explained in the next section, models all the functionality of the search subsystem.

3 PROTOCOL MACHINES

This section provides a brief summary of the semantics of Protocol Modeling. A fuller description is given in (A. McNeile, N. Simons, 2006).

Events. An ”event” in PM (more properly ”event instance”) is the data representation of an occurrence in the environment as a set of data attributes. Every event is an instance of an event type, and the type of an event determines its metadata or attribute schema.

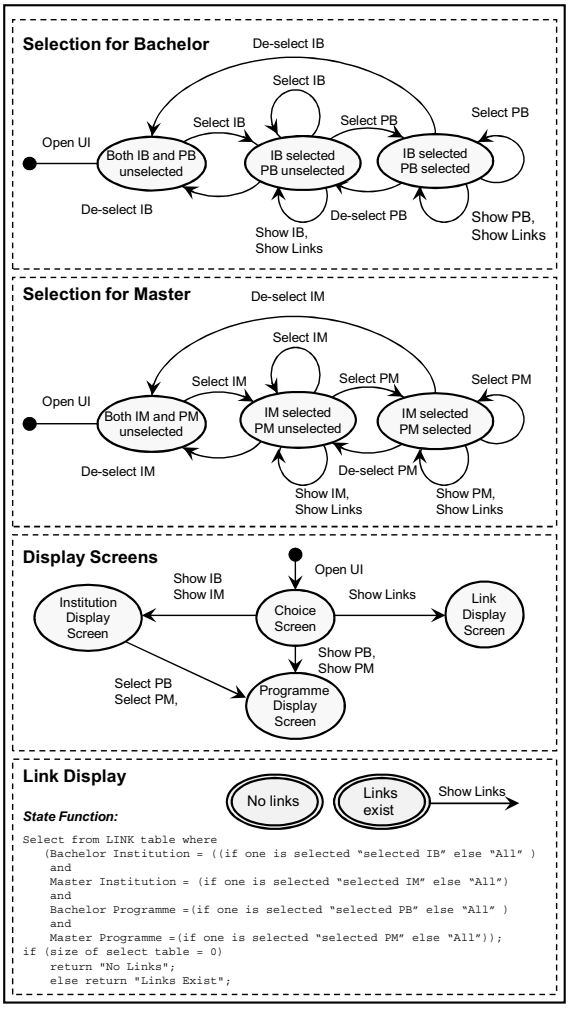

Figure 2: PM model of the search in BaMaS.

An attribute schema is being the set of data attributes that completely define an instance of the event type. A protocol machine has an ”alphabet” of event types that it understands. For example, machine *Selection for Bachelor* understands events {*Open UI, Select IB, Select PB, Show PB, Show Links, De-select PB, De-select IB*}

States and Variables. Between handling events, a protocol machine rests in a well defined quiescent state, meaning that it can undergo no further change of state unless and until presented with a new event. A machine may only be presented with a new event when in such a state.

A state of Protocol Machine is specified by its name and by values of local variables. A protocol machine can remember information from events in local variables. For example, protocol machine *Selection for Bachelor* has local variables presenting the chosen institution *IB* and the chosen programme *PB*. Protocol machines can read values of local variables of other composed protocol machines, but cannot change them.

Behavior of a PM. When a protocol machine is presented with an event it will either *ignore* it, *accept* it or *refuse* it as follows:

- When a machine is presented with an event that *is not* represented in its alphabet, the machine ignores it.
- When presented with an event that *is* represented in its alphabet, it will either accept it or refuse it.
- Acceptance or refusal of an event by the machine is determined by rules that the machine evaluates based on the values of its accessible storage.

Note that ”refusal” means that the machine is unable to handle the event at all, and this normally means that some kind of error message is generated back to the environment. How or where such an error is generated is not of concern for modeling purposes.

Composition. Composing two protocol machines yields another protocol machine. The alphabet of the composed machines is the union of the alphabets of the constituent machines; and the local storage of the composed machine is the union of the local storages of the constituent machines. The rules for whether the composed machine accepts, refuses or ignores a presented event are:

- If both constituent machines ignore the event, the composed machine ignores it;
- If either constituent machine refuses the event, the composed machine refuses it;
- Otherwise the composed machine accepts the event.

These rules correspond to the parallel composition operator $(P \parallel Q)$ of Hoare’s process algebra, Communicating Sequential Processes (C. Hoare, 1985).

The definition of a PM does not require that its state is stored, and so it is possible to have the state of a PM returned by a function (called the machine’s *State Function*). This is exactly analogous to a derived or calculated attribute, where the attribute’s value is calculated on-the-fly when it is required. Derived states are represented diagrammatically by a double outline around the state. The protocol machine *Link Display* (Figure 2) has derived states: *No links* and *Links exist*. Protocol machines with derived states take part in composition exactly like machines with stored states. Applying the composition rules, event *Show Links* is accepted iff it is

accepted by all machines: *Display Screens*, *Link Display*, *Selection for Bachelor* and *Selection for Master*. If, for example, machine *Link Display* is in the derived state *No links*, the event *Show Link* is refused (not possible).

Local Reasoning in Protocol Models. An important property of CSP composition is that it guarantees the ability to reason about the behavior of the whole (the result of composition) based on examination of the parts in isolation. This property is known as *local (or modular) reasoning* and is based on the fact that CSP composition ensures *Observational Consistency* (J. Ebert, G. Engels, 1994) between a composite machine and its constituents. Formally: If we take a sequence, S, of events that is accepted by the composition ($M_1 \parallel M_2$), then the subsequence, S', of S obtained by removing all events in S that are not in the alphabet of M_1 would be accepted by the machine M_1 by itself. In other words, composing another machine with M_1 cannot "break its trace behavior". For our BaMaS example the local reasoning means that, based on examination *Selection for Bachelor* alone, we can determine that the sequence $\langle OpenUI, SelectPB, SelectIM, SelectPM \rangle$ is not a possible sequence for *Selection for Bachelor*$\|$*Selection for Master*, as $\langle OpenUI, SelectPB \rangle$ is not a trace of *Selection for Bachelor*.

Local reasoning of this kind is an important in facilitating separate modeling of the parts of a software system, and in retaining intellectual control over complexity as the model grows. This property of CSP composition was established by Hoare in his work on CSP (C. Hoare, 1985). However, Hoare did not consider events with data or machines with derived states, as are used by Protocol Machines. The full proof of support for local reasoning for Protocol Machines can be found in (A. McNeile, E. Roubtsova, 2008).

4 CONCLUSIONS

The Protocol Modeling approach makes it possible make separate representations of parallel concerns within an HCI model and use CSP for composition of the parts. The use of CSP composition ensures that the behavior of the PM component models is preserved in the result.

HCI models built using the PM approach are directly executable using a suitable tool. The key features of such a tool are support for automatic composition of PMs, according to the CSP composition rules. Protocol Modeling support is implemented in the ModelScope tool (Metamaxim, 2006).

ACKNOWLEDGEMENTS

We thank Ad Slootmaker (Educational Technology Expertise Center) and Evert van de Vrie (OU) for sharing information about BaMaS.

REFERENCES

A. Dix, G. Abowd, R. Bealle, J. Finlaj (1998). *Human-computer interaction*. Prentice Hall Europe.

A. McNeile, E. Roubtsova (2008). CSP parallel composition of aspect models. *To appear in the Proceedings of the 12th International Workshop on Aspect-Oriented Modelling, AOM 2008, Brussels, ACM DL*.

A. McNeile, N. Simons (2006). Protocol Modelling. A modelling approach that supports reusable behavioural abstractions. *Software and System Modeling*, 5(1):91–107.

BaMaS (2007). BaMaS. http://www.bamas.nl.

C. Hoare (1985). *Communicating Sequential Processes*. Prentice-Hall International.

D. Hix, H. Hartson (1998). *Developing User Interfaces: Ensuring Usability Through Product & Process*. John Wiley & Sons Inc.

J. Ebert, G. Engels (1994). Dynamic models and Behavioural Views. *LNCS 858*.

Metamaxim (2006). *http://www.metamaxim.com/*.

M.Tauber (1988). On Mental Models and the User Interface. *G. Van der Veer et al. ed. "Working with Computers: Theory Versus Outcome"*.

P. Palanque, R. Bastide, L. Dourte, C. Sibertin-Blanc (1993). Design of User-Driven Interfaces Using Petri Nets and Objects. *LNCS 685*.

S. Payne, T. Green (2000). Usability Patterns for Applications on the World Wide Web. *In D. Diaper ed. "Task Analysis for Human-Computer Interaction, Ellis Horwood, Cambridge MA"*.