Supporting Resource-Based Analysis of Task Information Needs

José Creissac Campos[1] and Gavin J. Doherty[2]

[1] Departamento de Informática,
Universidade do Minho, Braga, Portugal
jose.campos@di.uminho.pt
[2] Department of Computer Science,
Trinity College Dublin, Ireland
Gavin.Doherty@cs.tcd.ie

Abstract. We investigate here an approach to modelling the dynamic information requirements of a user performing a number of tasks, addressing both the provision and representation of information, viewing the information as being distributed across a set of resources. From knowledge of available resources at the user interface, and task information needs we can identify whether the system provides the user with adequate support for task execution. We look at how we can use tools to help reason about these issues, and illustrate their use through an example. We also consider a full range of analyses suggested using this approach which could potentially be supported by automated reasoning systems.

1 Introduction

Usability testing is a time consuming and expensive activity, often performed too late in the software development life-cycle. Thus it is interesting to look at analyses which may be performed early in the development life cycle, particularly if we can obtain leverage from design artifacts which are produced for other purposes within the development process. Task models (such as CTT models [12]) are an obvious type of artifact which may be produced at such an early stage. Task models on their own however, are not sufficient for the analysis of modern interactive systems, as the same basic task structure can be supported by many possible designs. The focus of this paper is on information needs; whether the right information is provided in an appropriate fashion at the right time. While there is an obvious and direct mapping between user tasks and their information needs, this is not the same as knowing how best to represent information, nor how to reconcile competing information requirements for multiple tasks within a given application, with fixed and possibly very limited screen "real estate" (as in mobile devices). We must also avoid confusing and inconsistent information displays where an excessive degree of task based adaptation is employed.

Thus we can see an opportunity for model-based techniques that can be applied early in development, as they can capture this information about proposed designs and be used to reason about the designs. Some behavioural models focus on the system and can be supported by automated reasoning systems. The capabilities of such models to explore issues related to interaction has been the focus of much work in the

S.W. Gilroy and M.D. Harrison (Eds.): DSVIS 2005, LNCS 3941, pp. 188–200, 2006.

past [13, 3, 14, 9]. Another family of techniques are based on task models as mentioned above. Again the use of such models in the analysis of interactive systems is well documented [6, 12]. Providing analyses which could help answer the questions regarding information needs detailed above would require us to augment our task models with models to account for system behaviour. The contribution of this paper is threefold:

1. To look at analyses which may be performed by checking an interactive system specification against a task model.
2. To examine the use of *resources* as a concept and model component to allow us to move beyond what may be achieved with interactor models.
3. To formalise and automate a portion of the analysis, reducing analyst effort, increasing the likelihood of finding problems and opening the door to more intelligent dynamic behaviour by applications.

2 Reasoning About System Designs

In previous work we have approached reasoning about system designs from two distinct perspectives. We have shown that using device models (with an emphasis on behaviour) and model checking, we can identify behaviours that might lead into undesirable situations [3]. We have also shown how using partial models of system, interface and user (with an emphasis on representation) we can analyse whether a representation is appropriate for a specific use [4].

In both cases we look mainly for situations where the properties under analysis would fail. It is mainly in such situations that design knowledge about the system will be generated. In the case of the first approach, the aim is to identify possible behaviours that would lead the system into undesirable states. The cognitive plausibility of the traces needs to be determined, and assumptions about user perception must be encoded into the properties under verification. A major issue is that non-plausible behaviour must be identified and filtered out, ie. cases where potential problems are flagged but which in fact correspond to implausible behaviour by the user. Task models can be introduced as a means of reducing these unwanted false positives, but at the cost of making the analysis consider normative behaviours only. While this approach helps determine situations where something wrong will or might happen, it tells us little about the degree of support the interface provides to its users in achieving their goals.

The second approach attempts to reason about whether there is a straightforward and accurate representation of information in the interface appropriate for the way it is used in the user's task. A failure to prove the equivalence between the different models involved will highlight problems in the representation, reflected in the mapping between the system's internal state and the mental model built by the users from the interface. However, this approach is difficult to apply to complex behaviours, as it does not directly support analyses of the changing set of information needs over a typical task performance.

Recent years have seen a move towards a more contextualised and situated view of interaction, where users react and adapt to a variety of information sources and stimuli in the "environment" in order to achieve their goals. This has been accompanied by a

shift in the technology towards more pervasive and context-aware computing systems. This shift in emphasis places greater strain on the quality of the user interface, since it must provide sufficient appropriate information such that users can carry out their activities.

To address this problem we propose to relate tasks to the information and information representations that support them, and identify situations where support is not adequate. In order to do this, we take the following model components [2]:

- Device model—what is available at the interface, and how the interface behaves.
- Task model—how goals can be achieved.

The user's ability to infer a task model from the interface influences the execution gap in Norman's model [11]. At the very least, it is likely that the user will require information about the state of the system in order to decide on course of action and to check that the interaction is progressing as required. However, the nature of the information needs of the user may be more complex than this. One framework which allows us to think about the *nature* of information use is the resources model [15].

3 Resources for Action

The resources model is an interaction model whose aim is to support Distributed Cognition style analysis at an early stage of design, rather than as a means of understanding existing systems or prototypes. The model itself is simple; it views interaction as involving a number of information resources which can be characterised in terms of a number of different categories, depending on the role the information may play in interaction. Different interaction strategies on the part of the user (eg. goal-matching, plan following) will exploit different types of information, and place a different emphasis on such information. For those interested in developing mobile and context aware systems, the model is attractive since the information resources considered may be located in the environment, or provided by the system. The categories used to characterise information resources are as follows:

- *Plan*. A sequence of actions, events or states that should be carried out
- *Goal*. The required state of the world.
- *State*. A collection of relevant values of objects that feature in an interaction
- *Action-effect*. A relation between an action or event and its effect on the interaction.
- *History*. Actions, events or states already achieved in the interaction.
- *Possibilities*. The set of possible next actions of the user.

Thus we can posit a third starting point for our analysis—a model of user information needs, structured as resources needed for action.

A system comprised of a number of different devices, which together support the user in performing some complex task, with the user assessing the different devices according to his or her information needs, potentially requires a different form of analysis to reasoning about a user performing a task using a single device. An interesting aspect of a resource focus for the analysis is that it is consistent with such a heterogeneous multi-device view of interaction.

We view information elements in the interface as comprising resources, constituting not only information, but operations easily performed and encouraged by the representation. Such concepts can be incorporated into analytical models as perceptual and (logical) cognitive operations [4]. Modelling the use of information is itself an interesting problem; one possible approach would be to use the classification in the resources model to characterise information use.

Norman [11] explains the interaction process as a loop beginning with goal formation and ending with evaluation of the perceived response against the goal. When no strictly predefined plan exists, what the user will do depends solely on his or her evaluation of the system state and interpretation of the system behaviour. In this case we should provide resources appropriate to potential tasks and goals. We can also focus on actions themselves as a useful unit of analysis, without postulating detailed planning mechanisms. Instead, for possible actions which could be carried out as part of the user's activities we can ask if the action is properly resourced; is it possible to know if it is an appropriate action to take; what will the effect of the action be; will it bring the user closer to their goal; can the user evaluate how successful the action has been?

Resources take many shapes; but here we will focus on information presented at the interface, and actions available at the interface.

$$\text{Resource} = \text{Info}_{resource} + \text{Action}_{user} \tag{1}$$

We need to guarantee that user resource needs are met by the available resources in a dynamic task execution context. We need to ask what is the set of information resources which best suits the task at hand, at every stage, and under different (possibly concurrent) performance scenarios.

4 Device Model

In this section, we revisit the issue of device models, looking at how a standard device model (interactors) can be enriched with resource information. An interesting issue for further investigation would be to look at producing more dynamic interactive system specifications structured around the notion of resources themselves.

We can conduct our analysis using a range of behaviour modelling approaches. For simplicity and convenience, we use a domain specific language to model the systems: MAL interactors [3]. Figure 1 presents an example interactor.

We have discussed above how the concept of resource may help us to address a variety of questions regarding information needs in modern interactive systems; thus we have a question concerning how best to incorporate the notion of resources into an interactive system model.

In order to use automated reasoning support for the models we build, it is helpful to characterise what an interactor model is. An interactor defines a state space via its attributes, and the axioms define a relation on this state space labelled by the actions that cause the transitions:

$$\text{State}_i = \mathbb{P}\text{Attrib} \tag{2}$$

$$\text{Behaviour}_i = \mathbb{P}(\text{State}_i \times \text{Action} \times \text{State}_i) \tag{3}$$

interactor photocopier
attributes
door: {open, closed}
copying: boolean
error: {ok, abc, bc, ac, c}
vis display: {idle, copy, error_abc, error_bc, error_ac, error_c, door_open}
actions
vis open close start checkA checkB checkC
stop jam
axioms
per(open) → door=closed
[open] door'=open ∧ ¬copying' ∧ error'=error ∧ display'=door_open
per(close) → door=open
[close] door'=closed ∧ copying'=copying ∧ error'=error ∧ display'=idle
per(start) → ¬copying ∧ door=close ∧ error=ok
[start] copying' ∧ door'=close ∧ error'=ok ∧ display'=copy
per(stop) → copying
[stop] ¬copying ∧ door'=door ∧ error'=error ∧ display'=idle
...

Fig. 1. An example interactor

Available resources are associated with the modality that makes them available to the users. Different modalities might be selectable according to their appropriateness, eg. audio might be appropriate in one context (hands busy) and not in another (important meeting). Thus modalities are used to annotate appropriate attributes and actions. The resources define a state space (UI_{state}), which is a subset of the system state space defined by the interactor. Thus, for each user interface state we can identify a set of resources which are available at that point in the interaction.

$$\mathrm{UI}_{state} = \mathbb{P}\mathrm{Resource} \tag{4}$$

It is important to realise that there is not an exact match between the information resources used in the modelling and the information made visible by interactors. Resources needed for an action may be provided by several interactors. Conversely, we specify the resources needed for an action, and much of the information provided by an interactive system might not be relevant to this particular task. For example, an engineer repairing a machine might use an onboard display of status (state) information, along with a PDA displaying procedure information for the repair (plan information in a resource based view). Both the machine status panel and the handheld may be displaying additional information not relevant to the repair task. Actions can be classified as user actions (those with a modality) or system actions (response to user action or autonomous action).

$$\mathrm{Action} = \mathrm{Action}_{user} + \mathrm{Action}_{system} \tag{5}$$

Different interface components and/or resources might become available or unavailable as the interaction progresses. Also, modalities might also change over time.

5 Task Model

A task model defines the set of interactions that, from a given set of initial states of the system, lead to the fulfilment of a goal. These interactions can be seen as sequences of actions (traces of behaviour).

$$\text{Task} = \mathbb{P}\text{State}_i \times \mathbb{P}\text{Trace} \tag{6}$$

As above, two types of actions can typically be identified at task level: user actions (representing physical actions performed by the user at the user interface), and system actions (representing changes to the user interface performed by the system). A hierarchical task model can be mapped down onto such a set of traces, which may be reduced by operational and planning constraints, if present. This set of traces could be large, depending on the degree of visibility on the underlying system state, and the number of different ways to achieve the task using the interface, further motivating the use of tools. As a first approach we can model traces as sequences of actions:

$$Trace = Action^* \tag{7}$$

For clarity we will be using ConcurTaskTree (CTT) models [12] to support the exposition of ideas. CTT is a commonly used notation for task representation and analysis. Note, however, that the approach is not specific to CTT, and in fact is extensible to other task modelling languages. Consider the CTT model in figure 2. The behaviour it models can be expressed by the following set of traces:

{present jam info→open door→check A→check B→check C→close door,
present jam info→open door→check B→check C→close door,
present jam info→open door→check A→check C→close door, }

As can be seen CTT abstract nodes are not represented since they are a structuring mechanism only. For concurrent tasks there may be many possible interleavings of the traces.

Let $(\text{State}_i, \text{Behaviour}_i)$ be an interactor, using the definitions above we can already perform a number of tests:

Action completeness. All intended user actions are supported by the system.

$$\forall a \in \text{Action}_{user} \exists (s1, a, s2) \in \text{Behaviour}_i \tag{8}$$

This guarantees that all possible user actions the user might want to perform are possible in the system.

Predictability. No user action has two possible effects.

$$\forall a \in \text{Action}_{user}, s \in \text{UI}_{state} \exists^1 (s1, a, s2) \in \text{UI}_{behaviour} \tag{9}$$

This definition demands that no two actions might have the same effect in the interface. This might be too strong in specific cases (for instance, due to moding). In that case we can use states from State_i instead of from $\text{UI}_{behaviour}$ only. This way we are requiring that no two action will have the same effect on the system's state.

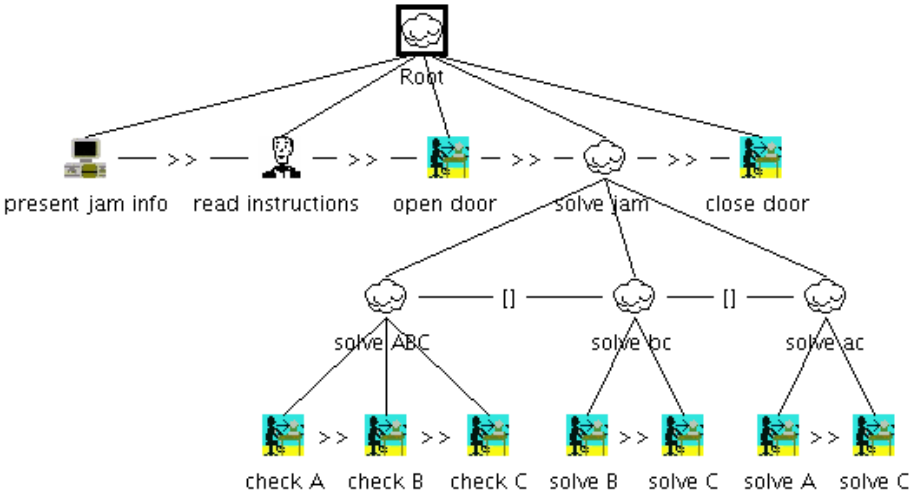

Fig. 2. An example task model

Task performance. The execution of a given task (s_t, ts) is possible if

$$\forall t \in ts\cdot, s \in s_t \text{trace\_poss}(s, t) \tag{10}$$

where

$$\text{trace\_poss}(s_t, t) \equiv \exists(s_t, head(t), s) \in \text{Behaviour}_i \wedge \text{trace\_poss}(s, tail(t))$$

The above type of analysis can be mechanised using model checking (cf. [2]), and indeed it would be difficult to perform in an accurate and complete fashion by hand. This analysis however, tells us only whether a given task is possible, not whether the system supports the user in performing it. To achieve this last goal, we need to consider the resources the system provides against the resources needed to perform the task.

6 Tasks and Resources

At specific stages in the interaction process, information resources will be needed by the user to decide on the course of action. It is this possibility of different courses of action that justifies using a set of traces to model a single task. In order to analyse this aspect we need to know whether the interface provides the adequate resources for the right choice to be made.

Using the models in the previous section would fail at this point because the process of decision is not explicitly represented in the traces. Hence, we must introduce another type of user action: choice actions (representing mental choices by the user regarding which physical action to take next).

$$\text{Action}_{user} = \text{Action}_{physical} + \text{Action}_{choice} \tag{11}$$

These actions exist in the CTT model as User Tasks. What needs to be done is to include them in the traces:

{present jam info→read instructions→open door→check A→check B→check C→close door,
present jam info→read instructions→open door→check B→check C→close door,
present jam info→read instructions→open door→check A→check C→close door, }

In order to analyse the system regarding its support for a given task we need to know what the task demands in terms of resources are. One obvious case is that each user action maps directly to the requirement that the resource corresponding to that action must be available at the interface at the required moment.

Additionally, for each user action in a task we must indicate what resources are needed (other more advanced options are discussed in section 8). For choice actions, there might be the need for some specific information to be present at the interface in order for the user to make the correct decision. For physical actions, besides the action being available, there might also be the need for some specific information to be present to prompt the user into performing the correct actions. We might consider whether information is made available at an earlier point in the task performance (trace) or whether it is visible at the current time. Focusing on resources, tasks become

$$\mathrm{Task}^{resourced} = \mathbb{P}\mathrm{State}_i \times \mathbb{P}\mathrm{Trace}^{resourced} \tag{12}$$

where State_i represents the states from which the task can be performed, and traces include the actions, and the resource needs:

$$\mathrm{Trace}^{resourced} = \mathrm{Action}^{resourced^*} \tag{13}$$

$$\mathrm{Action}^{resourced} = (\mathrm{Action}_{user} \times \mathbb{P}\mathrm{Resource}) + \mathrm{Action}_{system} \tag{14}$$

Hence, for each user action we can include a set of needed resources.

This information is not available at the original CTT model, and must be provided in order for the analysis to take place. Notice, however, that this is the type of information needed to carry out other types of analysis such as Cognitive Walkthroughs [8].

For example, if we go back to the example introduced in Figure 2 we can say that at the "read instructions" step the user needs the information about the type of jam to be present at the interface. We might also consider whether the user should be relied upon (or burden with the necessity) to remember the correct procedure. If we decide against that, all user actions after "read instructions" also require that information about the correct procedure be present at the interface. As an example, the trace for the ABC strategy becomes

present jam info→ (read instructions,{abc_error_info})→ (open door,{abc_error_info})→
(check A,{abc_error_info})→ (check B,{abc_error_info})→ (check C,{abc_error_info})→
(close door,{abc_error_info})

We can now verify if, for a given task (s_t, ts), the system always provides the needed resources. For that we must prove the following theorem:

$$\forall t \in ts, s \in s_t \cdot \mathrm{trace\_poss}^{resourced}(s, t) \tag{15}$$

where

$$\begin{aligned}&\text{trace\_poss}^{resourced}(s_t, t) \equiv \\ &\quad \exists_{(s_t, head(t), s) \in \text{Behaviour}_i} \cdot \quad (\text{is\_Action}_{system}(head(t)) \vee \pi_2(head(t)) \subseteq s_t) \\ &\qquad\qquad\qquad\qquad \wedge \text{trace\_poss}(s, tail(t))\end{aligned}$$

If this can be proved, then the needed actions and the specified information resources are available at each point in the task where they are needed. If not, then even if the task is possible, the user might find problems in performing it. While proving theorem 10 above would only guarantee that the task could be executed (nothing could be deduced regarding support to its execution), this analysis is comparable to a Cognitive Walkthrough type of approach, note however that we are attempting to make minimal assumptions about user cognition.

7 An Example

In this section we show how the approach can be applied to the running example in the context of the MAL interactors modelling and analysis work put forward in [3, 2]. Due to space constraints, we will not describe the modelling and analysis aspects in detail. Instead, we will focus on the impact of including resource related information into the analysis.

We use the model of a photocopier, already partially introduced in Figure 1, focusing specially on the handling of jamming problems. When a jam occurs, there are three different places inside the photocopier (A, B, C) which must be checked for paper. Depending on the type of jam a different sequence of actions must be performed. The photocopier has a display where this information is presented whenever a jam happens. Additionally the display can show information regarding the status of operation and of the door. Due to limited screen size, the display can only present one item of information at a time.

Using the model of the photocopier only we can already use i2smv and SMV to check whether it is possible to solve a jam problem. We will consider here the ABC jam error (all three places must be checked sequentially). First, we do a sanity check on the model and check whether a ABC jam is possible: EF(error=abc). What this formula states is that it possible to a a future state of the system where a ABC jam has occurred.

Next we check if the jam can be solved. This can be attempted by checking the following CTL formula: AG(error=abc –> AF(error=ok)). The property is not true. The model checker points out that the user might behave in a way which does not to solve the problem (for example, repeatedly opening and closing the door). At this stage we could introduce assumptions about user behaviour to the property being investigated. While this is useful in that it helps uncover those assumptions, it leads to hard to read properties. Also, since there are prescribed operating procedures for each type of jam, it would be interesting to analyse whether the interface adequately supports them.

We could use negation on the property to get a counter-example illustrating a strategy to solve the jam and compare it to our own pre-defined strategy. However, this will only

be one possibility among many (usually the shortest, but that is not guaranteed), and we still have to consider the cognitive plausibility of the user following that particular strategy, and the degree of support given by the interface to the strategy.

Instead we will directly encode the strategy into the model in a way similar to [2]. We start by modelling the procedure as an interactor that performs the sequence of steps expressed in the task model. We then place the two models together using the following interactor:

interactor main
includes
photocopier **via** device
solve_abc_jam **via** task
axioms
task.active → task.action=device.action
device.action=jam ↔ task.action=jam

What the axioms state is that: a) whenever the task is being performed, the system must behave according to the task description; b) if the device jams then the task model also processes that event (in order to activate the task).

We can now perform the tests proposed above. Testing the property AG(error=abc –> AF(error=ok)) we conclude that the strategy solves the problem. However we still do not know whether the strategy is supported by the interface. It might be, for example, that an action is required that in practice an user will have difficulty in identifying and performing. To address this, we include resource information in the task model. As has already been explained, we consider two types of resource needs:

- All of the needed action resources must be available when needed, otherwise it becomes physically impossible to perform the task.
- The strategy for solving the problem must be available during the solving procedure, otherwise the user will have to memorise the strategy or find some alternative means of externalising that knowledge.

Action resource verification is structurally guaranteed by the first theorem in the interactor above. If an action is not available when needed, task and device models cannot synchronize, and that behaviour deadlocks. Information resource needs are encoded using the following axiom:

$$\mathrm{task.action} \in \{\mathrm{open}, \mathrm{checkA}, \mathrm{checkB}, \mathrm{checkC}, \mathrm{close}\} \rightarrow \\ d\mathrm{evice.display} = \mathrm{error\_abc}$$

This axiom is an encoding of the resource testing condition of $\mathrm{trace\_poss}^{resourced}$ from Theorem 15 above.

Using this we can conclude that the system does not support the strategy as the combined system/task model is not capable of performing the task. The source of the problem resides in the fact that once the door is open the information regarding the strategy needed for solving the jam problem is no longer available. This happens because the door open information replaces the jamming information. In this particular design, closing the door again would not help since according to the model the jam information is no longer displayed.

If we enhance the display to be able to present two items of information so that the door open item does not hide the jam item, then it is possible to verify the model, and conclude that all needed resources to solve the jam are available.

8 Conclusions

A resources based approach can help in identifying potential usability problems by exploring what should be available at the interface to support users. We have presented an approach to the inclusion of resource needs into an approach to mechanised reasoning over models of interactive systems. This analysis complements a more unconstrained style of analysis where all possible behaviours of the system are analysed.

In [10] an approach to checking tasks against system models is put forward. The analysis is based on the assumption that the task model identifies the perceptual operations performed by the users. These perceptual operations must then have corresponding output operations in the device model. A resources based approach allows us to break away from this tight coupling between task and system operations. We can concentrate on the users' needs, and inspect whether the system provides the relevant information resources at the user interface. Additionally, we believe that by considering different types of resources we can expand on the type of analysis that can be performed.

Other approaches exist that attempt to fold human factors considerations into model-based approaches to reasoning about usability. One possibility is to build an explicit model of the user as in programmable user models (eg. PUMA [1]) and executable cognitive architectures (SOAR [7], EPIC [5] etc.). While these may deliver more detail on cognitive and performance aspects, they also tend to be complex to build to an automatable state. Another approach is to encode assumptions about the user directly into the model (cf. [14]). In this case the separation between device model and user assumptions is not clear and can bias the user assumptions towards those that are needed to make the system work. By working with assumptions at task level, we have a clear separation between models, assumptions about users are expressed in terms of tasks and user interface, and not directly about the system. Additionally, the models are relatively easy to build. The concept of resources is straightforward to work with and provides a natural way for the HCI expert to augment a task model.

We have outlined many more possibilities for analysis than could be explored fully for this paper (especially in terms of tool support). Among the suggested possibilities for further work, we have considered the following:

- Development of a more flexible view of resources, including representational (perceptual) aspects.
- Action based analysis with task fragments and constraints as an alternative to canonical task trees, which can also be seen as the use of partial specifications of user behaviour.
- Tool support for analysis of multiple and concurrent tasks. Could we help find a compromise set of resources which supports all or most tasks? A related question would be understanding conflict and interference between different tasks.
- Modelling changing resource needs. Can we integrate the resource model characterisations (plan, possibility, action effect, etc.) into a modelling approach and match

against appropriate presentations? How can this be done within the framework of existing task languages?
- Dynamic availability of resources—can we come up with designs with better and fewer transitions between sets of resources. Modelling the availability is straightforward, so this seems quite feasible.
- Encapsulating user interaction strategies, for example, when a user takes a side track to get some information. There are related questions about access to information in the history of an interaction which is required for a later action, and how this may be modelled.
- A new modelling approach directly based on resources. This is somewhat speculative but could lend itself towards intrinsically dynamic and adaptive interactive system specifications.

A final question pertains to the variety of possible analyses, how they relate to one another, and how they may be provided to the analyst as part of a coherent methodology.

Acknowledgements

The authors wish to thank Michael Harrison for his comments on earlier version of this work. Gavin Doherty would like to acknowledge the support of Enterprise Ireland in the form of an International Collaboration grant. José Campos would like to acknowledge the support of FCT (Portugal) and FEDER (EU) under contract POSC/EIA/56646/2004.

References

1. R. Butterworth, A. Blandford, D. Duke, and R. M. Young. Formal user models and methods for reasoning about interactive behaviour. In J. Siddiqi and C. Roast, editors, *Formal Aspects of the Human-Computer Interaction*, pages 176–192. SHU Press, 1998.
2. José C. Campos. Using task knowledge to guide interactor specifications analysis. In J. A. Jorge, N. J. Nunes, and J. Falcão e Cunha, editors, *Interactive Systems: Design, Specification and Verification — 10th International Workshop, DSV-IS 2003*, volume 2844 of *Lecture Notes in Computer Science*, pages 171–186. Springer, 2003.
3. José C. Campos and Michael D. Harrison. Model checking interactor specifications. *Automated Software Engineering*, 8(3-4):275–310, August 2001.
4. Gavin J. Doherty, José C. Campos, and Michael D. Harrison. Representational reasoning and verification. *Formal Aspects of Computing*, 12(4):260–277, 2000.
5. D. Kieras and D.E. Meyer. An overview of the EPIC architecture for cognition and performance with application to human-computer interaction. *Human-Computer Interaction*, 12:391–438, 1997.
6. B. Kirwan and L. Ainsworth. *A Guide to Task Analysis*. Taylor and Francis, 1992.
7. J.E. Laird, A. Newell, and P.S. Rosenbloom. Soar: An architecture for general intelligence. *Artificial Intelligence*, 33:1–64, 1987.
8. Clayton Lewis, Peter Polson, Cathleen Wharton, and John Rieman. Testing a walkthrough methodology for theory-based design of walk-up-and-use interfaces. In *CHI '90 Proceedings*, pages 235–242, New York, April 1990. ACM Press.
9. Karsten Loer. *Model-based Automated Analysis for Dependable Interactive Systems*. PhD thesis, Department of Computer Science, University of York, 2003.

10. D. Navarre et al. A tool suite for integrating task and system models through scenarios. In C. Johnson, editor, *Interactive Systems: Design, Specification, and Verification*, volume 2220 of *Lecture Notes in Computer Science*, pages 88–113. Springer, June 2001.
11. Donald E. Norman. *The Psychology of Everyday Things*. Basic Book Inc., 1988.
12. Fabio Paternò. *Model Based Design and Evaluation of Interactive Applications.* Applied Computing. Springer Verlag, Berlin, 1999.
13. Fabio D. Paternò. *A Method for Formal Specification and Verification of Interactive Systems*. PhD thesis, Department of Computer Science, University of York, 1995.
14. John Rushby. Using model checking to help discover mode confusions and other automation surprises. *Reliability Engineering and System Safety*, 75(2):167–177, February 2002.
15. P.C. Wright, R.E. Fields, and M.D. Harrison. Analyzing human-computer interaction as distributed cognition: the resources model. *Human Computer Interaction*, 15(1):1–42, 2001.