Resilience of Interaction Techniques to Interrupts[⋆]
A Formal Model-based Approach

FULL VERSION

Maurice H. ter Beek[1], Giorgio P. Faconti[1], Mieke Massink[1], Philippe A. Palanque[2], and Marco Winckler[2]

[1] Istituto di Scienza e Tecnologie dell'Informazione 'A. Faedo' (ISTI), CNR
via G. Moruzzi 1, 56124 Pisa, Italy {terbeek,faconti,massink}@isti.cnr.it
[2] Institute of Research in Informatics of Toulouse (IRIT), University Paul Sabatier
118, route de Narbonne, 31062 Toulouse cedex 9, France {palanque,winckler}@irit.fr

Abstract. In many modern working environments interruptions are commonplace as users must temporarily suspend their current task in order to complete an unexpected intervening activity. As users are faced with more and more sources of information competing for users' attention at any time, it is becoming increasingly important to understand how interruptions affect our abilities to complete tasks. The present work introduces a new perspective for the research in the field by employing analytical, model-based, techniques that are informed by well-established cognitive theories and experimental data available in the literature. We propose stochastic modelling and model checking to predict measures of the disruptive effects of interruptions to two well-known interaction techniques: Drag 'n Drop and Speak 'n Drop. The approach also provides a way to compare the resilience of different interaction techniques to the presence of external interruptions that users need to handle. The results obtained are in a form that allows validation with results obtained by empirical studies involving real users.

1 Introduction

In many modern working environments interruptions are commonplace as users must temporarily suspend their current task in order to complete an unexpected intervening activity [28, 74]. In this context, users are required to accomplish multitasking (i.e. two or more tasks performed during the same period of time) and to be able to do so has become a desired skill for many job functions [77].

Interruptions are unpredictable and quite often cannot be disregarded by the user in the working environment. Web page pop-ups, phone calls, emails, instant messaging and social events can also be disruptive when people need to concentrate on certain tasks. One of the interesting aspects of interruptions, according to O'Connaill and Frohlich [62], is that they reveal that the timespace of any individual is not owned and controlled in the same way as their workspace, but can collide and merge with that of another individual unexpectedly.

Research has shown that the level of disruption of different types of interruptions may vary considerably [7, 37]. Quite often, interruptions are associated with negative effects: resuming a task after an interruption is difficult and may take a long time [75], interrupted tasks are perceived as harder than uninterrupted ones, interruptions cause more cognitive workload and they are quite often annoying and frustrating because they keep people from completing their work [8].

Interruptions can also lead to incidents due to human error. According to Trafton & Monk [76], pilots experiencing interruptions during preflight checklists have been blamed for multiple aviation crashes [60, 61]. In addition, recent studies have shown that interruptions may be an important factor in driving [58], emergency room care [21], and nursing errors [78]. Indeed, frequent interruptions can reduce user performance, however not all interruptions bring negative impact:

[⋆] This work has been partially funded by the EU project Resist/Faerus (IST-2006-026764) and by the RSTL project XXL funded by the Italian National Research Council (CNR).

awareness systems such as alarms and alert systems effectively shift our attention to matters that need immediate care [16, 52] and, at least for simple tasks, interruptions may actually increase performance [73].

As users are faced with more and more sources of information competing for user's attention at any time, it is becoming increasingly important to understand how interruptions affect our abilities to complete tasks [19]. Interruptions raise questions of non-exclusive practical and theoretical significance including: How many interruptions occur at work? How performance is affected by various interruption characteristics, such as complexity, duration, timing and frequency? How many are disregarded rather than handled? Who benefits from the ensuing interactions? How disruptive are interruptions to prior tasks? What can be done to mitigate negative disruptive effects?

Most of the current research has tackled these questions by conducting empirical studies with users, either on controlled conditions (i.e. usability labs) or on working environment (e.g. ethnographical studies). The present work introduces a new perspective for the research in this field based on model-based analytical techniques that can be employed during early phases (i.e. specification phases) of the development process of interactive systems in order to investigate potential disruptive effects of interruptions on user performance and the resilience of interaction techniques to such interruptions. The proposed methodology consists of several phases. In the first phase we consider an abstract view of the flow of information between interaction devices, system and user. To facilitate the unambiguous and precise description and modelling at this level, we make use of a specification notation in which the various components (device, system and user) are modelled as syndetic models based on interactors [32, 35]. Interactors can describe the logical and physical components of an interactive system, but by themselves give little direct insight into how a user might or might not be able to use the system. Syndetic models [30, 12] address this problem by expressing the behaviour of computing and cognitive systems within a common framework that supports reasoning about the conjoint system. The *computing* component of a syndetic model is determined by the system being represented, but for the cognitive side there is a range of models to choose from, each emphasising different aspects of human information processing. The approach that we adopt for syndetic modelling in the present work is called Interacting Cognitive Subsystems [10, 11], or ICS, and is summarised in a later section. The objective of this phase is to get insights in the cognitive resources required to perform the task at hand.

The second phase of the methodology uses the results of the analysis of the first phase. In particular, the various mental configurations through which the user passes while performing tasks using an interaction technique are modelled as states of a process algebraic specification. The change of mental configuration can usually be identified by an observable action, such as the end of a selection movement or the start of a movement after a planning period. Such events are modelled as actions in the process algebraic specification. The average period of time during which a particular mental configuration is in place is modelled by the inverse of a rate parameter of an exponential distribution. How this works exactly is explained in more detail in later sections. In this second phase also the processes modelling the system and interrupt generation are specified. The three components (device/system, user and interrupts) are then composed together and their interaction is modelled by the common observable actions on which they synchronise.

The resulting stochastic syndetic model is then used for analysis with a stochastic model checker. In this work we used the PRISM model checker [48] to analyse and predict the disruptive effect of interruptions considering different frequencies of interruption. In particular we illustrated the approach by modelling two well-known interaction techniques for deleting items from a desktop; Drag 'n Drop, in which users select an item and then drag it over the trash visible on the screen and release the mouse button, and Speak 'n Drop, a multi-modal technique in which users select an item by means of a mouse and delete it by pronouncing the word 'delete'. The performance of a user using a particular technique can then be characterised by the number of items the user is able to drop during a fixed period of time. Using stochastic model-checking this measure can be expressed as a cumulative reward property. This property is analysed for the models of the two interaction techniques and for different frequencies of interrupts allowing for a comparison of their resilience to external interrupts.

Outline of the paper. We start by a review of the literature on interruptions in Sect. 2 followed by a brief presentation of the cognitive theory ICS in Sect. 3 that we use to motivate the models of the user aspects that we present in later sections. Sect. 4 introduces the stochastic modelling language used and the relative analysis tool. In Sect. 5 we describe our methodology for obtaining the stochastic models using the Drag 'n Drop interaction technique as a running example. The same method is then applied to the more complex multi-modal Speak 'n Drop interaction technique in Sect. 6. Sect. 7 presents the results of the performance analysis of the models and discusses their differences in resilience to external interrupts for different assumptions on the average number of interrupts that occur. Finally, in Sect. 8 we conclude the paper with some considerations on current work and a discussion of future research on this topic.

2 Task Interruptions

Literature about human interruption addresses this design problem from one the following different perspectives: a) psychology of human interruption [1, 7, 17, 63]; b) technologies for improving the quality of interruption generation [24, 44]; c) HCI methods for brokering interruptions [28, 76, 75, 71, 79]; d) the effects of interruptions in work settings [25, 62]; and e) case studies describing the results of introducing technologies into the workplace in an attempt to improve coordination performance [46, 59, 70].

2.1 The Anatomy of Interruptions

In the interruptions domain, there are relatively few reported task analyses. However, a number of simple task analyses were conducted across several different domains to capture the critical aspects of the tasks. These analyses are described by Trafton et al. [76, 75]. Fig. 1 shows some key features of the interruption process, based on these naturalistic observations.

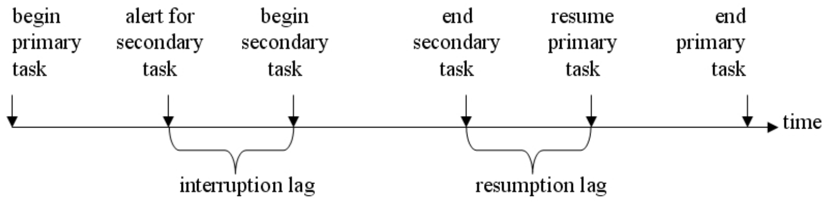

Fig. 1. Timeline: Anatomy of an interruption [75].

A person is working on a primary task, which can be thought of as similar to the complex, long-lasting task that Czerwinski et al. [24] described. Next, an alert for a secondary task occurs. Alerts come in different forms—e.g. a phone ringing, a person coming into the room to ask the person a question, or a fire alarm. During the interruption lag, the person has a moment (or longer) before turning his or her attention to the interrupting task. Then the person starts the secondary task. After the secondary task is completed, the person must resume the primary task. During the resumption lag, the person must figure out what he or she was doing during the primary task and what to do next. Finally, the person resumes the primary task. From this task analysis and the real-world examples, it is clear that different aspects of the cognitive system are relevant to the study of interruptions and resumptions. First, executive control is very important for all interruption/resumption tasks. Second, upon completing the secondary task, the person's main goal is to remember what task he or she was working on and what to do next (though in some real-world situations, new task demands occur or the environment may have changed so that significant re-planning may need to occur). Third, people may or may not use some sort of cue in the environment to actively help them remember what they were doing before the occurrence of the interrupt. Fourth, there may or may not be a link between the primary and secondary tasks. Fifth, in some situations (e.g. an emergency), cues may not have been thought about—there may be relatively different preparatory processes that occur.

2.2 Sources, Types and Taxonomy of Interruptions

Sources of task interruptions can be either external when caused by external entities such as events in the environment or internal when users decide to break the current task flow due, for example to new thoughts coming to their mind. Internal interruptions are very difficult to detect; in some cases they might lead to tasks being abandoned that should be considered normal deviations of the user scenario, for example when the user gives up to reach the initial goal. The sources of external are many and vary, ranging from social events from the environment (e.g. phone calls, instant messaging) to alarms and notification systems.

Types of interruptions that may serve beneficial purposes include warnings and alerts, reminders, notifications and suggestions. Of course warnings and alerts etc., may not always be interruptions if they are part of the normal task flow. More generally, warning and alert etc. should be considered interruptions when they cause a change or disturbance in a person's activity or behaviour. Waliji et al. [79] provide examples from a healthcare context, although these types of interruptions would also exist in other domains.

Warnings & Alerts are usually a sign or signal of some negative event occurring, or a notice to be careful. They are intended to make people aware of an impending danger or difficulty. For example, drug interaction warnings embedded into drug prescribing systems warn doctors and pharmacists about dangerous drug-drug interactions when prescribing or filling a prescription. These warnings are designed to interrupt the current task, and alert the clinician to a potential adverse event. Although such warnings may be critical in preventing errors, it is found that in practice such warnings are often ignored or overridden, suggesting the need for better designed warnings. Warnings and alerts are often urgent and need to be handled quickly. Warnings and alerts may either have an explicit or implicit action associated with them. For example a drug interaction warning may indicate explicitly that there is a potential interaction with a drug and provide a list of medications that may be suitable replacements. An audible alert may be more implicit, simply indicating an off nominal state, without providing any explicit instructions or actions.

Reminders are a form of interruption that causes an individual to remember or recall an event. Decision support systems often remind users of standard tests or procedures that conform to practice guidelines. Although the urgency or importance of reminders may vary, many will include an explicit associated action. For example a medication reminder may announce the time, dose and route for the drug.

Suggestions are ideas or proposals that are propagated to individuals. Patients often receive suggestions and recommendations from their care-givers. For example diabetics are urged to exercise more and eat healthier. Physicians may be informed that their patient may be eligible for a particular clinical trial. Pharmaceutical companies also engage in suggestive practices to prescribers when they promote their particular brand of medication. Such suggestive interruptions can be from face-to-face encounters with a pharmaceutical sales representative or through the use of sponsored drug reference databases. Suggestions are unlikely to be of high urgency or importance. But effective suggestions may explicitly state associated actions that are recommended.

Notifications are usually described as the process of informing. Notifications are defined as the most generic type of interruption, with the least degree of importance or urgency. A notification may be purely informational in purpose with no explicit instruction for action. For example a notice stating the availability of a patient's lab results informs a physician that their requested order is ready. However, notifications may lead to actions implicitly without specific instructions. For example the lab test may indicate that a particular patient needs an immediate surgical procedure.

A few researchers have attempted to define interruptions and establish a taxonomy that describes the different issues surrounding interruptions. McFarlane [54] developed a taxonomy of human interruption as a tool for answering interruptions research questions. The taxonomy, summarised in Table. 1, lists eight dimensions of human interruption. Manipulating each dimension, as discussed in the next section, can influence the disruptive effects of interruptions.

Dimension of Interruption	**Example of Dimension**
Source of interruption	Self; another person; computer
Individual characteristics of person receiving interruption	Limitations of: perceptual processors, cognitive processors, motor processors, memory, focus of consiousness, processing streams; willingness and ability to be interrupted
Method of coordination	Immediate (the person must leave current task to attend to interruption); negotiated (the interruption is announced to the person, and then the person decides when to attend to it); mediated (the interruption is announced to the person's personal digital assistant (or another third party), which determines when the best time is to interrupt the person); scheduled (the person is interrupted during prearranged time only)
Meaning of Interruption	Alert; stop; divert attention (task-switching); distribute attention (task-sharing); remind; communicate information
Method of expression	Physical (i.e. verbal); type (i.e. by purpose)
Channel of conveyance	Face-to-face; mediated by a person; mediated by a machine
Human activity changed by interruption	Conscious or subconscious; individual activities; joint activities
Effect of interruption	Change in activity; change in memory; change in awareness; change in focus of attention; loss of control over activity

Table 1. Summary of McFarlane's [54] taxonomy of human interruption.

Despite the limited number of studies on interruptions taxonomy, there is some evidence of its usefulness to report human errors (due to interruptions). Brixey et al. [16] reports on the use of the taxonomy of task interruption in healthcare systems as a tool to identify how the introduction of a technology might introduce new interruptions contributing to avoid medical error or change the work of clinicians.

2.3 Factors Influencing Disruptiveness

Gillie and Broadbent [37] presented a series of experiments aimed at elucidating features of interruptions that make them more or less disruptive to an ongoing computer task. They manipulated interruption length, similarity to the ongoing task, and the complexity of the interruption. They showed that being able to rehearse one's position in the main task does not protect one from the disruptive effects of an interruption. In addition, they discovered that interruptions with similar content could be quite disruptive even if they are extremely short.

McFarlane [55] examined four methods for deciding when to interrupt someone during multitasked computing. He explored several interruption policies, including immediate (requiring an immediate user response), negotiated (user chooses when to attend), mediated (an intelligent agent might determine when best to interrupt) and scheduled (interruptions come at prearranged time intervals) notifications. It was found that performance was affected by the method used for coordinating interruptions, but there was no one best method for all performance measures. For example, the immediate method showed the worst performance in terms of accuracy, but the best performance in terms of completeness, on the interruption-matching task. The negotiated method showed the best performance in terms of accuracy on the continuous task, whereas the pre-scheduled method showed the worst. Mediation did not appear to significantly improve performance for any measure, although this may be indicative of the type of task performed. McFarlane also cautions that users may postpone attending to interrupting messages in these cases. Also, if forced to acknowledge an interruption immediately, users in his study got the interrupting task done promptly but were less efficient overall.

Other researchers have also studied the timing of interruptions, and how a warning can allow a person to anticipate an interruption [37, 75, 58, 42]. Warnings essentially create an interruption lag (see Fig. 1), and results of these studies have shown that an interruption lag can reduce the disruptive effects of interruptions, primarily by reducing reorientation time to the primary task after the interruption task is completed and thereby reducing overall performance time of the primary task. Interruption lags in these studies allowed participants to either finish what they were working on before attending to the interruption, or encode retrieval cues to allow for better task resumption following the interruption. Most of these studies have focused on computerised work, where an automated computerised system must intermittently interrupt a user for input, while the user is focussed on other tasks. However, it is important to note that in safety-critical environments, such as a hospital, it may not be possible for healthcare workers to anticipate interruptions and have a substantial interruption lag.

2.4 Solutions and Design Support

Although the research on interruptions is still relatively new, and much work still needs to be done at both theoretical and applied levels, there is some evidence on how to make interactive systems more resilient and how to reduce the disruptive effects on user tasks. These strategies include *human training, guidelines for design and tool support.*

Human training. One of the most striking findings in many studies is the effectiveness of training strategies in mitigating the immediate disruptiveness of interruptions (e.g. rehearsal, environmental cues). According to Trafton [75], if people are simply learning the task, training people on the task itself would reduce the disruptiveness of the interruptions. However, if people are learning how to resume, then training on interruptions and resumptions should be built into current training regimens. They found that interruptions became less disruptive over time with experience and practice on the resumption process itself; experience on the primary task alone (without interruptions) did not reduce the disruptiveness of interruptions. This study strongly suggests that for training people in complex domains, training scenarios should include occasional naturalistic interruptions in order to reduce the disruptiveness of interruptions during actual performance.

Some *guidelines* have been proposed upon theoretical research for reducing the detrimental effects of interruptions. For example, within the prospective memory framework, McDaniel et al. [53] found that the use of a blue dot cue could improve performance upon resumption of the task. This suggests that providing an external mnemonic may greatly benefit performance for people who deal with real-world interruptions and prospective memory tasks. Using the Long Term Working Memory (LTWM) perspective, Oulasvirta and Saariluoma [63] also made several applied suggestions. Based on the results of their experiments, they suggested that system designers should keep 'interaction chains' (the number of interface actions that lead to a goal or subgoal) quite short. The amount of time does not seem to be theoretically determined, but 20 s seems to be a heuristic used by some designers. They also suggested preventing interruptions on tasks that require large amounts of encoding time (e.g. certain checklists that airline pilots go through). Finally, they suggested that user control of interruptions is beneficial (consistent with McFarlane [55]) because it allows the person to have control over the encoding time, which is critical under their framework. Finally, Table 2 presents guidelines arising out of the memory for goals theory and the theoretical justifications for why they should be followed.

Some of the design solutions for interruptions concern the development of specialised *tools.* For example, Czerwinski's group at Microsoft Research probably has the best track record of building good tools that are based on theoretically grounded applied research principles. For example, the diary study described previously [25] had a major impact on the creation of a prototype tool called the GroupBar. GroupBar allows people to save and retrieve application and window management setups, which can be extremely useful when switching tasks. Bailey and his colleagues also have built several tools based on empirical work [7]. They suggested that the best place to interrupt people is between 'coars' breakpoints between tasks. They have used an empirical approach to explore the linkages between traditional task analytic approaches (e.g. goals, operators, methods, and selection rules, or GOMS) and pupil size as a measure of mental workload. They have created a tool that is able to automatically detect times of high and low workload. They have suggested that interrupting people at times of low workload is best. They currently have several demonstration systems that perform components of this task.

Suggestion	**Theoretical Reason**
Minimise interruptions	Interruptions are disruptive (whole theory)
Do not set your e-mail to automatically alert you when you get e-mail	Minimize external interruptions
Turn off all 'intelligent' agents that interrupt you	Minimise external interruptions
When you get interrupted, take 2 s to figure out what you will do next	Rehearsal during interruption lag facilitates resumption
Make the next thing you do an action on a visible object	Environmental cues can prime previously suspended goals
Highlight the next thing you want to do upon resumption	Explict, blatant environmental cues help the resumption process

Table 2. Interface guidelines inspired by the memory for Goals Framework (Trafton [76]).

2.5 Formal Modelling of Task Interruptions

There have been several attempts to formalise cognitive models describing the impact of interruptions in the human behaviour [1, 75, 71, 73]. However, only a few have addressed formal description techniques to describe the occurrence of interruptions in system specifications [45].

Many situations involve the occurrence of multitasking, and thus the possibility to get work interrupted any time. In multitasking environments, interruptions should be seen as just a break in the current task execution that causes an (unexpected or intended) deviation of the control flow. This problem is quite well known in the domain of Operational Systems where the occurrence of interruptions during the execution of programs running in parallel presents many similarities with multitasking in human activity. Previous work in the Operational System domain [72] has demonstrated that systematic description of all deviations of the system control flow is almost impossible, as it would lead to an exponential and unpredictable number of states. The unpredictability of interruptions would favour the use of declarative models to describe what should be accomplished by the user system (whatever happens) rather than describing the steps required (i.e. the control flow) to accomplish it [68]. This notwithstanding, there are some situations where the interruption of actual tasks should be considered as part of the user goals as, e.g., to cancel document printing. Indeed, some task model notations, such as Concur Task Tree (CTT) [67], explicitly provide the operator suspend/resume (i.e. '|>') that allow explicitly modelling between tasks as presented by Fig. 2. Similarly, West and Nagy [80] have added theoretical structures to the notation GOMS in order to overcome its limitations for analysing interruptions when task switching is common.

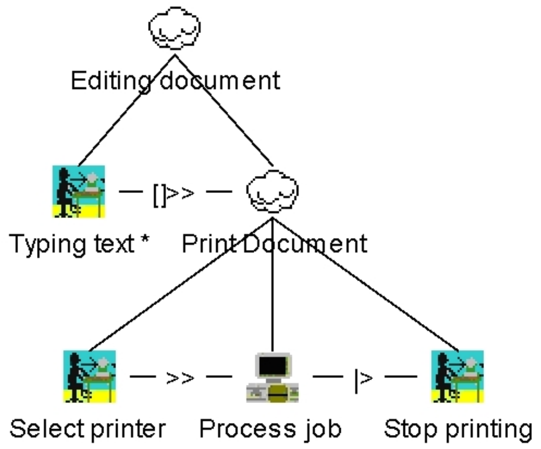

Fig. 2. Example of task modelling with interruption operator '|>' in CTT.

In a completely different approach, Jambon [45] analyses the idiosyncrasies of relationships between tasks (such as parallelism, interlacing and sequence) to derive a formal model (using automata) describing the semantics of interruptions in notations like MAD, UAN and Petri nets. For example, if two tasks are said to run in parallel it is assumed that there will not be any disruptive effect during their execution. If two tasks are interlaced, the disruptive effect should be counted as the effort required for switching tasks. If two tasks are supposed to run in sequence, the interruption of one task could be interpreted as a definite disruption (e.g. starting task $t2$ will interrupt and eventually cancel task $t1$) or an interlacing among tasks with a less disruptive effect on human activity (e.g. starting task $t2$ will interrupt task $t1$, but $t1$ could be resumed after task $t2$ has finished). Additionally, tasks resumption could be done at different steps of the task execution (i.e. restarting from the beginning, resuming the task at the point before the interruption occurred, resuming at the end of the task assuming it has been accomplished). In the stochastic process algebra used in the present paper there is no explicit suspend/resume operator, but its behaviour can be modelled using a combination of the more primitive interleaving and choice operators.

3 Interacting Cognitive Subsystems

In this section we recall the relevant parts of the cognitive theory of Interacting Cognitive Subsystems (ICS) [9] and in particular we show which cognitive configurations are involved in the execution of a simple Drag 'n Drop operation. We also explain the additional configurations involved in a Speak 'n Drop operation and discuss the consequences of cognitive limitations that have to be taken into consideration when developing a model of the user aspects of the use of the interaction techniques.

Interacting Cognitive Subsystems (ICS) [10, 11] is a comprehensive model of human information processing that describes cognition in terms a collection of subsystems that operate on specific mental codes. Although specialised to deal with specific codes, all subsystems have a common architecture, shown in Fig. 3. Incoming data streams arrive at an input array, from which they are copied into an image record representing an unbounded episodic store of all data received by that subsystem. In parallel with the basic copy process, each subsystem also contains transformation processes that convert incoming data into certain other mental codes. This output is passed through a data network to other subsystems. If the incoming data stream is incomplete or unstable, a process can augment it by accessing or buffering the data stream via the image record. However, only one transformation in a given processing configuration can be buffered at any moment. Coherent data streams (see [11]) may be blended at the input array of a subsystem, with the result that a process can 'engage' and transform data streams derived from multiple input sources.

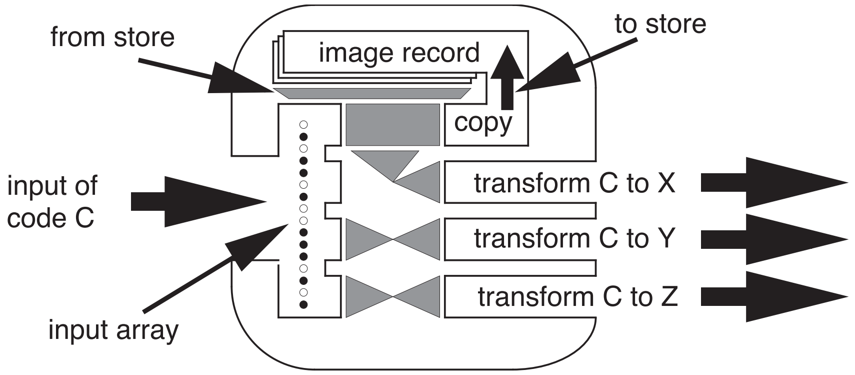

Fig. 3. Generic structure of an ICS subsystem.

ICS assumes the existence of 9 distinct subsystems, each based on the common architecture described above:

Sensory subsystems
VIS visual: hue, contour etc. from the eyes
AC acoustic: pitch, rhythm etc. from the ears
BS body-state: proprioceptive feedback

Structural subsystems
OBJ object: mental imagery, shapes, etc.
MPL morphonolexical: words, lexical forms

Meaning subsystems
PROP propositional: semantic relationships
IMPLIC implicational: holistic meaning

Effector subsystems
ART articulatory: subvocal rehearsal & speech
LIM limb: motion of limbs, eyes, etc

Overall behaviour of the cognitive system is constrained by the possible transformations and by several principles of processing. Visual information for instance cannot be translated directly into propositional code, but must be processed via the object system that addresses spatial structure. Although in principle all processes are continuously trying to generate code, only some of the processes will generate stable output that is relevant to a given task. This collection of processes is called a *configuration.* The thick lines in Fig. 4 shows the configuration of resources deployed while using a hand-controlled input device to operate on some object within a visual scene.

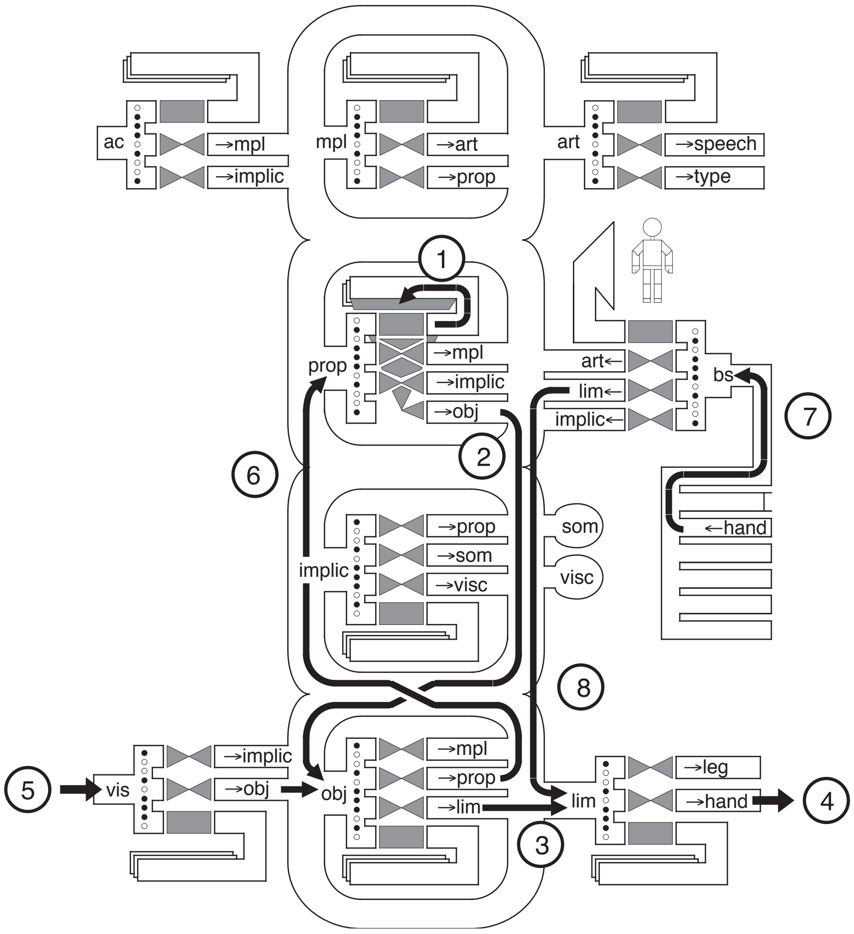

Fig. 4. ICS architecture.

The structures and principles embodied within ICS can be formulated as an axiomatic model in the same way as any other information processing system. This means that the cognitive resources of a user can be expressed in the same framework as the behaviour of the computer-based interface, allowing the models to be integrated directly. We will refer to this way of modelling as *syndetic* modelling. To begin this process, we define some sets to represent those concepts of ICS that will be used here. Here and elsewhere in this document we will make use of common mathematical conventions for sets and relations, for example '$\times$' for cartesian product and '$\mathbb{P}$' for power set.

$[sys]$	- ICS subsystems, e.g. vis, prop, obj etc.
$[repr]$	- mental representententations
$tr == sys \times sys$	- transformation processes, e.g. :vis-obj:.

Representations consist of basic units of information organised into superordinate structures. Coherence of units depends on several issues, including the timing of data streams, that will not be addressed here. Instead, coherence is captured abstractly in the form of an equivalence relation over representations:

$$\_ \approx \_ : repr \leftrightarrow repr$$

In describing ICS it is also useful to discuss the representations that are being delivered as part of a particular data stream. We therefore introduce a further set, *code*, whose elements are representations that have been labelled by the subsystem in which they were generated. Representations from or to the outside world are tagged with '*':

$code == repr \times sys$	- located representations

In general we will write R_{sys} for the code (R, sys), and ':src-dst:' for the transformation (src, dst).

interactor ICS
attributes

sources	:	$tr \rightarrow \mathbb{P}\, tr$
stable	:	$\mathbb{P}\, tr$
$\_@\_$	:	$code \leftrightarrow sys$
coherent	:	$\mathbb{P}\,\mathbb{P}\, tr$
buffered	:	tr
config	:	$\mathbb{P}\, tr$

The state of the ICS interactor captures the data streams involved in processing activities and the properties of the streams such as stability and coherence which define the quality of processing, or in other words, user competence at particular tasks. The sources of data for each transformation is represented by a function 'sources' that takes each transformation 't' to the set of transformations from which 't' is taking input. In general only a subset of transformations are producing stable output, and this set is defined by the attribute 'stable'. The codes that are available for processing at a subsystem are identified by a relation $\_@\_$, where '$c@s$' means that code 'c' is available at subsystem 's'. As not all representations are coherent, only certain subsets of the data streams arriving at a system can be employed by a process to generate stable output. The set 'coherent' contains those groups of transformations whose output in the current state can be blended. If the inputs to a process are coherent but unstable, the process can still generate a stable output by buffering the input flow via the image record and thereby operating on an extended representation. However, only one process in the configuration can be buffered at any time, and this process is identified by the attribute 'buffered'. The configuration itself is defined to be those processes whose output is stable and which are contributing to the current processing activity.

actions

engage	:	$tr \times tr$
disengage	:	$tr \times tr$
buffer	:	tr
trans		

Four actions are addressed in this model. The first two, 'engage' and 'disengage', allow a process to modify the set of streams from which they are taking information, by adding or removing a stream. A process can enter buffered mode via the 'buffer' action. Lastly, the actual processing of information is represented by 'trans', which allows representations at one subsystem to be transferred by processing activity to another subsystem.

The principles of information processing embodied by ICS are expressed as axioms over the model defined above. For a more complete discussion refer to [33].

axioms

1 $\forall\, trs : \mathbb{P}\, tr \bullet trs \in coherent$
$\Leftrightarrow$
$$\exists\, dest : sys \bullet \begin{pmatrix} \forall\, s, t : sys \bullet \text{:s-t:} \in trs \Rightarrow t = dest \\ \wedge \\ \forall\, s, t : sys;\ p, q : repr \bullet \begin{pmatrix} \text{:s-dest:} \in trs \wedge p_s @ dest \\ \wedge \\ \text{:t-dest:} \in trs \wedge q_t @ dest \end{pmatrix} \Rightarrow p \approx q \end{pmatrix}$$

2 $t \in stable \Leftrightarrow sources(t) \in coherent \wedge (t = buffered \vee sources(t) \subseteq stable)$

3 $t \in config \Leftrightarrow (t \in stable \wedge \exists\, s \bullet t \in sources(s))$

4 $\textbf{per}(engage(t, src)) \Rightarrow src \in stable$

5 $$t \notin stable \Rightarrow \begin{pmatrix} \exists\, s \bullet s \in stable \wedge s \notin sources(t) \wedge \textbf{obl}(engage(t, s)) \\ \vee \\ \exists\, s \bullet s \notin stable \wedge s \in sources(t) \wedge \textbf{obl}(disengage(t, s)) \\ \vee \\ \textbf{obl}(buffer(t)) \end{pmatrix}$$

6 $[buffer(t)]\ buffered = t$

7 $sources(t) = S \Rightarrow [engage(t, s)]\ sources(t) = S \cup \{s\}$

8 $sources(t) = S \Rightarrow [disengage(t, s)]\ sources(t) = S - \{s\}$

9 $p_x @ src \wedge \text{:src-dst:} \in stable \Rightarrow [trans]\ p_{src} @ dst$

10 $(\exists\, p : repr;\ src, dst : sys \bullet [trans]\ p_{src} @ dst) \Rightarrow \exists\, x : sys \bullet p_x @ src \wedge \text{:src-dst:} \in stable$

11 $\forall\, s, t : sys \bullet \text{:s-t:}, \text{:t-s:} \in config \wedge (buffered = \text{:s-t:} \vee buffered = \text{:t-s:}) \wedge x : tr \bullet x \notin stable$
$\Rightarrow \textbf{per}(buffer(x))$

Graphical Input Devices The most common and widespread graphical device is the 2D mouse, a physical device equipped with two transducers able to measure the distance between a current position and a next point along two axes and with a number of buttons (usually from one to three). The mouse can be described by a very simple interactor [31, 35], where the type *RelPos* represents *relative* positions, i.e. offsets, and the type *Switch* represents the state {UP, DOWN} of mouse's button.

interactor Mouse
attributes
mouse : *RelPos*
button : *Switch*
actions
$\boxed{\textsf{lim}}$ *operate* : *RelPos*
$\boxed{\textsf{lim}}$ *push*
$\boxed{\textsf{lim}}$ *release*
axioms

1 $[operate(\delta)]\ mouse = \delta$

2 $[]\ button = UP$

3 $[push]\ button = DOWN$

4 $[release]\ button = UP$

5 $[push]\ \textbf{obl}(release)$

6 $[\![operate]\!] \wedge [\![push]\!] \wedge [\![release]\!]$ in $[\![Mouse]\!]$

The $\boxed{\textsf{lim}}$ decoration of the *operate*, *push*, and *release* actions means that the device is sensed by the body-state subsystem when it is used, and the notation $[\![\ldots]\!]$ is used to refer to the perceivable aspect of an attribute, interactor or action. The axioms describe the operations that can be performed. Axiom 1 captures the movement of the mouse by a distance δ, Axiom 2 states that initially the button of the mouse is in position UP. Axiom 3 states that after a push operation on the button of the mouse its position changes into DOWN, and Axiom 4 states that after a release operation the button returns to its UP position. Axiom 5 states that after a push the button will eventually and mandatory be released. And, finally, Axiom 6 states the perceivable operations possible with the mouse.

While the mouse can be used as a pure input device, it is usually coupled with a cursor that provides the feedback of the current position in a reference space (usually a display).

interactor Cursor
attributes
$\boxed{\textsf{vis}}$ $\quad cursor \; : \; DispCoord$
actions
$\quad render$
axioms
1 $\quad [\![cursor]\!]$ in $[\![Cursor]\!]$

The state space of the cursor device is represented by a pair of coordinates within the *absolute* display coordinate system ($DispCoord == xDisplay \times yDisplay$). The $\boxed{\textsf{vis}}$ decoration indicates that the cursor position is visually perceivable, while the axiom states that the cursor must be visible whenever the interactor is activated. Mouse and cursor are linked by composing the two interactors and adding further information describing their mutual relations:

interactor Mouse-Cursor
$\quad Mouse, Cursor$
attributes
$\quad mouseLocation \; : \; DispCoord \times RelPos \rightarrow DispCoord$
axioms
1 $\quad cursor = P \Rightarrow [render]\; cursor = mouseLocation(P, mouse)$
2 $\quad [push]\; \textbf{obl}(render)$
3 $\quad [release]\; \textbf{obl}(render)$

The attribute named ‘mouseLocation’ computes a new cursor position from the previous cursor position and the relative position of the mouse. The axiom states that after the rendering has taken place the new cursor position is the one computed by mouseLocation. In addition, after either a ’push’ or a ’release’ the system is forced to perform a ’render’. With this simple mechanism, relative positions in the mouse space can be reflected to the user in terms of absolute positions in the display space. As we will see this has a number of implications when considering the cognitive resources necessary to operate the device effectively.

Positioning and selecting Mice are used for positioning and selecting. Positioning refers to the task of identifying a specific point in a coordinate system. As an example, positioning in a graphics editor refers to identifying the starting position from where an object is drawn. Similarly, in a text editor it refers to identifying the position at which text is written. Selecting refers to the task of identifying an object in a given context. This can be a graphical object, a menu entry, or a character in a text.

Both positioning and selecting operations are influenced by the same kind of parameters:

Current position: the point or the entity at which the device is currently located;
Target position: the point or the entity that must be identified;
Distance: the distance from the current to the target position;
Size: the area of the target position.

The device position is moved from the current to the target position. When the target is acquired an input token is sent to the interactive system, usually following the firing of a trigger. We already know from experiments [36, 49] that the movement performed with a specific device is influenced by the distance and the size of the target, assuming that no constraints are imposed over the trajectory. Similarly, performance varies across devices due to the different muscles involved in controlling the movement.

The interactors defined in the previous section adequately represent the system requirements and are suitable to be reused in a syndetic specification for the positioning operation. For what concerns selection and delete, a further interactor is required from the system side to express the rendering of the objects currently selected. The selection itself can be triggered either on a button down or on a button up transition. For this reason we specialise two interactors Select-Push and Select-Release to take this possibility into consideration.

interactor Select-push
$\quad Mouse - Cursor$
attributes

$\boxed{\text{vis}}$	*objects*	:	$\mathbb{P}\, Obj$
	objLocation	:	$Obj \rightarrow DispCoord$
$\boxed{\text{vis}}$	*selected*	:	Obj

axioms
1 $\quad selected = s \Rightarrow s \in objects$
2 $\quad [\,]\ selected = \varnothing$
3 $\quad \exists\, o : Obj \bullet o \in objects$
$\quad \wedge\ cursor = P \wedge objLocation(o) = mouseLocation(P, mouse)$
$\quad \Rightarrow [push]\ selected = o$
4 $\quad \forall\, o : Obj \bullet o \in objects \wedge cursor = P \wedge objLocation(o) \neq mouseLocation(P, mouse)$
$\quad \Rightarrow [push]\ selected = \varnothing$
5 $\quad [\![objects]\!] \wedge [\![selected]\!]$ in $[\![Mouse - Select]\!]$

The *selected* object is one of *objects*. Initially, it is empty and nothing is selected. On button down the object, over which the cursor is positioned, becomes selected. Pushing the button outside of objects resets the selection.

interactor Select-release

refers to the case of selection on button up, in which case Axiom 3 is modified accordingly.

axioms
3 $\quad \exists\, o : Obj \bullet o \in objects$
$\quad \wedge\ cursor = P \wedge objLocation(o) = mouseLocation(P, mouse)$
$\quad \Rightarrow [push][release]\ selected = o$

Both the objects and the current selection are visually perceivable.

Removing objects In this paper we are interested in the development of syndetic models that can be used to analyse the resilience of different interaction techniques to the effect of external interrupts. The specific interaction techniques that we address and analyse are the common Drag 'n Drop technique used to remove items from a desktop and an alternative multimedia combination of a mouse and voice commands, Speak 'n Drop, in which a user uses the mouse to select items on a display and pronounces the command *delete* in order to remove the selected item. For each of the techniques, the above system specification is extended accordingly.

Drag 'n Drop. For Drag 'n Drop, the Mouse-Select interactor is augmented with a trash.

interactor Remove-DnD
$\quad Select - push$
attributes

$\boxed{\text{vis}}$	*trash*	:	Obj
	objMove	:	$DispCoord \times RelPos \rightarrow DispCoord$

axioms

1 $trash = t \Leftrightarrow t \notin objects$

2 $button = DOWN \wedge selected = s$
$\wedge\ cursor = P \wedge objLocation(s) = mouseLocation(P, mouse)$
$\Rightarrow$
$[operate(\delta)]\ objLocation(s) = objMove(objLocation(s), mouse)$

3 $button = DOWN \wedge objects = O \wedge selected = s$
$\wedge\ trash = t \wedge cursor = P \wedge objLocation(t) = mouseLocation(P, mouse)$
$\Rightarrow [release]\ objects = O - \{s\} \wedge selected = \varnothing$

4 $[\![trash]\!]$ in $[\![Remove - DnD]\!]$

When the mouse button is pushed down over the selected object, it is moved accordingly to mouse movement. If a 'release' action occurs when the mouse is over the trash then the selected object is removed from the set of objects. It is required that the trash is perceived.

Speak 'n Drop. With this interaction technique, the user selects an icon by clicking on it and deletes it by pronouncing the word *delete*. The Select-release interactor can be reused here for the selection task. A new interactor is required to deal with the speech recogniser.

interactor Record-Speech
attributes
$\boxed{\text{ac}}$ *record* : *Cmd*
actions
$\boxed{\text{art}}$ *speak* : *Cmd*
recognize
axioms

1 $[]\ record = \varnothing$

2 $[speak(\rho)]\ record = \rho \wedge \mathbf{obl}(recognize)$

3 $[\![speak]\!] \wedge [\![record]\!]$ in $[\![Record - Speech]\!]$

When the device is operated it is sensed by the body-state subsystem, $\boxed{\text{art}}$, and the speak action is made perceivable, $\boxed{\text{ac}}$. When a command is pronounced, it is also recorded.

The interaction technique is built by the following interactor:

interactor Remove-SnD
$Select - release$
$Record - Speech$
axioms

1 $objects = O \wedge record = \rho \wedge selected = s$
$\Rightarrow [recognize]\ objects = O - \{s\} \wedge selected = \varnothing \wedge record = \varnothing$

An object is removed when it is selected and a command has been recognised. Here we do not enter into the details of commands. We assume that only one is available (i.e. $\rho = delete$) and that it is always recognised when pronounced.

Syndetic Model The syndetic model of device interaction is created by introducing both the user and system models into a new interactor and then defining the axioms that govern the conjoint behaviour of the two agents. A new attribute (*goals*) is used to 'contextualise' the generic ICS model to the task of selection and removal formation by representing the sequence of points that the user wants to follow in the display space and the clicks. Of course, it is highly unlikely that users will have such a precise mental model of their goals, and a more realistic approach might be to describe a class of desired or acceptable displays. However, it would add little to the analysis.

Drag 'n Drop: Syndesis. The syndetic model for Drag 'n Drop style of interaction is almost straightforward:

interactor User-DnD

Remove-DnD	- the system component: the device interactor
ICS	- the user component: ICS resources and constraints

attributes

$goal \ : \ DispCoord*$

The 'operate', 'push' and 'release' actions defined in the Mouse interactor are driven by the user's limb subsystem, and in order for the user to operate the device, the configuration must be set to transform a propositional representation of the desired display coordinate to be reached by the cursor into musculature control, using the processes illustrated in Fig. 4.

axioms

1 $\mathbf{per}(operate(D)) \Rightarrow [\![cursor]\!] \text{ in } [\![User - DnD]\!]$

2 $\mathbf{per}(operate(D)) \wedge \mathbf{per}(push) \wedge \mathbf{per}(release)$
$\Rightarrow DnD - Config \subseteq config \wedge buffered = \text{:prop-obj:}$

3 $goal = \langle objLocation(s) \rangle \frown \langle objLocation(t) \rangle$
$\Rightarrow$
$[operate(P_s)][push] \ goal = \langle objLocation(t) \rangle [operate(P_t)][release] \ goal = \langle \rangle$

In order to operate the device and select a display coordinate 'D', the operator must be able to perceive the 'current' position of the cursor. Whether or not the position of the cursor reflects all of the actions that the user has carried out is an issue that we will return to later. In addition, the set 'DnD-Config' of transformations which is assumed to contain the processes deployed in Fig. 4, must be part of the configuration; that is

$$\{\text{:prop-obj:}, \text{:vis-obj:}, \text{:obj-prop:}, \text{:obj-lim:}, \text{:bs-lim:}, \text{:lim-hand:}\} \subseteq config,$$

These two requirements are captured in Axiom 1 and Axiom 2. Axiom 3 and Axiom 4 simply state that a user works sequentially through the sequence of points that make up their current goal taking the appropriate sequence of actions.

Drag 'n Drop: Analysis. At the propositional system, a goal is formulated that consists of only one coordinate, namely the propositional representation of the target point or area location(s) that the cursor is to be moved to. In terms of the User-DnD interactor, this is an assumption that in the 'initial' state $goals = \langle location_{PROP} \rangle$. The goal is satisfied when the distance of the cursor or of the digitiser from the location$_{PROP}$ is perceived to be zero (or within some threshold).

If :vis-obj: and :bs-lim: are to be part of the configuration, we must also assume that the input to these sensory systems is stable, and thus after a processing cycle (modelled by the 'trans' action) we have that $cursor_*@VIS$ and $operate_*@BS$.

Since :prop-obj: and :vis-obj: are part of the configuration and are both sources of :obj-prop: and :obj-lim:, they must also be coherent. Provided this condition is met, processes within the object system are permitted to engage the streams and thus blending of the input data streams can occur. After a 'trans' action, the visual information becomes available at the object system, so that the following holds:

$$location_{PROP}@OBJ \wedge cursor_{VIS}@OBJ$$

Using its own encoding of this information, object system processes are able to derive propositional information on the distances between objects, and limb-based code specifying the musculature control needed for the cursor or the digitiser to get closer to the target location within the display space. Let us use Δpos to refer to the difference between the intended position $location_{PROP}@OBJ$ and current cursor position $cursor_{VIS}@OBJ$. The two representations produced by OBJ processes will then be $\Delta pos_{OBJ}@PROP$ (from :obj-prop:) and $\Delta pos_{OBJ}@LIM$ (from :obj-lim:). These will be transferred over a stable stream to the destination systems.

Provided that the new representation Δpos_{OBJ} of the required device movement is coherent with respect to the current goal, PROP can sustain its output toward OBJ in a reciprocal loop.

In fact, if $\mathit{location}_{PROP}$ and Δpos_{OBJ} are coherent, the reciprocal exchange of information will be stable and self-sustaining, and as a result it will not be necessary for :prop-obj: to be buffered.

The limb subsystem always receives an input stream from the body-state system that is responsible of proprioceptive feedback encoding dimensions such as skeletal muscle tensions. According to the definition of the Mouse interactor, the limb system produces an output stream that enables the hand to control the corresponding devices through the 'operate', 'push' and 'release' actions. Consequently, the body-state system receives a proprioceptive feedback from these devices. From this information the system is able to derive information on the direction and on the velocity and the acceleration of the arm/hand movement as perceived in the device spaces, so that we can state that $\Delta dev_{*}@BS$. This information is carried on a stream of data from the :bs-lim: transformation, and will be stable provided that the movement is within expected bounds. Finally, the limb system is permitted to engage the stable streams from the object and body-state systems and after a trans action we will have

$$\Delta pos_{OBJ}@LIM \wedge \Delta dev_{BS}@LIM.$$

Now, for the output of the :lim-hand: transformation to be stable we know that its sources (:obj-lim: and :bs-lim:) must also be both stable and coherent. Since the representations of $\Delta pos_{OBJ}@LIM$ and $\Delta dev_{BS}@LIM$ are actually based in different coordinate spaces (AbsPos and RelPos) in the case of a mouse-like device, we argue that they are not coherent. Under this assumption, the limb system will either (a) try to engage a new stable stream, or (b) try to disengage from one of the two streams, or (c) enter buffered mode. There is no other available stream in ICS for LIM code, so option (a) is eliminated.

Option (b) captures the situation where a user consciously tries to ignore either visual or proprioceptive feedback, possibly by watching the motion of the mouse rather than the cursor. In the case of option (c) buffering is transferred from :prop-obj: to :limb-hand:. However, it can be achieved, as once a stable representation of the desired location has formed in the :prop-obj::obj-prop: loop, the buffer is no longer required and can be used elsewhere. That way, user's goals can be satisfied considering *nominal* user and system performance.

In addition to this analysis we also recall a few facts about pointing movements. It is well known and largely accepted that pointing movements are governed by Fitts' law [36]:

$$MT = a + b \log_2 \left(\frac{ID}{W} + 1 \right) \tag{1}$$

where the logarithmic factor called the index of difficulty ID, describes the difficulty to achieve the pointing task [50]. More recent research has shown that a number of phases can be distinguished in the movement itself [34]:

- an optional planning phase where the display is investigated and the propositional goal is formed;
- a ballistic phase from the current position to close to the target, where the movement is based on low level hand control that does not require focus of attention;
- a phase in which the target is approached slowly, performed under visual control requiring focus of attention;
- an optional adjustment phase in the case the target is not completely reached, also requiring visual control.

The mean time interval for each phase depends on the difficulty of the task. In particular, for a given subject, the duration of the target approaching phase and the adjustment phase depend heavily on the size of the target, whereas the duration of the ballistic phase is mainly depending on the distance that needs to be covered. The duration of the planning phase is relatively short and is constant, which is directly related to the cognitive configuration that needs to be set-up in order to prepare for the task to be performed.

Speak 'n Drop: Syndesis. The syndetic model for pointing and speaking has the following structure:

interactor User-SnD
  Remove-Speech - the system component: the device interactor
  ICS - the user component: ICS resources and constraints
attributes
  $goalSelect \ : \ DispCoord*$
  $goalRemove \ : \ Cmd*$

The 'operate', 'push' and 'release' actions are driven by the user's limb subsystem, and in order for the user to operate the device, the configuration must be set to transform a propositional representation of the desired display coordinate into musculature control. In addition, the 'speak' action is driven by the articulatory subsystem. Consequently, a second configuration is needed for the user to be able to speak.

axioms

1 $\mathbf{per}(operate(D)) \Rightarrow [\![cursor]\!] \text{ in } [\![User - SnD]\!]$

2 $\mathbf{per}(operate(D)) \wedge \mathbf{per}(push) \wedge \mathbf{per}(release) \wedge \mathbf{per}(speak)$
$\Rightarrow Speak - Config \subseteq config \wedge buffered = \text{:prop-obj:}$

3 $$\begin{pmatrix} goalSelect = \langle objLocation(o_i)\rangle \frown G \\ \Rightarrow \\ [operate(P_i)][push][release]\ goalSelect = G \end{pmatrix} \wedge \begin{pmatrix} goalRemove = \langle record(delete)\rangle \frown G \\ \Rightarrow \\ [speak(delete)]\ goalRemove = G \end{pmatrix}$$

In order to operate the devices and select a display coordinate 'D', the operator must be able to perceive the "current" position of the cursor. In addition, the set 'Speak-Config' of transformations which is assumed to contain the processes deployed in Fig. 5, must be part of the configuration; that is

$$\begin{pmatrix} \{\text{:prop-obj:, :vis-obj:, :obj-prop:, :obj-lim:, :bs-lim:, :lim-hand:}\} \\ \cup \\ \{\text{:prop-mpl:, :mpl-prop:, :mpl-art:, :bs-art:, :art-speech:, :ac-mpl:}\} \end{pmatrix} \subseteq config$$

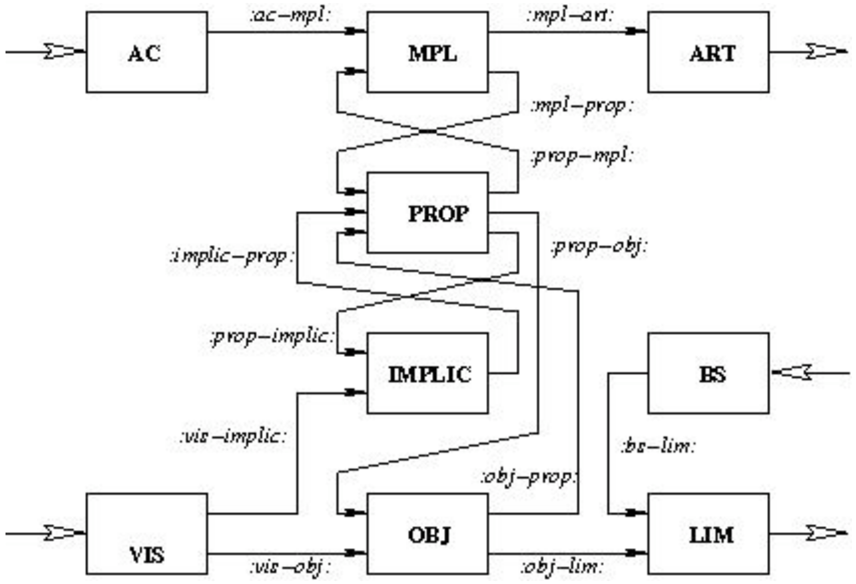

Fig. 5. ICS configuration for the Speak 'n Drop interaction.

Speak 'n Drop: Analysis. The structure of the user's goal at the propositional subsystem is similar to the case of Drag 'n Drop. However, the deployed configuration (Fig. 5) is very different: potentially, it allows both pointing and speaking to be performed in parallel, provided the streams flowing to the same cognitive subsystem are coherent and stable. Clearly, the system might be implemented in order to force the user to achieve the two goals (pointing and speaking) in sequence and the user will adapt to this. However, studies and experiments such as those reported in [57, 64], that have analysed the temporal synchrony between speech and gestures for different languages, show that there exist variations in user behaviour and suggest that systems must be prepared to cope with this variety. In terms of ICS theory, we already know from [33] and from the analysis of Drag 'n Drop that operating any device in a space different from the visual one, under explicit visual control, requires the transfer of the buffer to the limb subsystem since the :bs-limb: and :obj-limb: transformations are not coherent. When the goal is a simple one (e.g. pointing), the operation can be executed smoothly since the goal itself is self-sustaining through the :obj-prop:, :prop-obj: loop. However, in more complex cases of multimodal interaction (e.g. point and speak) transfer of the buffer from the prop subsystem to elsewhere may interfere with the proper execution of operations. In our case, transferring the buffer to the limb subsystem will have the effect of separating the two parallel goals (i.e. *goalSelect* and *goalRemove*) with the incoming streams :obj-prop: and :mpl-prop: contributing to self-sustain only that goal related to them with the consequence that only one will be processed and the other stream be disengaged. The problem here is to determine whether the buffer transfer will occur after both pointing and speaking have started. This is indeed the case and this can be understood as follows. As explained in the Drag 'n Drop case, we know that the first phase of a movement is ballistic and occurs without visual control and attentional focus. Consequently, during the first phase of movement there's no need for buffer transfer (the device is operated with a *natural* gesture, i.e. without technological awareness). Under this condition, a user performing the action 'operate(D)' and starting the action 'speak(delete)' before entering visual control will achieve both goals in parallel. In the case the visual control is entered (i.e. the buffer is tranferred to limb) before speech is started, pointing will always occur first and speaking will be done in parallel with mouse clicking (i.e. push and release) usually terminating later in time, in agreement with the experimental results.

4 PEPA: A Process Algebra for Performance Evaluation

Before continuing with the development of stochastic syndetic models that are amenable to performance analysis we first briefly introduce the formalism and the analysis tool that we will use.

The specification language that we use is the Performance Evaluation Process Algebra PEPA. In PEPA, systems can be described as interactions of components that may engage in activities in much the same way as in other process algebras. Components reflect the behaviour of relevant parts of the system, while activities capture the actions that the components perform. A component may itself be composed of components. The specification of a PEPA activity consists of a pair *(action type, rate)* in which *action type* denotes the type of the action, while *rate* characterises the negative exponential distribution of the activity duration. A positive real-valued random variable X is exponentially distributed with rate r if the probability of X being at most t, i.e. $Prob\{X \leq t\}$, is $1 - e^{r \cdot t}$ if $t \geq 0$ and is 0 otherwise, where t is a real number. The expected value of X is $1/r$. Exponentially distributed random variables are more tractable because they enjoy the memoryless property, i.e. $Prob\{X > t + t' \mid X > t'\} = Prob\{X > t\}$ for $t, t' \geq 0$. Exponential distributions are widely used in the modelling of the dependability and performance of real systems where they form the basis for *Continuous Time Markov Chains* (CTMC), see e.g. [47].

Moreover, proper compositions of exponential distributions can be used to approximate any non-negative distribution. The PEPA expressions used in this paper have the following syntax:

$$\boxed{P ::= (\alpha, r).P \;\Big|\; P \;+\; P \;\Big|\; P \bowtie_L P \;\Big|\; A}$$

Behavioural expressions are constructed through prefixing. Component $(a, r).P$ carries out activity (a, r), with action type a and duration Δt determined by rate r. The average duration is given by $1/r$. It is defined that Δt is an exponentially distributed random variable with rate r. After performing the activity, the component behaves as P. Component $P + Q$ models a system that may behave either as P or as Q, representing a race condition between components. The cooperation operator $P \bowtie_L Q$ defines the set of action types L on which components P and Q must synchronise (or cooperate); both components proceed independently with any activity not occurring in L. The expected duration of a cooperation of activities a belonging to L is a function of the expected durations of the corresponding activities in the components. Typically, it corresponds to the longest one (see [41] for definition of PEPA). An important special case is the situation where one component is *passive* (a rate $\top$ indicates this) in relation to another component. Here the total rate is determined by that of the active component only. The behaviour of process variable A is that of P, provided that a defining equation $A = P$ is available for A. We introduce two shorthand notations. If the set L is empty $P \bowtie_L Q$ is written as the parallel composition of P and Q: $P \mid Q$.

PEPA specifications can be analysed in several ways, but in particular with the stochastic model checker PRISM [48]. With PRISM temporal logic properties involving performance aspects of the system can be formulated as Continuous Stochastic Logic (CSL) formulae and automatically verified. We briefly present CSL and PRISM in the following sections.

4.1 Stochastic Logic: CSL

The properties that one wants to verify by stochastic model checking are usually expressed in some form of extended temporal logic. In this paper, we use Continuous Stochastic Logic (CSL) [4, 5], which is a stochastic variant of the well-known Computational Tree Logic (CTL) [22]. CTL allows one to state properties over states as well as over paths.

CSL extends CTL by two probabilistic operators that refer to the steady-state and transient behaviour of the system being studied. The steady-state operator refers to the probability to reside in a particular state or set of states (specified by a state formula) in the long run. The transient operator allows one to refer to the probability mass of the set of paths in the CTMC that satisfy a given (path) property. In order to express the time span of a certain path, the path operators until ($\mathcal{U}$) and next (X) are extended with a parameter specifying a time interval. Let I be an interval on the real line, p a probability value and $\bowtie$ a comparison operator, i.e. $\bowtie \in \{<, \leq, \geq, >\}$. The syntax of CSL is:

State formulae

$$\Phi ::= a \;\Big|\; \neg\,\Phi \;\Big|\; \Phi \vee \Phi \;\Big|\; \mathcal{S}_{\bowtie p}\,(\Phi) \;\Big|\; \mathcal{P}_{\bowtie p}\,(\varphi)$$

$\mathcal{S}_{\bowtie p}\,(\Phi)$: probability that Φ holds in steady state is $\bowtie p$
$\mathcal{P}_{\bowtie p}\,(\varphi)$: probability that a path fulfills φ is $\bowtie p$

Path formulae

$$\varphi ::= X^I\,\Phi \;\Big|\; \Phi\,\mathcal{U}^I\,\Phi$$

$X^I\,\Phi$: next state is reached at time $t \in I$ and fulfills Φ
$\Phi\,\mathcal{U}^I\,\Psi$: Φ holds along path until Ψ holds at $t \in I$

The meaning of atomic propositions (a), negation ($\neg$) and disjunction ($\vee$) is standard. Using these operators, other boolean operators like conjunction ($\wedge$), implication ($\Rightarrow$), true (TRUE) and false (FALSE), etc., can be defined in the usual way. The state formula $\mathcal{S}_{\bowtie p}\,(\Phi)$ asserts that the steady-state probability for the set of states satisfying Φ, the Φ-states, meets the bound $\bowtie p$. $\mathcal{P}_{\bowtie p}\,(\varphi)$ asserts that the probability measure of the set of paths satisfying φ meets the bound $\bowtie p$. The operator $\mathcal{P}_{\bowtie p}\,(.)$ replaces the usual CTL path quantifiers $\exists$ and $\forall$. In CSL, the formula $\mathcal{P}_{\geq 1}\,(\varphi)$ holds if *almost all* paths satisfy φ. Moreover, clearly $\exists\ \varphi$ holds whenever $\mathcal{P}_{>0}\,(\varphi)$ holds.

The formula $\Phi\,\mathcal{U}^I\,\Psi$ is satisfied by a path if Ψ holds at time $t \in I$ and at every preceding state on the path, if any, Φ holds. In CSL, temporal operators like $\Diamond$, $\Box$ and their real-time variants $\Diamond^I$ or $\Box^I$ can be derived, e.g. $\mathcal{P}_{\bowtie p}\,(\Diamond^I\,\Phi) = \mathcal{P}_{\bowtie p}\,(\text{TRUE}\ \mathcal{U}^I\ \Phi)$ and $\mathcal{P}_{\geq p}\,(\Box^I\,\Phi) = \mathcal{P}_{<1-p}\,(\Diamond^I\,\neg\Phi)$. The untimed next and until operators are obtained by $X\,\Phi = X^I\,\Phi$ and $\Phi_1\,\mathcal{U}\,\Phi_2 = \Phi_1\ \mathcal{U}^I\ \Phi_2$ for

$I = [0, \infty)$. In a variant of CSL the probability p can be replaced by a question mark, denoting that one is looking for the value of the probability rather than verifying whether the obtained probability respects certain bounds.

In addition to CSL formulae, stochastic model-checking algorithms have been extended to deal with reward-based measures. The CSL state formulae have been extended with formulae such as:

Reward formulae

$$\Phi ::= R_{\bowtie r}[F\Phi] \;\Big|\; R_{\bowtie r}[C \leq t] \;\Big|\; R_{\bowtie r}[I = t] \;\Big|\; R_{\bowtie r}[S]$$

$R_{\bowtie r}[F\Phi]$: Reachability reward Φ is bounded by r
$R_{\bowtie r}[C \leq t]$: Cumulative reward within time t is bounded by r
$R_{\bowtie r}[I = t]$: Instantaneous reward at time t is bounded by r
$R_{\bowtie r}[S]$: Steady-state reward is bounded by r

Reachability reward properties refer to the reward accumulated along a path until a certain state is reached which satisfies state formula Φ. Cumulative reward properties associate a reward with each path of a model until t time units have elapsed. Instantaneous rewards refer to the rewards at a particular instant in time. Steady-state rewards refer to the rewards in the long run. In this paper we will use mainly the cumulative reward that allows us to verify for example the number of items a user has been able to remove from the screen in the presence of a number of interrupts for a particular interaction technique.

The reward formulae make implicitly use of so-called reward-structures which have to be included in the formal model specification and which define for example which transitions generate a certain amount of reward when executed. We address this in more detail in the section that describes the models.

4.2 Stochastic Model Checker: PRISM

The Probabilistic Symbolic Model Checker PRISM [48, 66] is a prototype tool that supports, among others, the verification of CSL properties and Reward properties over CTMCs. It accepts system descriptions in different specification languages, among which PEPA and the PRISM language—a simple state-based language based on the Reactive Modules formalism of Alur and Henzinger [3], from which it automatically generates CTMCs. PRISM checks the validity of CSL properties for given states in the given model and provides feedback on the calculated probabilities of such states where appropriate. It uses symbolic data structures (i.e. variants of Binary Decision Diagrams).

5 Methodology

Syndetic modelling is a conceptual breakthrough in interactive system and man-machine interface design [29]. Syndesis provides design, and formal specification and verification techniques that take into account both human's capabilities and limitations together with robustness of interactive systems, thus enabling the study of the joint man-machine behaviour. Such joint models allow for the investigation of properties expressing requirements or expectations and provide insight in the extent to which an interactive system meets such requirements and constraints. The point of departure from known design methods is the requirement that the system should be usable. This is not just a mechanical property of the system, but a statement that implicitly or explicitly must embody some claim or understanding about human capabilities and limitations. In other words, in addition to being a (formally) provable consequence of the specification, the property must also be psychologically valid.

With a syndetic approach capabilities of and constraints on user behaviour are expressed explicitly by representing a cognitive model (or an approximation of such) as a formal theory, which can then be integrated or combined with the model of system behaviour. In this way, the user model is explicit and it contains the theoretical basis for the claims that are based on it; the model is as correct as the theory that it encapsulates. However, properties can be expressed and verified only at the level of abstraction at which available models describe cognitive behaviour. Such models could be refined but at the risk of making unvalidated assumptions.

To deal with this issue, we retain the approach of modelling explicitly aspects of user behaviour; however, we represent directly a cognitive theory only when it matches the level of abstraction at which properties are to be verified. When this is not the case, we model both systems' and aspects of users' behaviour by proposing that, given a set of statistical assumptions about both system and user performance, we can use stochastic modelling techniques to understand the character of the interaction between user and system. We can see these specifications as a means to make explicit the assumptions about the capabilities of both user and system, and of exploring the behaviour of the combination of system and user on the basis of these assumptions. In this way, answers to design questions can be both easier to relate to empirical performance data from human factors and usability studies, and also the results of the analysis can be more meaningful for interpretation by human factors experts. Additionally, much modern and emerging user interface technology is stochastic in nature, which provides additional motivation for the application of stochastic techniques to modelling interaction.

5.1 Modelling the "Drag 'n Drop" Interaction

In order to illustrate the methodology, as applied for the purposes of the FAERUS project, we introduce first the general assumptions we have made and subsequently the steps necessary to develop a stochastic model for "Drag 'n Drop" interaction specified and analysed from a cognitive point of view in Sect. 3.

In this paper we are interested in developing a model-based approach that can be used to analyse the resilience of interaction techniques to external interrupts. In order to study this we include in the model only correct behaviour both of that of the user and that of the system. In principle the model could be enriched with erroneous behaviour, which would be more realistic. However, such an approach would complicate the interpretation of the analysis results because it is then no longer clear to which extent a certain outcome should be attributed to erroneous behaviour or to the effect of interrupts or both. So we prefer to separate these concerns for the time being.

A further concern is the level of abstraction used for modelling. We choose to keep the models relatively abstract, modelling observable events and refine the models only when we find a clear indication that this would lead to significantly better approximations.

Also our choice to use exponential distributions to approximate the average duration of activities should perhaps be motivated briefly. Exponential distributions are fully determined by their only parameter, i.e. the rate of the distribution, or in other words, the inverse of the average time. This means that when using exponential distributions we make minimal assumptions on the exact shape of the distribution of the duration of individual actions. On the other hand, it is also well-known that any distribution can be approximated by a combination of exponential distributions, the so-called phase-type distributions. This gives us an opportunity to refine the models when more precise information on the probability distribution is available, for example from experiments published in the literature, or when we obtain more detailed insights when validating the models. So, our approach is to make minimal assumptions for the time being and enrich or refine the models when we obtain clear indications that this leads to better predictions of reality.

Displays are common devices for the communication in man-machine interfaces; they are used to present visual scenes to users and operators of computers. It is well known from psychological theories [38] that design and layout of objects in a visual scene, as well as other (multi-)media structures, play a fundamental role in the way people perceive, think and react to sensorial stimuli. However, this level of detail is beyond the scope of our current work. We assume that the symbolic configuration or pattern of icons is based on a set of features that directs the structuring of the visual scene into one group of icons with one distinguished icon (the trash in our case) becoming the initial focus of attention, as shown in Fig. 6.

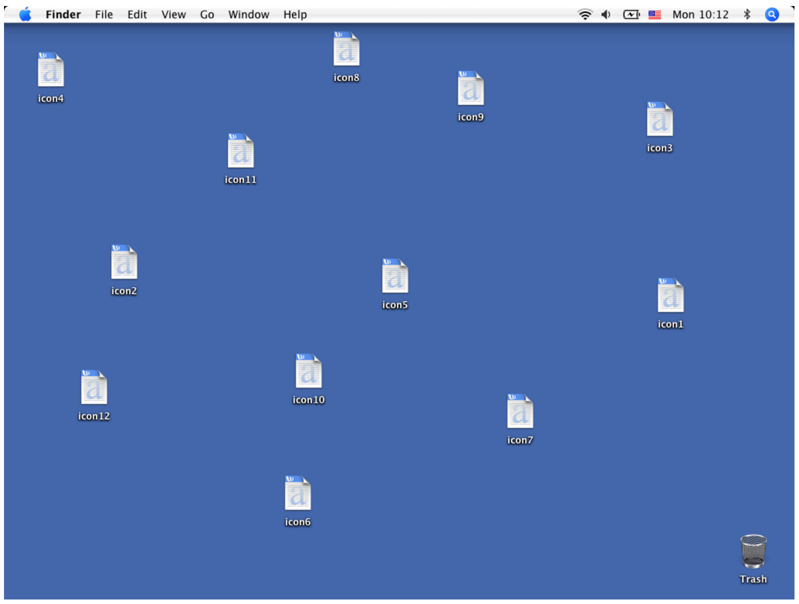

Fig. 6. Example of display.

Features, such as colours, textures and edges, contribute to form a mental *visual representation* derived from raw sensorial data acquired by the eyes. The structure of the visual scene in terms of icons is a more abstract *object representation* obtained by combining the visual representation with the knowledge and the experience the perceiver has of the world. This knowledge comes from another level of abstraction where objects are named and their properties identified, in terms of a *propositional representation*. Thus, the current structuring of the visual scene is used to produce/augment the propositional knowledge about the objects that are being sensed. Finally, sensorial raw data and propositional representation can be combined to produce a further level called *implicational representation* where the general meaning of information is stated. Combining the implicational and the propositional knowledge, people are able to define goals and to (re-)act accordingly. In the Drag 'n Drop example, this is thinking mentally of moving the cursor over one of the icons, push the mouse button to select it, drag it over the trash and release the button. That way, the object representation enriched by propositional and implicational knowledge can be transformed into *limb representation* controlling physical actions performed as an example by hands and eyes. A continuous source of sensorial information, *body state representation*, provides feedback to the co-ordination of the physical actions.

Referring to the ICS theory, the above method is described by a set of subsystems, where knowledge is stored as representations at different levels of abstraction, a set of processes encoding and transforming the knowledge from one representation into another, and a set of communication paths carrying the information from one subsystem to another. The set of transformations, in place at a given moment in time, characterises completely the mental activity and is referred to as a *configuration* (see Fig. 7). The *reaction time* is the interval from the acquisition of sensorial data to the production of physical actions. It represents the time needed to put in place the appropriate mental configuration in order to react to sensorial data.

The level at which mental activity is described is very abstract, dealing only with generic subsystem's structure and communication. What we obtain is an actual mental configuration and the number of steps needed to put it in place. This is very useful information since it provides us with a lower bound for the reaction time: in fact, it is known from psychology literature that each transformation between representations takes approximately 40 ms on average. Consequently, the time spent to deploy the configuration required for Drag 'n Drop is at least 240 ms (with variations among people depending on their biology, knowledge, experience, emotional status, etc.). This time is represented, in the PEPA model, with a value added to any initial move operation performed

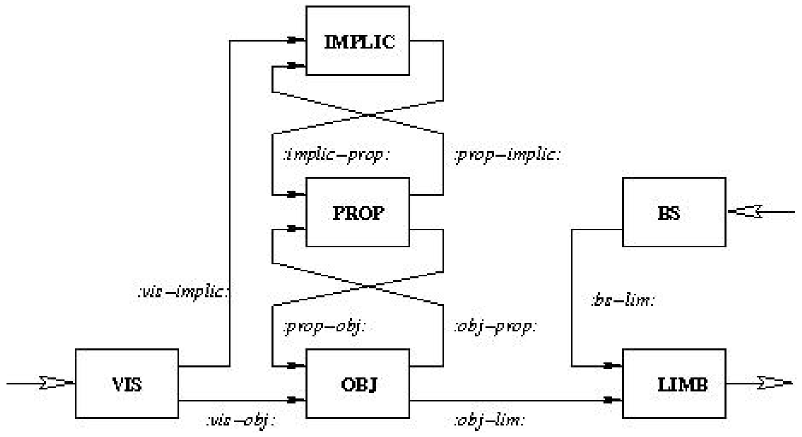

Fig. 7. ICS configuration for graphical interaction.

during interaction; we will refer to it as the *planning phase (UsrPlanMove)* state. Also, the 240 ms planning refers to a skilled performer since, under this assumption, the user is able to start the operation after only one cycle of transformations.

After the planning phase, a user is able to start to move towards the icon to be selected. ICS does not provide information on the user's performance. However, one may derive from the theory that in order to enable the pointing to an icon by using a mouse-like device, the *limb* subsystem needs to be buffered. Consequently, we know that such a pointing movement cannot be performed in parallel with another activity requiring also the transfer of the buffer to a different ICS subsystem.

In order to get better insight in the user's performance in pointing movements, we need to refine the ICS model by addressing human factors and usability studies. It is well known that pointing movements are fairly well approximated by Equation (1) of Sect. 3 (i.e. Fitts' law [36]).

Also, the index of difficulty for a number of devices has been defined from experiments. These experiments show that the average time spent to point at an icon on a computer display by operating a mouse is in the order of 1000 ms [50]. Additionally, more recent research [34] has shown that several phases can be distinguished in the movement itself including a *planning phase*, where the display is investigated and the propositional goal is formed, a *ballistic phase*, where the movement is based on low-level hand control (i.e. without buffering taking place at limb subsystem), an *approach phase*, performed under *visual control* requiring focus of attention (i.e. with buffering taking place at the limb subsystem), and an *adjustment phase* to check that the target has been reached, similarly requiring the focus of attention.

Experiments have indicated an average value of 968 ms for random pointing at icons on a 17" computer display. The distribution of time over the *ballistic phase* and the *approach* and *adjustment phases* varies linearly depending on the distance between the icon and the initial cursor position, with average values of 69% for the ballistic phase and 31% for the visually-controlled one.

With this information, it is possible to refine the ICS analysis by splitting the pointing movement into two distinct phases; the first one includes the planning phase plus the ballistic movement, and the second one consists of visual control. With this refinement, we derive that the transfer of the buffer, described by ICS, occurs later with respect to the start of the movement and this will have important effects in the case of multimodal interaction, as we will see later on.

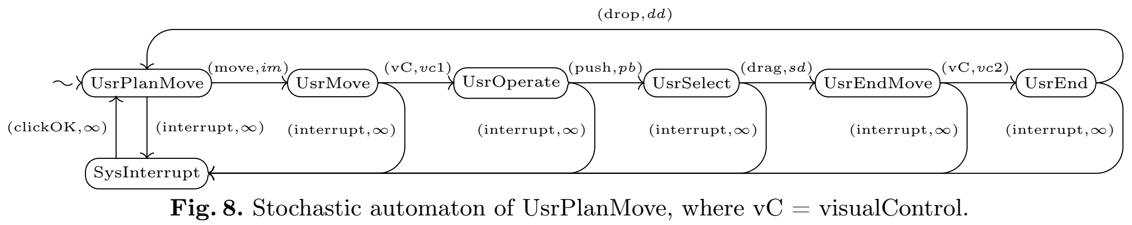

Fig. 8. Stochastic automaton of UsrPlanMove, where vC = visualControl.

Following the above line of reasoning and referring to Fig. 8, the PEPA model describes the pointing movement through the states *UsrPlanMove*, where the user is defining the goal, *UsrMove*, where the user ends the ballistic phase and implements the visual control, and *UsrOperate*, where the user reaches the target and is ready to select it. The transition from state *UsrPlanMove* to *UsrMove* is governed by action *move*, which has an average duration of $240 + 670$ ms expressed as rate *im*, and the subsequent transition to *UsrOperate* is governed by action *vC*, which has an average duration of 290 ms, expressed as rate $vc1$ (in number per 1000 ms), with

$$im = 1000/(planning_t + ballistic_t) = 1.0989$$
$$vc1 = 1000/(approach_t + adjust_t) = 3.4483$$

where $planning_t = 240$ ms, $ballistic_t = 670$ ms and $approach_t + adjust_t = 290$ ms.

In state *UsrOperate* the user can push the mouse button to select the icon. The model performs the transition to state *UsrSelect* by means of action *push* occurring at rate *pb*. With reference to the ICS theory, this rate is set to

$$pb = 1000/push_t = 8.33$$

where $push_t = 120$ ms.

In fact, the buffer is released by the limb subsystem as soon as the target icon has been reached. Then the overall goal is restored and the button push can be performed in a procedural way.

Similarly, the ballistic phase of dragging the selected icon can be performed in a procedural way with the goal being sustained by procedural knowledge without planning, until visual control is enforced again to approach and adjust the selected icon over the trash. We understand that the approach and adjustment time for the visually controlled phase of dragging is shorter than the time required for selection. This is due to the fact that the target of drag (i.e. the trash) is in a fixed position on the display and the performance of pointing at it can be optimised by frequent recurrence of the movement. This movement is specified as a first transition from state *UsrSelect* to *UsrEndMove* governed by action *drag* occurring at rate *sd*, followed by a transition to *UsrEnd* governed by action *vC* occurring at rate $vc2$, with

$$sd = 1000/ballistic_t = 1.49$$
$$vc2 = 1000/(approach_t + adjust_t) = 8.33$$

where $ballistic_t = 670$ ms and $approach_t + adjust_t = 290$ ms.

Finally, the user can release the mouse button dropping the selected icon into the trash and causing its removal from the display. This is specified in the model by the transition from state *UsrEnd* back to the initial state, governed by action *drop* that is performed procedurally and is set to occur at rate

$$dd = 1000/release_t = 8.33$$

where $release_t = 120$ ms.

The model developed so far describes the behaviour of a skilled user performing nominally without making errors or mistakes. This is done in order to precisely understand the effect of interruptions only that occur during the performance of a Drag 'n Drop activity.

The ICS theory can be used to describe which changes in mental configurations occur when unexpected events happen during a users' activity aiming at satisfying a specific goal. The mechanism put in action according to the ICS architecture is the stopping of the activity of effector subsystems and the deployment of the configuration described in Fig. 9.

The sensorial and propositional information is blended at the *implicational subsystem* and fed back into the *propositional subsystem*. This loop with information mutually exchanged between implicational and propositional subsystems, enables to get insights on what we know both as facts and feelings and to reason about the current context in which the unexpected event has occurred.

Again, the level of description dealt with in ICS is very abstract. However, we are not refining it since we are not interested in describing why and how interrupts are handled; here, we are

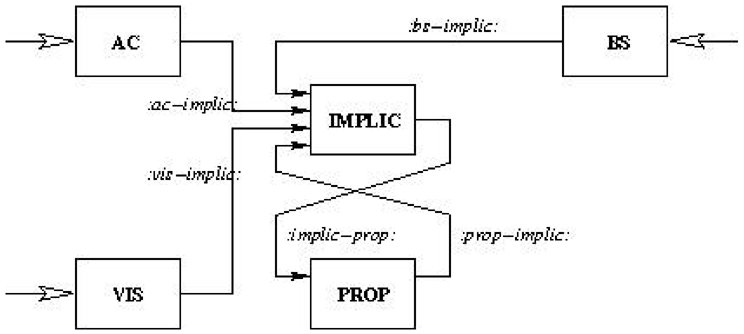

Fig. 9. ICS recovery configuration for unexpected events.

merely interested in observing the effects on the interrupted task. For this purpose, we assume that the interrupt manifests itself as a pop-up window completely covering the display. The user has to click the mouse button over a push button positioned in the center of the window in order to make the pop-up window disappear and be able to resume the previous task.

Interruption is modelled by adding to the PEPA specification a further state, *SysInterrupt*, representing the handling of the interrupt by the user. One extra transition to the new state is added to each state of the Drag 'n Drop model, which is governed by the *interrupt* action occurring with ∞ rate. This reflects the fact that the action is generated outside the user model, but that the user is always ready to deal with it. In other words, it models that the user cannot prevent the interrupt from happening and is forced to deal with it whenever it occurs. Action *clickOK* represents that the interrupt has been serviced. Also this action occurs with rate ∞ in the user model being generated externally by the process in charge of interrupt handling.

The PEPA model just described is a user specification derived from ICS theory augmented with known facts from other branches of cognition studies. It has a general validity within the bounds of the theory it was derived from and can thus be applied wherever this theory applies.

In order to investigate the performance of such a user model we couple it with a specification of the system supporting the Drag 'n Drop task. This specification is represented by the stochastic automaton of Fig. 10. The system is always ready to reply to user's initiated actions as described as well as to interrupts.

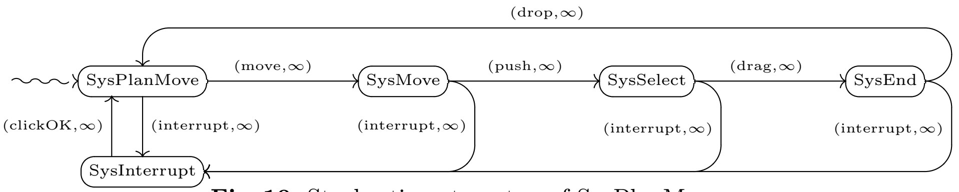

Fig. 10. Stochastic automaton of SysPlanMove.

Generation and handling of interrupts is described separately by a third process, represented in Fig. 11. It is composed of two states, *Interrupt* and *InterruptOK* and two actions governing the transition between them. We assume that no nested interrupts occur, i.e. no new interrupt occurs while a previous one is still being handled. Action *interrupt* represents interrupting events occurring at rate *in*. Action *clickOK* represents that the interrupt has been handled with rate *ok*.

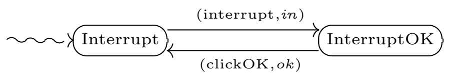

Fig. 11. Stochastic automaton of Interrupt.

We assume that the service rate is

$$ok = 1000/service_t = 0.77$$

where $service_t = 1300$ ms, which is based on the assumption made about the kind of interrupt and the user actions required to react.

When the three systems identified by their initial states *UsrPlanMove*, *SysPlanMove*, and *Interrupt* are composed in parallel, model checking and simulation tools such as PRISM [48] can be used to verify properties and analyse the behaviour and the performance of the conjoint system as with syndetic modelling. The composition is formally defined as:

$$(\text{UsrPlanMove} \bowtie_{\{\text{move,push,drag,drop,interrupt,clickOK}\}} (\text{SysPlanMove} \bowtie_{\{\text{interrupt,clickOK}\}} \text{Interrupt}))$$

The full textual PEPA specifications used for analysis can be found in Appendix A. Finally, we need to add a few additional reward structures to the PRISM version of this specification to make it ready for the particular analysis of the cumulative reward properties by means of the model checker. The transformation of the PEPA specification into the PRISM language is performed automatically by a front-end PEPA to PRISM compiler. Examples of the reward structures can be found in Appendix C. They only serve to define which observable actions we want to count over a certain period of time (e.g. the number of drops and the number of interrupts that occur).

6 Modelling the "Speak 'n Drop" Interaction

We introduce the steps necessary to develop a stochastic model for the "Speak 'n Drop" interaction. We follow the same methodology as used for "Drag 'n Drop" in Sect. 5.1.

We retain the general assumptions, already made there, together with the symbolic configuration of icons that make up the presentation on the display (Fig. 6). Consequently, the initial mental activity taking place to deploy the appropriate mental configuration does not change with respect to the one already explained in Sect. 5.1. However, a different goal is formulated that directs the deployment of the configuration shown in Fig. 12 when the *implicational* and the *propositional* representations are combined together.

In the Speak 'n Drop technique, this involves thinking mentally of moving the cursor over one of the visible icons on the display followed by a mouse click, while pronouncing the word *delete*. That way, the *object* representation enriched with propositional knowledge can be transformed into the *limb* representation that is blended with the *body state* representation to provide for coordinated physical action control. At the same time, the propositional knowledge is transformed into a *morphonolexical* representation that describes the structure of phrases (in our case just the word *delete*). This is subsequently transformed into an *articulatory* representation controlling the physical production of speech.

Usually, skilled people may control their speech sustained by the morphonolexical representation enriched with both *acoustic* and propositional knowledge without explicit blending of body state information. For this reason, the communication between the body state to the articulatory subsystems is not explicitly shown in Fig. 12. Similarly, the communication from the body state and the acoustic subsystems to the implicational one are not shown since they do not play a direct role in the formulation of the users' goal at the propositional level. However, they are always active when recovering from unexpected events as shown in Fig. 9.

This interaction is multimodal in nature requiring both speech and gesture at the same time in order to enable the performance of deictic references. The meaning of words (i.e. *delete*) is resolved by the selection of the icon the user is referring to, as described for example in [14].

Since the performance of deixis requires the combination of data from multiple streams under temporal constraints, a key question is whether users will be able to deploy the resources of the interface to achieve their tasks by performing a deixis or by using a sequential construct. For example, in [30] it is shown that a user is not able to articulate a phrase at the same time as they try to locate textual information on some part of the display.

A possible solution to this problem is that the system might be implemented in order to enforce explicitly that the users perform their tasks sequentially thus requiring them to adapt their behaviour accordingly. However, studies and experiments [57, 64] show that there exist variations

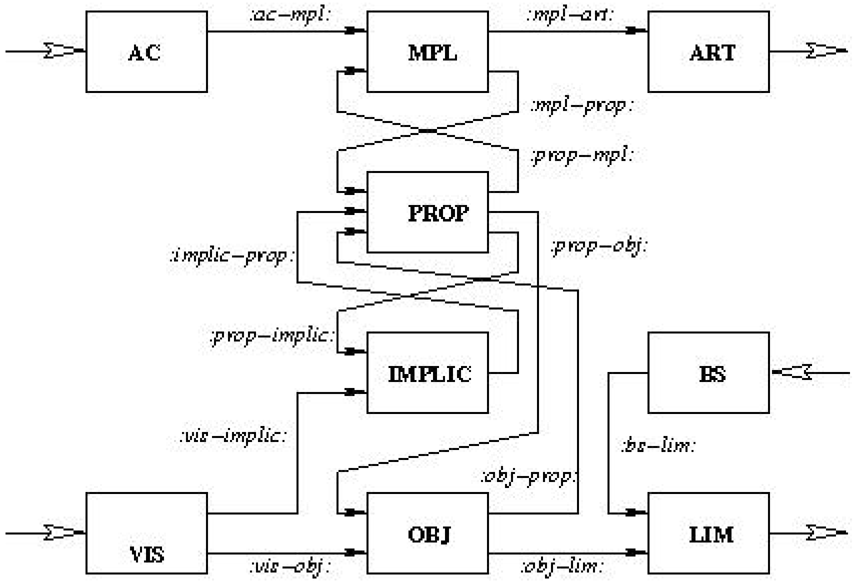

Fig. 12. ICS configuration for the pointing and speaking interaction.

in user behaviour and suggest that systems must be prepared to cope with this variety. Following this suggestion, we specify the system as composed of two processes running in parallel, each one handling one modal interaction independently.

The first system component supports the selection task and it is represented by the stochastic automaton of Fig. 13. This component is always ready to reply to user's initiated actions as well as to interrupts.

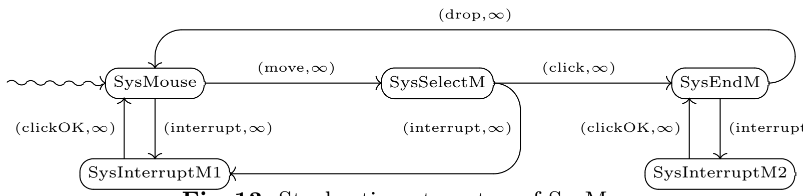

Fig. 13. Stochastic automaton of SysMouse.

The second system component supports the speech action and is represented by the stochastic automaton of Fig. 14. This component is ready to record phrases pronounced by the user (action *startSpeak*) and, subsequently, to recognise them (action *endSpeak*). In addition, it is always ready to reply to interrupts.

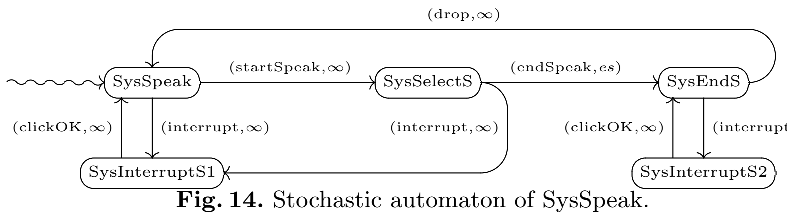

Fig. 14. Stochastic automaton of SysSpeak.

The SysSpeak specification imposes a strong constraint on users' behaviour concerning the performance of the speech modality. This is unavoidable and depends on the currently available technology for speech recognition systems.

In fact, the performance of a speech recognition system is affected by many parameters, such as the quality of the speech signals, any corrupting environmental influences, the complexity of the underlying task, and the memory and computational resources available. Moreover, there is general agreement on the fact that performance accuracy at a rate below 95% is not acceptable, but this again may be syntax and/or domain specific. Although a few speech recognition systems work in real time, their word error rate does not match the agreed performance accuracy [43]. Realistic systems operate at a rate that is in the range of $2/2.5 \times$ *real time*. However, for isolated word recognition systems, such as in the current example, performance close to real time can be envisaged.

Following this line of reasoning, we have made the informed assumption that the average time interval to pronounce the word *delete* is 630 ms. Consequently, we have set the overall speech recognition process to span over 1630 ms so that action *endSpeak* occurs at rate

$$es = 1000/(recognition_t - speaking_t) = 1000/(1630 - 630) = 1$$

With this specification the system is able to respond to users' actions in whatever order they are performed (concurrent gesture and speech, speech followed by gesture, gesture followed by speech) provided that users adapt their behaviour to the speech recognition system. This support by the system makes them to behave as in everyday communication acts where deictic references are usually performed. Gestures and speech are synchronised in order to disambiguate the interpretation of the deictic word with an exact point in space at a reference point in time.

In addition to the speech recogniser, also the use of a pointing device such as the mouse constrains further users' performance. In fact, it is known from the application of the ICS theory [33] that operating any device in a space different from the visual one, under explicit visual control, requires the transfer of the buffer to the limb subsystem. Consequently, users will focus on operating the mouse device and they will be unable to initiate speech. However, they will be able to sustain previously initiated speech that is not in conflict with resource allocation (i.e. it can be performed procedurally).

The problem here is to determine whether the buffer transfer will occur after both pointing and speaking have started. This is indeed the case for a skilled user and it can be explained by the fact that the first phase of a pointing movement is ballistic and occurs without visual control and attentional focus [34]. Consequently, during the first phase of the movement there is no need for buffer transfer since the device is operated with a *natural* and proceduralised gesture regardless of the device space (i.e. without technological awareness). Under this condition, a user that starts to speak while operating the mouse before entering visual control, will achieve both goals in parallel. In the case the visual control is entered (i.e. the buffer is tranferred to limb) before speech is started, pointing occurs first and speaking is done in parallel with the mouse click, usually terminating later in time, in agreement with the experimental results.

As in the case of Drag 'n Drop interaction, the consideration of human factors and usability studies allows to get better insight in the users' performance of the Speak 'n Drop interaction and it enables the development of a detailed stochastic model of users' behaviour.

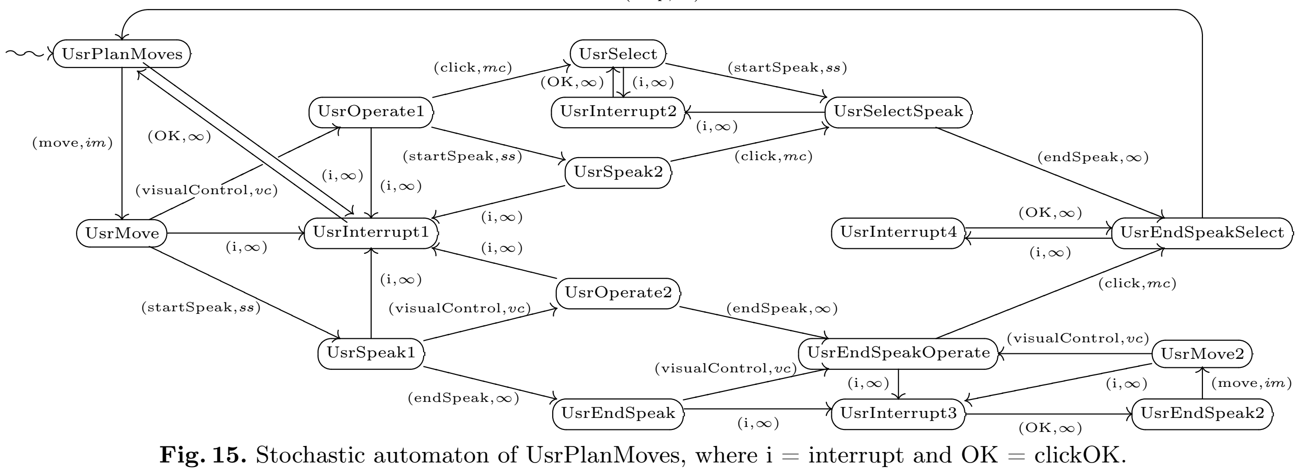

Fig. 15. Stochastic automaton of UsrPlanMoves, where i = interrupt and OK = clickOK.

With reference to Fig. 8, users' behaviour is modelled starting from the initial state *UsrPlanMove*, where the goal is defined. The transition to state *UsrMove*, indicating the ballistic movement, is governed by action *move* occurring at rate *im*1.

$$im1 = 1000/(planning_t + ballistic_t) = 1000/(240 - 670) = 1.1$$

In state *UsrMove* a race condition occurs: users have started speaking already or the visual control is being put in place.

- In the former case, a transition is made to state *UsrSpeak1* governed by action *startSpeak* occurring at rate *ss* and depending on the time required to pronounce the word *delete*. Speech and gesture proceed in parallel according to action *visualControl* occurring at rate *vc1*, and action *endSpeak* [1].
  In state *UsrEndSpeakOperate* both gesture and speech are completed: the spoken command is entered and the icon to be deleted is identified but not selected. Its selection and the subsequent removal happen only when the transition to state *UserEndSpeakSelect* is triggered by the action *click* performed procedurally at rate *mc*.

$$\begin{aligned} ss &= 1000/speaking_t = 1000/630 = 1.6 \qquad (2)\\ vc1 &= 1000/(approach_t + adjust_t) = 1000/290 = 3.4\\ mc &= 1000/click_t = 1000/120 = 8.33 \end{aligned}$$

  An interrupt occurring when in state *UsrEndSpeakOperate* requires the repetition of the pointing gesture to re-identify the icon to be deleted. This is done through state *UsrMove2* with action *move* at rate *im2* and action *visualControl* at rate *vc2*. We have analysed the model with different rates, as discussed in Sect. 7, but we can anticipate that the Speak 'n Drop interaction is stable with respect to those variations.
  The repetition of the spoken command is not required since it has been completed already. Consequently, the selection of any icon causes its removal. A key question is raised here whether or not the user remembers that he or she has entered a (temporary) *removal mode* where any icon in the display can be removed by pure selection without invoking the related command again. This is not dealt with here but it is an interesting point to be investigated in future research on this topic.
- In the latter case, visual control is entered and a transition is made to state *UsrOperate1* by performing the action *visualControl* with rate *vc1*. Here, the pointing movement ends with the identification of the icon to be removed. The selection of the icon and the pronunciation of the command are performed in parallel according to action *click* performed procedurally at rate *mc* and action *startSpeak* performed at rate *ss* as defined in Equation (2).
  In state *UsrSelectSpeak* the icon to be removed is selected but the *delete* command has not been pronounced completely nor has it been recognised. Its completion and the subsequent removal of the selected icon happen only when the transition to state *UserEndSpeakSelect* is triggered by the action *endSpeak* whose rate is a system dependent parameter.
  An interrupt occurring at this point requires the repetition of the spoken command while one of the icons on the display is selected already. This is done through state *UsrSelect* with action *startSpeak* with rate *ss* as defined in Equation (2).

It is interesting to note the duality of the states *UsrEndSpeakOperate* and *UsrSelectSpeak*. Both of them identify a condition in which one of the modalities has reached a stable point in the sense that the task can be completed re-starting from that state after an interrupt has occurred. This requires special attention to be paid to the presentation of the system state in order to help the recovery of the propositional goal. For example, how do users know whether or not the system has already recognised the pronounced words? and by what means is this knowledge made persistent over a period of time?

[1] Note that the rate at which the action *endSpeak* occurs is defined by variable *es* in the automaton described in Fig. 14. It requires the adaptation of users to the system performance.

One cycle of interaction is terminated by the *drop* action occurring at rate $dd = 8.33$. It represents the actual removal of the selected icon from the display and that this is perceived by the users.

Generation and handling of interrupts is described separately by the same process used for the Drag 'n Drop technique and is represented in Fig. 11. The handling of the interrupts by the users is specified by adding to the PEPA specification further states, *UsrInterrupt1*, *UsrInterrupt2*, *UsrInterrupt3*, and *UsrInterrupt4*. An extra action i triggers the transitions to one of these states, occurring with rate ∞. This reflects the fact that the action is generated outside the user model, but that the user is always ready to deal with it. Action *OK* represents that the interrupt has been serviced. Also this action occurs with rate ∞ in the user model being generated externally by the process in charge of interrupt handling. With respect to the Drag 'n Drop model, we have here four states instead of only one, since the interaction can be resumed at different stages of advancement and does not always need to be restarted from the very beginning.

Also the PEPA model described above is a user specification derived from ICS theory augmented with known facts from other branches of cognition studies. It thus also has a general validity within the bounds of the theory from which it was derived and can thus be applied wherever this theory applies.

The full Speak 'n Drop model is specified by the parallel composition of the four systems identified by their initial states *UsrPlanMove*, *SysSpeak*, *SysMove*, and *Interrupt* as follows.

$$((\mathrm{UsrPlanMoves} \bowtie_{\{\mathrm{move,startSpeak,click,endSpeak,drop,interrupt,clickOK}\}} (\mathrm{SysMouse} \bowtie_{\{\mathrm{drop,interrupt,clickOK}\}} \mathrm{SysSpeak})) \bowtie_{\{\mathrm{interrupt,clickOK}\}} \mathrm{Interrupt})$$

Property verification and behavioural analysis can be conducted for the conjoint system as with syndetic modelling. The full textual PEPA specifications used for analysis can be found in Appendix B. For analysis purposes reward structures are added to the PRISM version of the specification in the same way as for the Drag 'n Drop specification.

7 Analysis Results

In this section we present the results of the analyses of both models. As an indicator of the resilience of an interaction technique to external interrupts we study the number of effective 'drops' a user manages to perform during a fixed period of time under a varying number of interrupts occurring randomly during that period. We also study the sensitivity of the models themselves to small but reasonable variations of user behaviour such as the distribution of time over the various phases of cursor movement and the effect of learning. Finally, we show the effect of substituting a 2.5 $\times$ real-time speech recogniser by an ideal real-time one.

7.1 Comparison of nominal performance of interaction techniques.

The expected number of effective 'drop' actions over the first 300 s (i.e. 5 minutes) is a reward measure that can be formalised as follows in the extended CSL temporal logic:

$$R\{\textit{"drops"}\}_{=?}\ [C \leq 300]$$

and the number of 'interrupts' can be formalised in a similar way as:

$$R\{\textit{"interrupts"}\}_{=?}\ [C \leq 300]$$

The notation $R_{=?}$ means that instead of comparing the results with a specific bound the model checker calculates the effective number of drops and interrupts respectively.

Fig. 16 shows the expected number of drops a user manages to perform in the presence of a number of interrupts over a time span of 300 s. So, with an interrupt rate close to zero (left side of the figure) the user performs on average (given the values chosen for the parameters of the model)

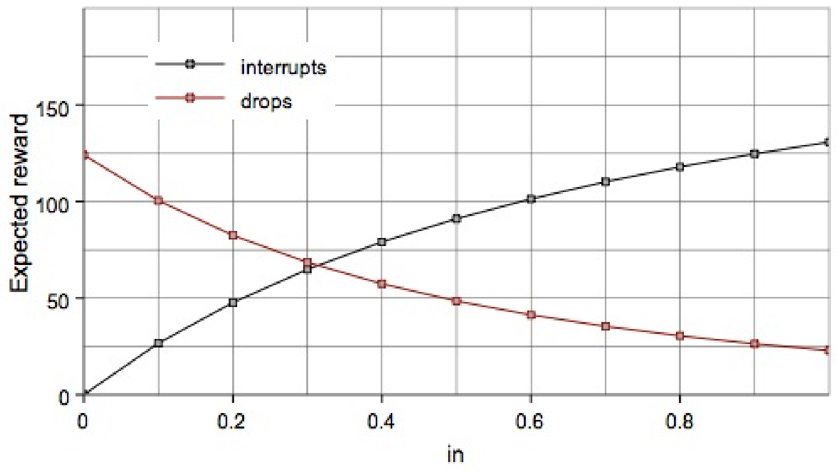

Fig. 16. Drag 'n Drop for nominal performance.

134 drops when using the standard Drag 'n Drop interaction technique. As expected, the user's performance decreases if the interrupt rate increases. In the presence of about 130 interrupts in 300 s (right side of the figure) the user manages to perform only about 27 drops.

The numbers at the x-axis give the value of the rate of action *in* (interrupts), in the models. This rate ranges from (almost) 0 to 1 per s. The reason why the effective number of interrupts at rate 1 in the figures is 130 and not 300 is that the user needs time to handle the interrupt (i.e. moving the cursor to a button and click on it). In our model we have made the assumption that no new interrupts arrive during the time that the user is still handling a previous interrupt. This assumption has been modelled by blocking the occurrence of interrupts during the period when the user is handling an interrupt.

Fig. 17 shows the expected number of drops when a Speak 'n Drop interaction technique is used over a time period of 300 s in the presence of the same number of interrupts as for the Drag 'n Drop interaction technique.

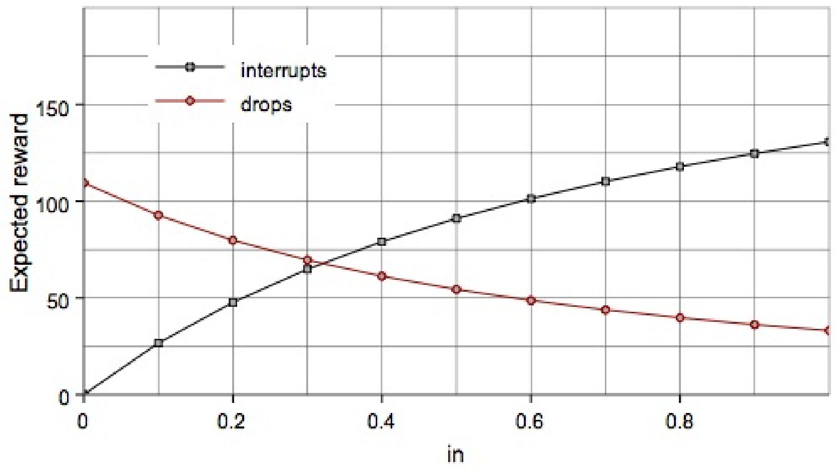

Fig. 17. Speak 'n Drop for nominal performance.

Fig. 18 shows the difference between the expected number of drops for the two interaction techniques. Although the differences over a relatively short time period of 300 s are relatively small, we can nevertheless make some observations. The first observation is that in the absence of interrupts the standard Drag 'n Drop technique leads to a higher average number of drops by the user. This can be explained by the fact that the we have assumed a time of recognition of spoken words equal to 2.5 × *real time* for the model of Speak ’n Drop which is a realistic assumption for the performance of current speech recognition software but which also puts a bound on the speed of the interaction that a user can reach with this interaction technique.

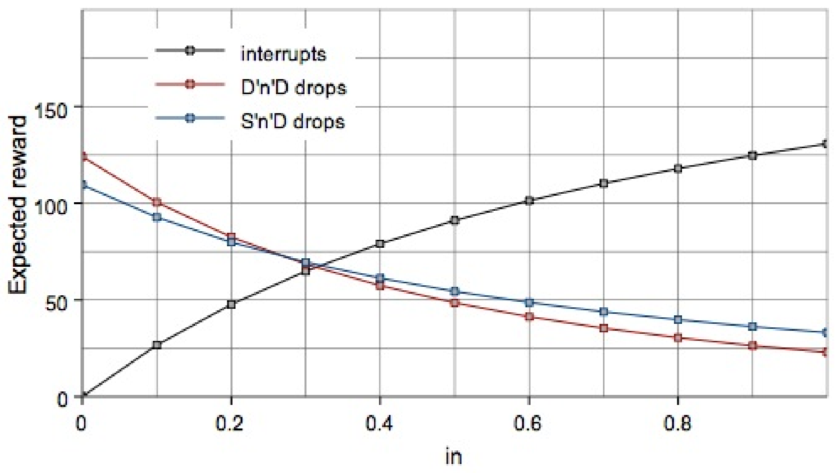

Fig. 18. Comparison of Drag ’n Drop and Speak ’n Drop performance.

A further observation is that when the number of interrupts increases, the Speak ’n Drop technique leads to better performance than Drag ’n Drop. In fact, in extreme circumstances with 130 interrupts per 300 s, with the Speak ’n Drop technique the user manages to drop 33 items, whereas with the Drag ’n Drop technique only 27 items get deleted. The Speak ’n Drop technique scores slightly better than the Drag ’n Drop as the number of interrupts keep increasing. This can be explained by the fact that the Drag ’n Drop technique is more sensitive to interrupts because the total time involved in dragging is relatively long and when an interrupt occurs the user needs to start from the beginning, re-selecting the item and dragging it again towards the trash.

Apart from the basic comparison of performance of the two techniques, under the assumptions made, it is interesting to get a more detailed insight on how the global performance is obtained. In particular we investigate the sensitivity of the conjoint behaviour of the system to the values of the rate parameters of various actions.

7.2 Drag ’n Drop

In Fig. 19, the performance for the Drag ’n Drop technique is shown for different users’ behaviour concerning the distribution of time between the ballistic phase, the approach phase and adjustment phase of a movement. As expected, when the percentage of time spent in the ballistic phase increases with respect to the one spent on the visually controlled phase, the performance is slightly improving. However, a variation of 12% of the splitting between ballistic and visually controlled phases accounts only for a 4% difference in number of drops. This shows that the model is not very sensitive to how the movement is partitioned over time into the phases.

Following the same line of reasoning, in Fig. 20 the line labelled *drops (vc2=120)* shows the effect on performance of a user skilled in sustaining the overall propositional goal and able to drag in a completely procedural way. The line labelled *drops (vc2=290)* represents the performance of a less skilled user that behaves in the same way both when selecting an icon and dragging it to the trash. Clearly, in the former case the performance is uniformly better than in the latter; this situation describes the effect of learning due to frequent recurrence of the operation. According to the modelling experiments, an already skilled user can increase the performance by 8% with such a learning effect.

Finally, Fig. 21 relates the performance of various actions with respect to the overall performance. In other terms, it shows the total number of *move*, *push*, and *drag* actions required to obtain the corresponding number of drops in the presence of the various number of interrupts

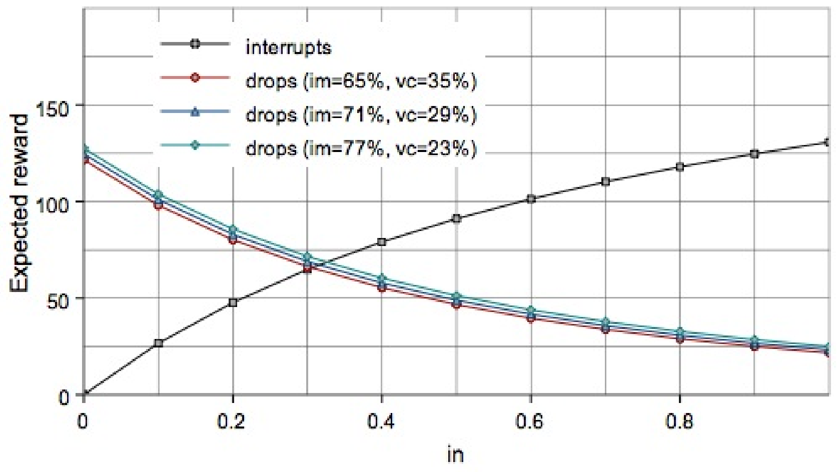

Fig. 19. Drag 'n Drop performance with varying behaviour.

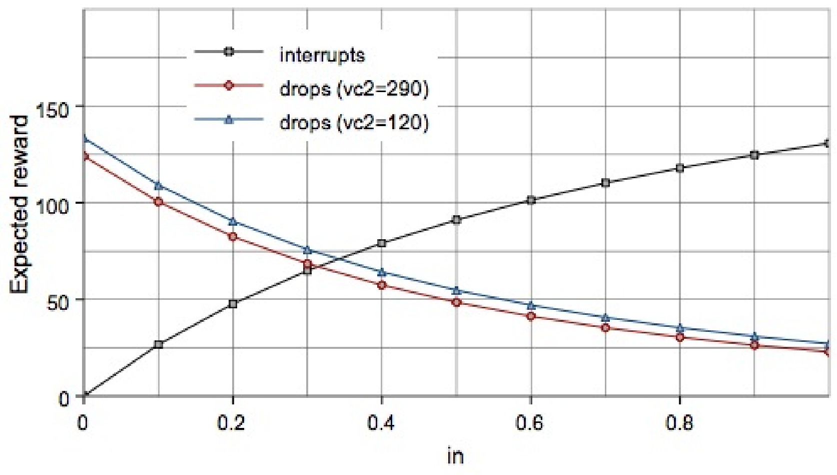

Fig. 20. Effect of learning in Drag 'n Drop performance.

occurring in 300 s. Since the Drag 'n Drop interaction technique is a sequential one, it is clear that actions occurring towards the beginning of the sequence fail more often than those close to the end.

Fig. 22 shows the increasing percentage of intermediate actions with the increase of the number of interrupts. Already with about 100 interrupts in 300 s ($in = 0.5$ or 1 interrupt every 3 s) 50% of move actions, 35% of push, and 10% of drags are interrupted. Obviously, such interrupted actions do not lead to a successful drop.

7.3 Speak 'n Drop

We perform the same type of analysis as for the Drag 'n Drop technique so we can compare the results. Fig. 23 shows the performance for the Speak 'n Drop technique comparing different distributions of movement time over the the ballistic and the visually-controlled phases of user movement. The difference of performance is negligible, under the nominal assumptions, since a variation of 12% in the distribution of the time accounts for only a 1% difference in number of drops. This is not surprising since the speech modality of this technique is predominant over that of gesture. Variations in the performance of movement have a lower impact w.r.t. pure gestural interaction.

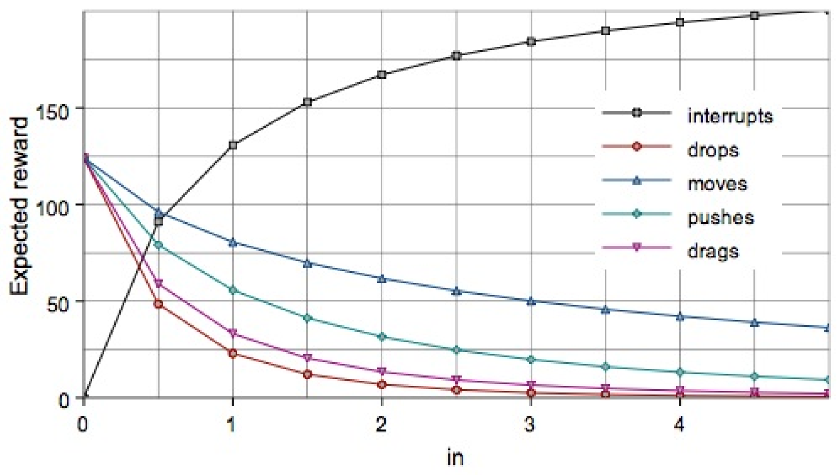

Fig. 21. Performance relation among the number of actions in Drag 'n Drop.

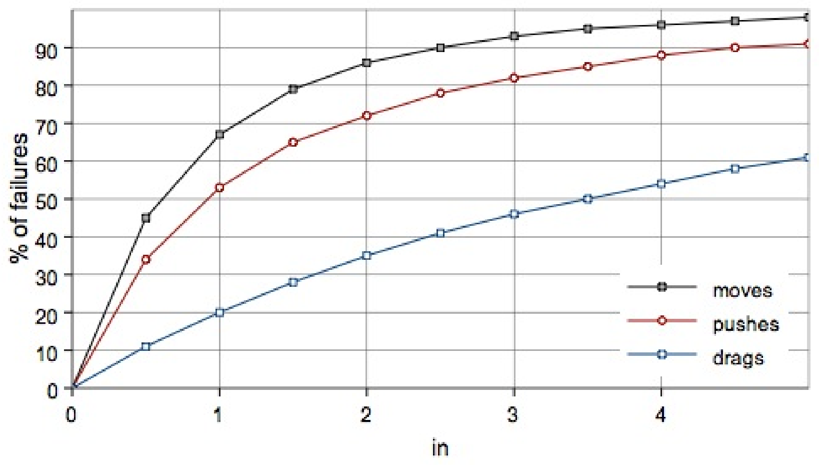

Fig. 22. Percent of failures of intermediate actions in Drag 'n Drop.

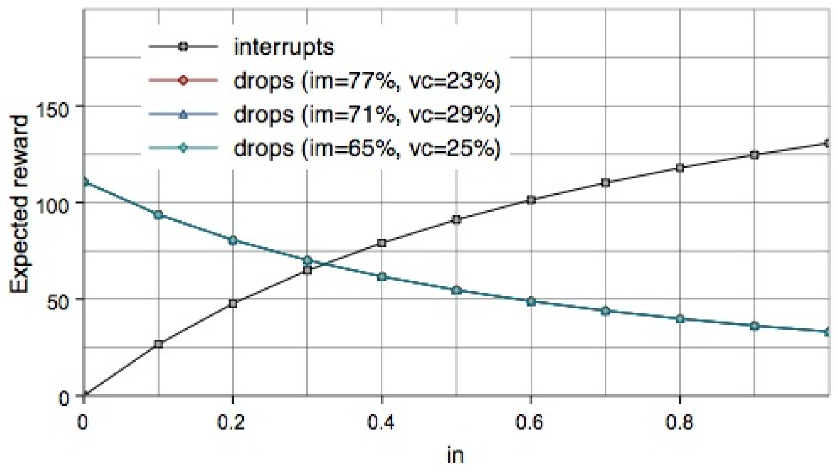

Fig. 23. Speak 'n Drop performance with varying behaviour.

For the same reason, also the learning effect that can be observed with the Drag 'n Drop technique, does not apply. In addition, the learning effect is mitigated by the fact that, in principle, the icons are always randomly positioned on the display and, consequently, there is no fixed position to point to. In fact, there is no significant variation in performance regardless of the user behaving procedurally or not. This is clearly shown in Fig. 24.

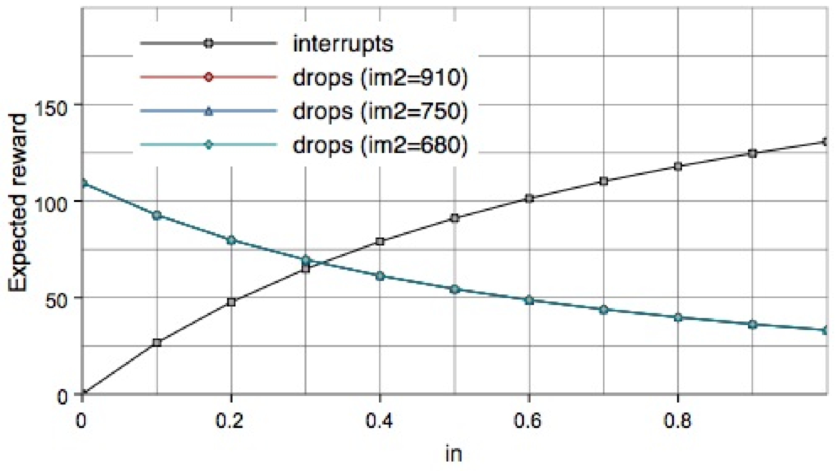

Fig. 24. Effect of learning moves in Speak 'n Drop performance.

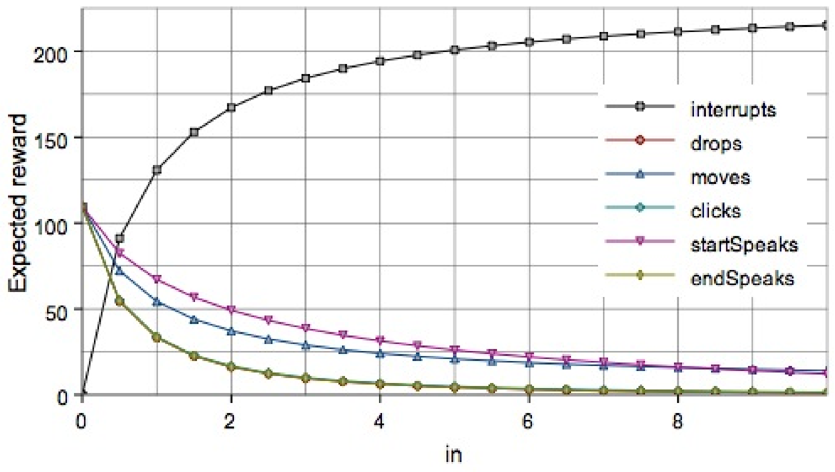

Fig. 25. Performance relation of number of actions in Speak 'n Drop.

With the Speak 'n Drop technique, it is particularly interesting to relate other actions to the number of drops. In Fig. 25 two facts show up clearly:

1. The number of *startSpeak* actions keep decreasing also when the number of other actions have reached a more or less stable minimum, as demonstrated by the crossing of the *startSpeaks* and *moves* curves at a high number of interrupts (in the order of 200) in 300 s.
2. The number of failures of the *click* and the *endSpeak* actions are substantially equal to the number of drops for any number of interrupts considered.

The first phenomenon is related to the choice of the *endSpeak* action rate. In line with the available literature, we have specified a speech recognition engine performing 2.5 $\times$ *real time*. Considering that the time for pronouncing the word *delete* is 630 ms on average with very minor variations, the *endSpeak* rate has been set to 1. This means that the average time needed to recognise the command is 1 second (1000 ms) after it has been pronounced: a relatively long time when compared to the average duration of the other actions. Fig. 26 shows the results for a model where we assume a *real-time* speech recogniser. In this case the number of *startSpeak* actions decreases in a similar way as for the other actions. This is an example of a case where the user is forced to adapt its behaviour to the performance of the computing system; a better performance of one component results in a better performance of the overall system.

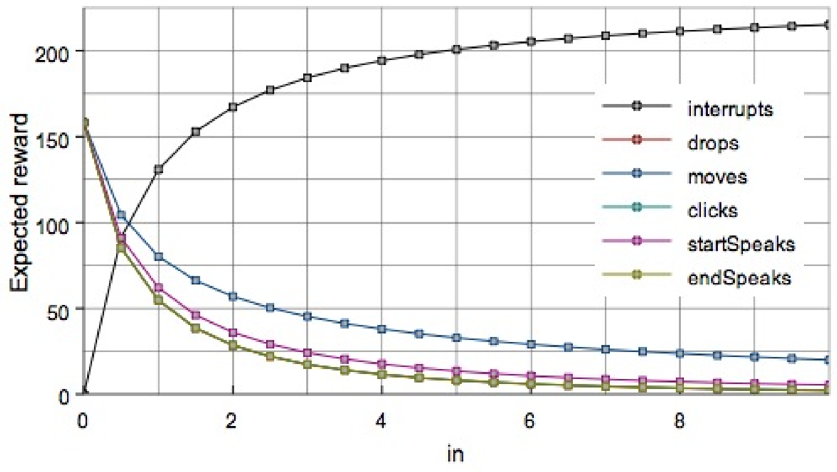

Fig. 26. Performance relation of number of actions in Speak 'n Drop with real-time speech recognition.

The second phenomenon observable in Fig. 25 is that once one subtask in the interaction technique (i.e. gesture or speech modality) has been completed, there is almost a 100% probability that also the other will complete successfully without an intervening interrupt. This can be explained by the level of parallelism that a skilled user can reach performing the tasks. Since both subtasks may proceed in parallel, when one of them completes also the other one is near completion and the duration between the two is so short that further interrupts are unlikely to happen.

Figs. 27 and 28 show the percentage of failures for the intermediate actions. It is clear from the figures that the percentage of failing *clicks* and *endSpeaks* is very low, that the increase of failing actions is moderate, and that speech fails less often than selection.

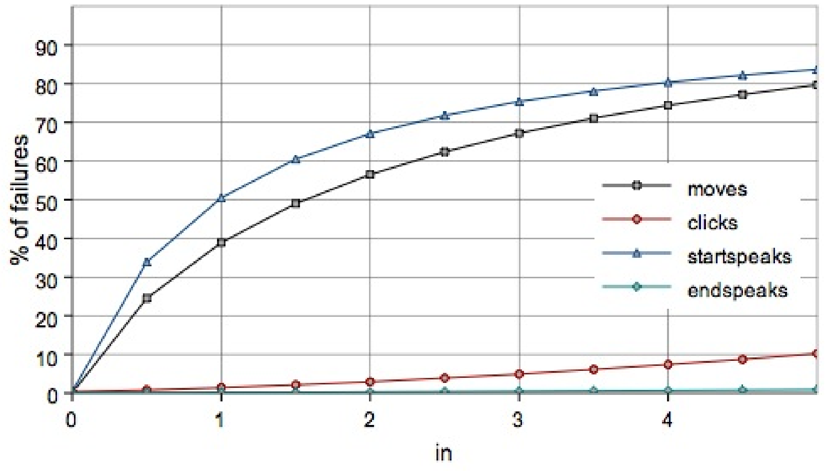

Fig. 27. Percentage of failures of intermediate actions in Speak 'n Drop.

7.4 Comparison of results

Considering the results analysed so far, a comparison can be made between the two interaction techniques, that have been explored, in terms of their relative performance. Referring to Fig. 18 one might infer that Drag 'n Drop interaction is better suited for low interrupt rates while as

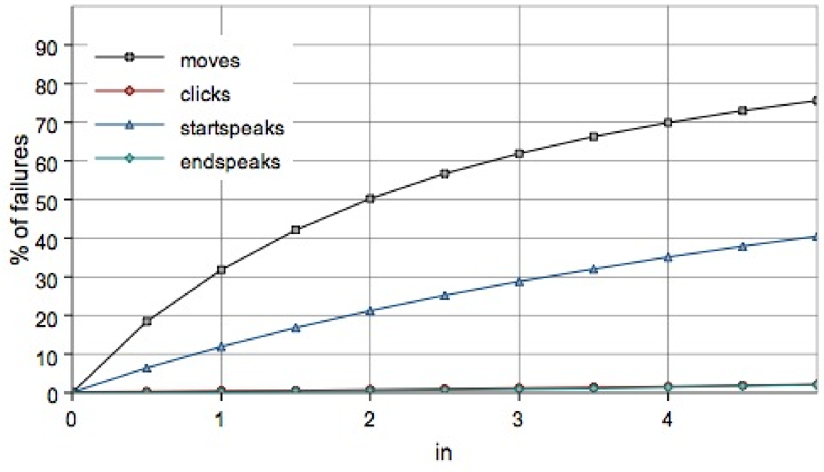

Fig. 28. Percentage of failures of intermediate actions in Speak 'n Drop with real-time speech recognition.

the number of interrupts increases, multimodal Speak 'n Drop takes the lead. On the other hand, referring to Figs. 19 and 23, and 20 and 24, the multimodal interaction proves to be more resilient both to the number of interrupts and to the varying performance of users.

Also, taking into account the percentage of failures of intermediate actions caused by interrupts, it can be expected that the Speak 'n Drop interaction might be more appreciated by the users because they experience less frustration than with the Drag 'n Drop technique where for many interrupts they continuously need to restart their primary activity.

Moreover, under the objective assumption that simple command languages can be recognised in real time by a speech recogniser, Fig. 29 shows the comparison of the overall performance of the interaction techniques (i.e. number of *drops*) comparing drag 'n drop and speak 'n drop with real-time speech recognition. It is clear that multimodal interaction gives the best results for any number of interrupts.

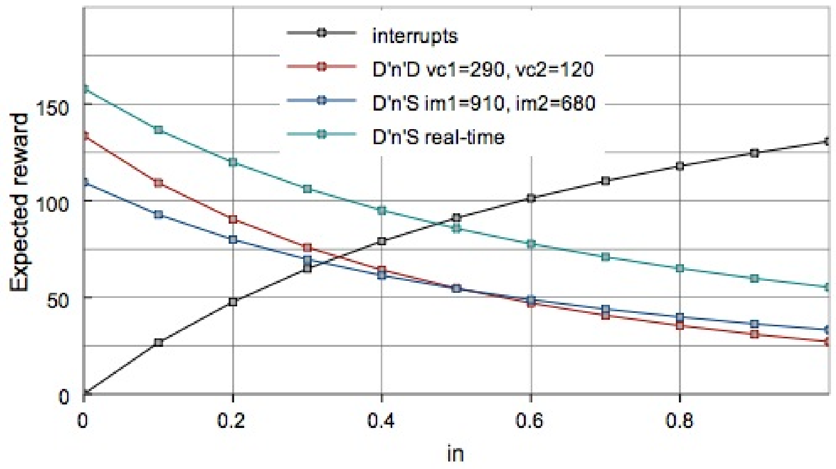

Fig. 29. Comparison of performance of Drag 'n Drop, Speak 'n Drop, and Speak 'n Drop with real-time speech recognition.

The models seem to produce rather plausible results despite their relatively high level of abstraction and the fact that the values of the parameters have been gathered from experiments published in the literature each referring to different concerns of the overall user behaviour. Further validation of this kind of models could be performed by empirical research. If the models turn out to be quite reasonable predictors of the performance and the resilience to interrupts of the interaction techniques then this would allow designers of such techniques to use model-based analysis during design. Empirical validation, which is a costly and time consuming activity, could then be used only during the later stages of development.

8 Conclusions and Future Work

In this paper we have developed syndetic stochastic models of two kinds of interaction techniques used for the removal of items on a screen (Drag 'n Drop and Speak 'n Drop) to analyse their resilience to the presence of external interrupts appearing in the form of pop-up screens. The part of the models concerning human behaviour has been based on the well-established cognitive theory ICS that provides information on mental configurations involved in the performance of subtasks of the Drag 'n Drop and the multi-modal Speak 'n Drop interactions, as well as on estimates of average durations involved in the cognitive aspects of these tasks. Literature on Fitts' law has been used to obtain timing information on task execution such as limb movements involved in selection and dragging.

The models in this work have been specified in the process algebra PEPA and analysed with the stochastic model checker PRISM. The latter also provides automatic verification of measures, expressed as logical formulae, that concern rewards associated with actions. This allows for the analysis of the resilience of the interaction techniques to external interrupts by studying how the number of drop-actions changes when the number of interupts increases.

The results obtained so far are encouraging because with relatively simple models we have obtained rather plausible and attainable results. To what extent exactly these models may serve to predict performance of human interaction in the presence of external interrupts is a topic for future research. However, the results of the model-based analysis are in a form that allows for validation and comparison with e.g. results obtained from empirical studies, i.e. with real human users. This brings us also to a further issue and that is that in real settings human errors cannot be excluded and may clearly influence the overall performance. In our current work we have not included erroneous behaviour in our models because we wanted to study the effect of interrupts separately from a possible effect of errors. Once the effect of both aspects are better understood it would be very interesting to add erroroneous behaviour to our models to investigate whether it is possible to predict their combined effect with a model-based approach. A particular challenge in this respect will be to take the effects of cognitive load into account. A high cognitive load may result in more errors, but perhaps also in a slower capability of resuming the original task after an interrupt.

A further topic for future research is the adaptation of the proposed methodology to formalisms that are more commonly used in the HCI community such as ICO [65], which is based on Petri nets. This involves the development of a stochastic extension of ICO, inspired by stochastic Petri nets, and the automatic translation of such specifications into the input language of a stochastic model checker, e.g. PRISM, for example by adapting existing translators that convert stochastic Petri net models into PRISM specifications.

Acknowledgements

We would like to thank Stephen Gilmore of the University of Edinburgh for providing us a stand-alone version of their pepa2prism compiler.

References

1. E.M. Altmann and J.G. Trafton, Memory for goals: An activation-based model. *Cognitive Science* 26(1), 2002, 39–83.
2. E.M. Altmann and J.G. Trafton, Timecourse of Recovery from Task Interruption: Data and a Model. *Psychonomics Bulletin and Review* 14(6), 2007, 1079–1084.

3. R. Alur and T. Henzinger, Reactive modules. *Formal Methods in System Design* 15(1), 1999, 7–48.
4. A. Aziz, K. Sanwal, V. Singhal and R. Brayton, Model checking continuous time Markov chains. *ACM Transactions on Computational Logic* 1(1), 2000, 162–170.
5. C. Baier, J.-P. Katoen and H. Hermanns, Approximate symbolic model checking of continuous-time Markov chains. In *Proceedings CONCUR'99*, LNCS 1664, Springer, 1999, 146–162.
6. C. Baier, B.R. Haverkort, H. Hermanns and J.-P. Katoen, Automated performance and dependability evaluation using model checking. In *Performance'02 Tutorial Lectures*, LNCS 2459, Springer, 2002, 261–289.
7. B.P. Bailey, J.A. Konstan and J.V. Carlis, Measuring the effects of interruptions on task performance in the user interface. In *Proceedings SMC'00*, IEEE Press, 2000, 757–762.
8. B.P. Bailey, J.A. Konstan and J.V. Carlis, The effects of interruptions on task performance, annoyance, and anxiety in the user interface. In *Proceedings INTERACT'01*, IOS Press, 2001, 593–601.
9. P.J. Barnard, Interacting Cognitive Subsystems: Modelling working memory phenomena within a multi-processor architecture. Chapter 9 in *Models of Working Memory: Mechanisms of Active Maintenance and Executive Control* (A. Miyake and P. Shah, Eds), Cambridge University Press, 1999, 298–339.
10. P.J. Barnard and J. May, Cognitive modelling for user requirements. In *Computers, Communication and Usability: Design Issues, Research and Methods for Integrated Services* (P.F. Byerley, P.J. Barnard and J. May, Eds.), Elsevier, 1993, 101–145.
11. P.J. Barnard and J. May, Interactions with advanced graphical interfaces and the deployment of latent human knowledge. In *Proceedings DSV-IS'94*, Springer, 1994, 15–49.
12. P.J. Barnard, J. May, D.J. Duke and D.A. Duce, Systems, interactions and macrotheory. *ACM Transactions on Human-Computer Interaction* 7(2), 2000, 222–262.
13. R. Bastide, P.A. Palanque, D.-H. Le and J. Munoz, Integrating rendering specifications into a formalism for the design of interactive systems. In *Proceedings DSV-IS'98*, Springer, 1998, 171–190.
14. R.A. Bolt, "Put-that-there": Voice and gesture at the graphics interface. In *Proceedings SIGGRAPH'80*, ACM Press, 1980, 262–270.
15. E. Brinksma, H. Hermanns and J.-P. Katoen (Eds.), *Lectures on Formal Methods and Performance Analysis*, LNCS 2090, Springer, 2001.
16. J.J. Brixey, M. Walji, J. Zhang, T.R. Johnson and J.P. Turley, Proposing a Taxonomy and Model of Interruption. In *Proceedings Healthcom'04*, IEEE Press, 2004, 184–188.
17. M.E. Brudzinski, R.M. Ratwani and J.G. Trafton, Goal and spatial memory following interruption. In *Proceedings ICCM'07*, Psychology Press, 2007, 139–144.
18. P. Buchholz, J.-P. Katoen, P. Kemper and C. Tepper, Model-checking large structured Markov chains. *Journal of Logic and Algebraic Programming* 56, 2003, 69–96.
19. D.M. Cades, J.G. Trafton, D.A. Boehm-Davis and C.A. Monk, Does the difficulty of an interruption affect our ability to resume? In *Proceedings HFES'07*, Human Factors and Ergonomics Society, 2007, 234–238.
20. S. Caffiau, P. Girard, D.L. Scapin, L. Guittet and L. Sanou, Assessment of Object Use for Task Modeling. In *Proceedings TAMODIA'08*, LNCS 5247, Springer, 2008, 14–28.
21. C.D. Chisholm, E.K. Collison, D.R. Nelson and W.H. Cordell, Emergency department workplace interruptions: Are emergency physicians "interrupt-driven" and "multitasking"? *Academic Emergency Medicine* 7, 2000, 1239–1243.
22. E.M. Clarke, E.A. Emerson and A.P. Sistla, Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems* 8, 1986, 244–263.
23. E.M. Clarke, O. Grumberg and D.A. Peled, *Model Checking*. MIT Press, 1999.
24. M. Czerwinski, E. Cutrell and E. Horvitz, Instant Messaging and Interruption: Influence of Task Type on Performance. In *Proceedings OzCHI'00*, 2000, 356–361.
25. M. Czerwinski, E. Horvitz and S. Wilhite, A diary study of task switching and interruptions. In *Proceedings CHI'04*, ACM Press, 2004, 175–182.
26. D. Diaper and N.A. Stanton (Eds.), *The Handbook of Task Analysis for Human-Computer Interaction*. Lawrence Erlbaum Associates, 2004.
27. M. Diez, D.A. Boehm-Davis and R.W. Holt, Model-based predictions of interrupted checklists. In *Proceedings HFES'02*, Human Factors and Ergonomics Society, 2002, 250–254.
28. A. Dix, D. Ramduny-Ellis and J. Wilkinson, Trigger Analysis: understanding broken tasks. Chapter 19 in [26], 2004, 381–400.
29. D.A. Duce and D.J. Duke, Syndetic Modelling: Computer Science Meets Cognitive Psychology. In *Proceedings FME'00*, ENTCS 43, Elsevier, 2001, 50–74.

30. D.J. Duke, P.J. Barnard, D.A. Duce and J. May, Syndetic modelling. *International Journal of Human Computer Interaction* 13(4), 1998, 337–393.
31. D.J. Duke and M.D. Harrison, Abstract interaction objects. *Computer Graphics Forum* 12(3), 1993, 25–36.
32. D.J. Duke and M.D. Harrison, Interaction and task requirements. In *Proceedings DSV-IS'95*, Springer, 1995, 54–75.
33. G.P. Faconti and D.J. Duke, Device models. In *Proceedings DSV-IS'96*, Springer, 1996, 73–91.
34. G.P. Faconti and M. Massink, Analysis of pointing tasks on a white board. In *Revised Papers of DSV-IS'06*, LNCS 4323, Springer, 2007, 185–198.
35. G.P. Faconti and F. Paternò, An approach to the formal specification of the components of an interaction. In *Proceedings Eurographics'90*, Elsevier, 1990, 481–494.
36. P.M. Fitts, The information capacity of the human motor system in controlling amplitude of movement. *Journal of Experimental Psychology* 47(6), 1954, 381–391.
37. T. Gillie and D. Broadbent, What makes interruptions disruptive? A study of length, similarity and complexity. *Psychological Research* 50(4), 1989, 243–250.
38. I.E. Gordon, *Theories of Visual Percept*. Psychology Press, 2004.
39. B.R. Haverkort, Markovian models for performance and dependability evaluation. In [15], 2001, 38–83.
40. H. Hermanns, J-P. Katoen, J. Meyer-Kayser and M. Siegle, A tool for model-checking Markov chains. *International Journal on Software Tools for Technology Transfer* 4(2), 2003, 153–172.
41. J. Hillston, *A Compositional Approach to Performance Modelling*. Cambridge University Press, 1996.
42. E. Horvitz and J. Apacible, Learning and reasoning about interruption. In *Proceedings ICMI'03*, ACM Press, 2003, 20–27.
43. *ICASSP (IEEE International Conference on Acoustics, Speech and Signal Processing) Proceedings series*, IEEE Press, 1976–2009.
44. S.T. Iqbal and B.P. Bailey, Effects of intelligent notification management on users and their tasks. In *Proceedings CHI'08*, ACM Press, 2008, 93–102.
45. F. Jambon, Formal modelling of task interruptions. In *Proceedings CHI'96*, ACM Press, 1996, 45–46.
46. A. Kapoor and E. Horvitz, Experience sampling for building predictive user models: A comparative study. In *Proceedings CHI'08*, ACM Press, 2008, 657–666.
47. V. Kulkarni, *Modeling and Analysis of Stochastic Systems*. Chapman & Hall, 1995.
48. M. Kwiatkowska, G. Norman and D. Parker, Probabilistic symbolic model checking with PRISM: A hybrid approach. In *Proceedings TACAS'02*, LNCS 2280, Springer, 2002, 52–66.
49. G.D. Langolf, *Human motor performance in precise microscopic work*. Ph.D. Thesis, University of Michigan, 1973.
50. I.S. MacKenzie, Fitts' law as a research and design tool in human-computer interaction. *International Journal of Human Computer Interaction* 7(1), 1992, 91–139.
51. G. Mark, D. Gudith and U. Klocke, The cost of interrupted work: more speed and stress. In *Proceedings CHI'08*, ACM Press, 2008, 107–110.
52. D.S. McCrickard and C.M. Chewar, Attuning notification design to user goals and attention costs. *Communications of ACM* 46(3), 2003, 67–72.
53. M.A. McDaniel, G.O. Einstein, T. Graham and E. Rall, Delaying execution of intentions: overcoming the costs of interruptions. *Applied Cognitive Psychology* 18(5), 2004, 533–547.
54. D.C. McFarlane, Interruption of People in Human-Computer Interaction: A General Unifying Definition of Human Interruption and Taxonomy. NRL Formal Report NRL/FR/5510-97-9870, US Naval Research Laboratory, Washington, 1997.
55. D.C. McFarlane, Coordinating the interruption of people in human-computer interaction. In *Proceedings INTERACT'99*, IOS Press, 1999, 295–303.
56. D.C. McFarlane, Comparison of four primary methods for coordinating the interruption of people in human-computer interaction. *Human-Computer Interaction* 17, 2002, 63–139.
57. D. McNeill, Language as gesture (Gesture as language). In *Proceedings WIGLS'96*, University of Delaware, 1996, 1–20.
58. C. Monk, D. Boehm-Davis and J.G. Trafton, Recovering from interruptions: Implications for driver distraction research. *Human Factors* 46, 2004, 650–663.
59. D. Morris, A.B. Brush and B.R. Meyers, SuperBreak: Using interactivity to enhance ergonomic typing breaks. In *Proceedings CHI'08*, ACM Press, 2008, 1817–1826.
60. National Transportation Safety Board, Aircraft Accident Report NTSB/AAR-69/08: Pan American World Airways. National Transportation Safety Board, 1968.
61. National Transportation Safety Board, Aircraft Accident Report NTSB/AAR-88/05: Northwest Airlines. National Transportation Safety Board, 1988.

62. B. O'Conaill and D. Frohlich, Timespace in the workplace: Dealing with interruptions. In *Proceedings CHI'95*, ACM Press, 1995, 262–263.
63. A. Oulasvirta and P. Saariluoma, Surviving task interruptions: Investigating the implications of long-term working memory theory. *International Journal of Human-Computer Studies* 64(10), 2006, 941–961.
64. S.L. Oviatt, A. De Angeli and K. Kuhn, Integration and synchronization of input modes during multimodal human-computer interaction. In *Proceedings CHI'97*, ACM Press, 1997, 415–422.
65. P.A. Palanque and R. Bastide, Petri net based Design of User-driven Interfaces Using the Interactive Cooperative Objects Formalism. In *Proceedings DSV-IS'94*, Springer, 1994, 383–400.
66. D. Parker, G. Norman and M. Kwiatkowska, *PRISM 2.0—Users' Guide*, February 2004. URL `http://www.cs.bham.ac.uk/~dxp/prism`.
67. F. Paternò, C. Mancini and S. Meniconi, ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models. In *Proceedings INTERACT'97*. Chapman & Hall, 1997, 362–369.
68. P. Pinheiro da Silva, User Interface declarative models and Development environments: A survey. In *Proceedings DSV-IS'00*, LNCS 1946, Springer, 2000, 207–226.
69. L. Ramanna and R. Ramachandran, Real-time Pitch Tracking of Speech Signals using the Robust Algorithm for Pitch Tracking on a Personal Digital Assistant. *Speech Signal Processing: Final Project Report*, 2007. URL `http://www.utdallas.edu/~lakshmish/pubs_files/Speech RAPT Report.pdf`.
70. R.M. Ratwani, J.M. McCurry and J.G. Trafton, Predicting post completion errors using eye movements. In *Proceedings CHI'08*, ACM Press, 2008, 539–542.
71. J.A. Rukab, K.A. Johnson-Throop, J. Malin and J. Zhang, A Framework of Interruptions in Distributed Team Environments. In *Proceedings MEDINFO'04*, Studies in Health Technology and Informatics 107, IOS Press, 2004, 1282–1286.
72. A. Silberschatz, P. Galvin and G. Gagne, *Operating Systems Concepts*. John Wiley & Sons, 2008.
73. C. Speier, I. Vessey and J.S. Valacich, The effects of interruptions, task complexity, and information presentation on computer-supported decision-making performance. *Decision Sciences* 34(4), 2003, 771–797.
74. N.M. Su and G. Mark, Communication chains and multitasking. In *Proceedings CHI'08*, ACM Press, 2008, 83–92.
75. J.G. Trafton, E.M. Altmann, D.P. Brock and F.E. Mintz, Preparing to resume an interrupted task: Effects of prospective goal encoding and retrospective rehearsal. *International Journal of Human-Computer Studies* 58(5), 2003, 583–603.
76. J.G. Trafton and C.A. Monk, Task Interruptions. *Reviews of Human Factors and Ergonomics* 3, 2007, 111–126.
77. K. Tsukada, K. Okada and Y. Matsushita, A Cooperative Support System Based on Multiplicity of Task. In *Proceedings WCC'94*, North-Holland, 1994, 69–74.
78. A.L. Tucker and S.J. Spear, Operational failures and interruptions in hospital nursing. *Health Services Research* 41, 2006, 643–662.
79. M. Walji, J. Brixey, K. Johnson-Throop and J. Zhang, A theoretical framework to understand and engineer persuasive interruptions. In *Proceedings CogSci'04*, Lawrence Erlbaum Associates, 2004.
80. R.L. West and G. Nagy, Using GOMS for Modeling Routine Tasks Within Complex Sociotechnical Systems: Connecting Macrocognitive Models to Microcognition. *Journal of Cognitive Engineering and Decision Making* 1(2), 2007, 186–211.

A PEPA Specification of Drag 'n Drop

```
                    % all times in ms
im = 1000/910;      % time of planning (240 ms) plus ballistic (670 ms) movement
vc1 = 1000/290;     % time of approach + adjust movement
vc2 = 1000/290;     % as above (1000/120 for procedural case)
in ;                % interrupt time variable
pb = 1000/120;      % time of completion of movement finishing with a push button
sd = 1000/680;      % time planning (0) and ballistic (680 ms)
dd = 1000/120;      % time to release (120 ms)
ok = 1000/1300;     % time needed to handle pop-up interrupt (1300 ms)

#UsrPlanMove = (move,im).UsrMove + (interrupt,infty).UsrInterrupt;
#UsrMove = (visualControl,vc1).UsrOperate + (interrupt,infty).UsrInterrupt;
#UsrOperate = (push,pb).UsrSelect + (interrupt,infty).UsrInterrupt;
#UsrSelect  = (drag,sd).UsrEndMove + (interrupt,infty).UsrInterrupt;
#UsrEndMove = (visualControl,vc2).UsrEnd + (interrupt,infty).UsrInterrupt;
#UsrEnd = (drop,dd).UsrPlanMove + (interrupt,infty).UsrInterrupt;
```

```
#UsrInterrupt = (clickOK,infty).UsrPlanMove;

#SysPlanMove = (move,infty).SysMove + (interrupt,infty).SysInterrupt;
#SysMove = (push,infty).SysSelect + (interrupt,infty).SysInterrupt;
#SysSelect = (drag,infty).SysEnd + (interrupt,infty).SysInterrupt;
#SysEnd = (drop,infty).SysPlanMove + (interrupt,infty).SysInterrupt;
#SysInterrupt = (clickOK,infty).SysPlanMove;

#Interrupt = (interrupt,in).InterruptOK;
#InterruptOK = (clickOK,ok).Interrupt;

(UsrPlanMove <move,push,drag,drop,interrupt,clickOK> (SysPlanMove <interrupt,clickOK> Interrupt))
```

B PEPA Specification of Speak 'n Drop

```
                 % all times in ms
im = 1000/910;   % time of planning initial movement plus ballistic movement
vc = 1000/290;   % time of visual control
in ;             % interrupt time variable
mc = 1000/80;    % time of completion of movement finishing with a mouse click
ss = 1000/630;   % time for user to start speaking and completing the utterance
es = 1000/1000;  % time for user to end speaking (plus recognition and feedback)
dd = 1000/120;   % time to drag icon to trash and drop it there
ok = 1000/1300;  % time to handle pop-up interrupt

#UsrPlanMove = (move,im).UsrMove + (interrupt,infty).UsrInterrupt1;
#UsrMove = (visualControl,vc).UsrOperate1 + (startSpeak,ss).UsrSpeak1 + (interrupt,infty).UsrInterrupt1;
#UsrOperate1 = (startSpeak,ss).UsrSpeak2 + (click,mc).UsrSelect + (interrupt,infty).UsrInterrupt1;
#UsrSpeak2 = (click,mc).UsrSelectSpeak + (interrupt,infty).UsrInterrupt1;
#UsrSelect = (startSpeak,ss).UsrSelectSpeak + (interrupt,infty).UsrInterrupt2;
#UsrSelectSpeak = (endSpeak,infty).UsrEndSpeakSelect + (interrupt,infty).UsrInterrupt2;
#UsrInterrupt2 = (clickOK,infty).UsrSelect;
#UsrSpeak1 = (visualControl,vc).UsrOperate2 + (endSpeak,infty).UsrEndSpeak + (interrupt,infty).UsrInterrupt1;
#UsrOperate2 = (endSpeak,infty).UsrEndSpeakOperate + (interrupt,infty).UsrInterrupt1;
#UsrInterrupt1 = (clickOK,infty).UsrPlanMove;
#UsrEndSpeak = (visualControl,vc).UsrEndSpeakOperate + (interrupt,infty).UsrInterrupt3;
#UsrEndSpeak2 = (move,im).UsrMove2 + (interrupt,infty).UsrInterrupt3;
#UsrMove2 = (visualControl,vc).UsrEndSpeakOperate + (interrupt,infty).UsrInterrupt3;
#UsrEndSpeakOperate = (click,mc).UsrEndSpeakSelect + (interrupt,infty).UsrInterrupt3;
#UsrInterrupt3 = (clickOK,infty).UsrEndSpeak2;
#UsrEndSpeakSelect = (drop,dd).UsrPlanMove + (interrupt,infty).UsrInterrupt4;
#UsrInterrupt4 = (clickOK,infty).UsrEndSpeakSelect;

#SysMouse = (move,infty).SysSelectM + (interrupt,infty).SysInterruptM1;
#SysSelectM = (click,infty).SysEndM + (interrupt,infty).SysInterruptM1;
#SysInterruptM1 = (clickOK,infty).SysMouse;
#SysEndM = (drop,infty).SysMouse + (interrupt,infty).SysInterruptM2;
#SysInterruptM2 = (clickOK,infty).SysEndM;

#SysSpeak = (startSpeak,infty).SysSelectS + (interrupt,infty).SysInterruptS1;
#SysSelectS = (endSpeak,es).SysEndS + (interrupt,infty).SysInterruptS1;
#SysInterruptS1 = (clickOK,infty).SysSpeak;
#SysEndS = (drop,infty).SysSpeak + (interrupt,infty).SysInterruptS2;
#SysInterruptS2 = (clickOK,infty).SysEndS;

#Interrupt = (interrupt,in).InterruptOK;
#InterruptOK = (clickOK,ok).Interrupt;

((UsrPlanMove <move,startSpeak,click,endSpeak,drop,interrupt,clickOK>
         (SysMouse <drop,interrupt,clickOK> SysSpeak)) <interrupt,clickOK> Interrupt)
```

C Added Reward Structures

These are two examples of the reward structures added to the PRISM version of the specifications:

```
rewards "drops"
        [drop] true : 1;
endrewards

rewards "interrupts"
        [interrupt] true : 1;
endrewards
```