

Comparing non-deterministic models of tasks and devices

Panos Markopoulos

Department of Computer Science
Queen Mary and Westfield College
University of London
Mile End Road, London E1 4NS

Abstract. This paper discusses a simple model of the temporal behaviour of users and devices. A range of possible comparisons that can be drawn between the two models is examined. Without assuming a theory of cognition or a process model for user interface software design, this exercise reveals a significant range design issues and demonstrates how the simplest models of interaction can provide a framework for reasoning about usability.

Keywords. Labelled Transition Systems, Observational Equivalence, Testing, Task Conformance, Design of Interactive Systems.

1. Introduction

This paper discusses the semantics underlying the comparison between models of user behaviour to models of interactive systems. The formal framework for this discussion is that of Labelled Transition Systems (LTS). This is a very simple and general model of behaviour which underlies many languages for the specification of concurrent systems, that have also been used for the specification of interactive systems. Higher level models of systems or users like interactors (Markopoulos 1997, Paternó and Faconti 1992) and tasks (Markopoulos, Johnson and Rowson 1997, Breedvelt-Schouten, Paternó and Severijns 1997) can be interpreted as LTS. The discussion here assumes that such a higher level representation exists and can be interpreted to a LTS according to the semantics of the specification language used. The discussion does not adopt or assume any theoretical model of user behaviour or cognition or any architectural organisation of the interactive system. Rather, the paper investigates the assumptions that the comparison of system and user behaviours necessitates and their practical ramifications for interactive systems design.

Without implying any cognitive theory or view of the system design process, we shall call the user behaviour a *task* and a system behaviour the *device*. The comparison of devices and tasks which are modelled as LTSs revolves around two issues. First, the correspondence between task actions and device actions. Second, the comparison between the task and device behaviour specifications. These are discussed in detail in the following sections.

The remainder of this paper is structured as follows: Section 2 introduces labelled transition systems and discusses the implications of using them to model behaviour (for both systems and devices). Section 3 discusses the correspondence (or lack of correspondence) between task actions and device actions. Section 4 discusses possible relationships for comparing the two models. Section 5 discusses a small example, which is indicative of the problems which this relationship helps highlight. Section 6 summarises the conclusions of this paper.

Notation	Meaning
$q \xrightarrow{\mu_1 \dots \mu_n} r$	$\exists q_1 \dots q_n \in Q \mid q \xrightarrow{\mu_1} q_1 \xrightarrow{\mu_2} q_2 \rightarrow \dots \rightarrow q_{n-1} \xrightarrow{\mu_n} r$
$q \Rightarrow^\varepsilon r$	$q \equiv r \vee \exists n \geq 1 \mid q \xrightarrow{\tau^n} r$
$q \Rightarrow^\mu r$	$\exists q_1, q_2 \in Q \mid q \Rightarrow^\varepsilon q_1 \xrightarrow{\mu} q_2 \Rightarrow^\varepsilon r$
$q \Rightarrow^{\mu_1 \dots \mu_n} r$	$\exists q_1 \dots q_{n-1} \in Q \mid q \Rightarrow^{\mu_1} q_1 \Rightarrow^{\mu_2} \dots \Rightarrow^{\mu_{n-1}} q_{n-1} \Rightarrow^{\mu_n} r$
$q \not\Rightarrow^{\mu_1 \dots \mu_n}$	$\nexists r \in Q \mid q \Rightarrow^{\mu_1 \dots \mu_n} r$
out(q)	$\{\mu \in A \mid \exists r \in Q \bullet q \Rightarrow^\mu r\}$
Tr(q)	$\{\sigma \in A^* \mid \exists r \in Q \bullet q \Rightarrow^\sigma r\}$

Table 1. Some notation for describing a LTS.

2. Modelling devices and tasks with LTS

Labelled Transition Systems (LTS) are a very general and elementary framework for modelling behaviour. The LTS discussed here model systems that interact with an external environment by accepting symbols (its labels) and that are capable of performing internal (hidden) actions which the external environment cannot observe. LTSs provide the underlying semantic model of higher level languages for the specification of concurrent systems, like LOTOS, ESTELLE, or even some variants of Petri Nets, which have been used quite extensively for the specification of interactive systems. LTS are discussed briefly in order to draw attention to the trade-offs of using them to model interaction.

In the framework of LTS, a system is described using the notions of a global state and indivisible actions (external or internal) that cause state transitions. The specification of the global states is not discussed here, as in practice the LTS will result from an interpretation of a process algebraic specification as mentioned earlier. What is most significant for this discussion is the set of actions that the LTS specifies.

The constitution of the set of actions determines the abstraction level of the specification and defines the perspective of the model. For example, these actions could concern only input to a device, e.g., the models of the vending machines studied extensively by Hoare (1985), or it could concern both input and output interactions, e.g., Paternó and Faconti (1992), Markopoulos (1997). In this discussion we shall assume the most general case where any interaction between user and system can be modelled as an externally observable action of the respective LTSs. Any user action can be modelled by the LTS: whether the model describes a procedure from a manual, the result of a cognitive task analysis, or simply a designer's intended user activity. While, the meaning of the specification will be different in each case it is assumed here that an unambiguous model of the sequencing of user actions can be obtained and represented as a LTS.

Definition. A Labelled Transition System is a tuple $(Q, A, \xrightarrow{\mu}, q_0)$, where Q is a countable set of states, A is a countable set of elementary actions, $\xrightarrow{\mu}$ is a set of binary relations on Q , indexed by $\mu \in A \cup \{\tau\}$, representing transitions between states, and q_0 is the initial state.

Table 1 summarises some notation which is used in this paper. A relation $\xrightarrow{\mu}$ between two states, with $\mu \in A$, describes the execution of an elementary externally observable action μ . If $q_1, q_2 \in Q$, then $q_1 \xrightarrow{\mu} q_2$, indicates that when the system is in state q_1 it can perform an action μ and reach state q_2 . The symbol τ is used to denote a hidden action that effects a 'silent' transition $q_1 \xrightarrow{\tau} q_2$. Table 1 also introduces notation for abstracting away from hidden actions and for representing sequences of externally observable actions specified by the LTS. Relation \Rightarrow^μ , with $\mu \in A$, denotes a visible transition possibly preceded and followed by a sequence of internal actions. Finally, table 1 defines the set of traces $\text{Tr}(q)$ of a labelled transition system and the set of actions $\text{out}(q)$ which may follow a state $q \in Q$.

The assumptions we make by choosing LTS as representations of user and system are the following:

- Behaviour is modelled by actions which are represented as transitions of the LTS. There is no notion

of direction in actions, e.g., a distinction between input and output, nor is causality represented, e.g., a stimulus-response distinction. Actions represent elementary components of activity which are modelled as atomic entities. The meaning attached to user actions is dictated by the theoretical assumptions embodied in the representation which are outside the scope of this paper. For example, they could correspond to basic tasks of ETAG (De Veer et al 1990), simple tasks of GOMS (Card, Moran and Newell 1983), task actions of TKS (Johnson 1992), etc. The meaning of device actions is also dictated by higher order modelling concerns: they may concern only input-output events, a model of the input language will only deal with user input, an architectural model will describe the communication between software components of the user interface (e.g., Markopoulos 1997, Paternó and Faconti 1992).

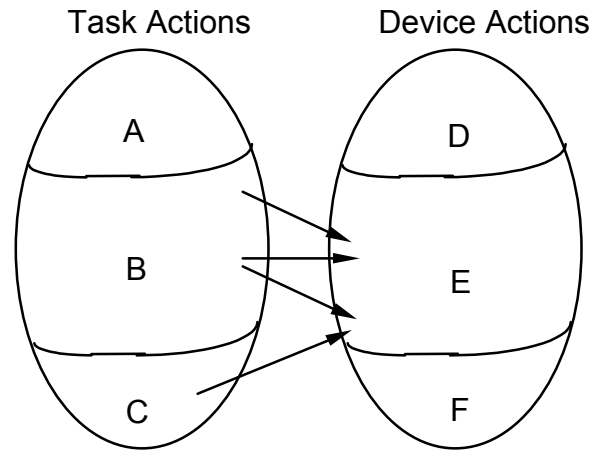
The present discussion makes no assumption regarding the abstraction level of the actions. This abstraction level can range from the superficially abstract, where one action can be 'interact', to the most detailed study of interaction at a physical and perceptual level, e.g., 'lift finger', 'press', 'hear the click of the mouse', etc. Such descriptions may themselves be of interest: for example to specify physical interaction devices (Accot, Chatty and Palanque 1996), to classify them, or at the more abstract side of the spectrum to provide an account of some activity of a very large scale (Johnson 1995).

- LTS are non deterministic. A non-deterministic system model means that a single interaction with the environment can give rise to different behaviours. The user may perform an action and may decide to do the subsequent tasks without the 'observer', in this case the interactive system, being able to tell how this decision is reached. A device model may seem non-deterministic to its user as the internal workings of the system are not observable to the user.
- A LTS can be infinite and non convergent. This may be appropriate for modelling computer systems where finiteness may only result from limited system resources. User behaviour is not infinite and non-convergent, although representations of user tasks in higher level languages which employ recursion iteration may easily result in non-finite interpretations in LTS.

The scope of a LTS is the temporal aspects of system and user behaviour it models. It does not model structural aspects of the models, e.g., decomposition relations, containment relations, architectural relations, etc. The scope of the present paper is thus limited to discussions of temporal ordering of user and system activity. By modelling only events we do not model status explicitly. Concerning interactive systems there are several limitations to this which are discussed by Dix and Abowd (1996). More specifically, an event based model cannot model integrity constraints/invariants that must hold at all times between state components. Such a relationship can be modelled by intensional means, e.g., by a series of notification and update events.

3. Correspondence of task and device actions

4. Consider a task and a device modelled by two LTSs. Interaction is modelled, rather abstractly, as the interaction of the two models excluding any other influential factors, such as, other users, models of organisations, etc. Clearly, these are very important factors in designing interaction, but the aim here is only to examine the relationship between task and device behaviour specifications.
5. Figure 1 illustrates the sets of task and device actions TA and DA respectively. These are partitioned into subsets to reflect their correspondence or the lack of it. The subsets listed below can be identified.
6. A: Some task actions represent user behaviour which the device has no way of observing, e.g., facial expressions or sentiments of the user are normally not known to the device. This could represent task activity not supported by the device studied. To the device these are non-deterministic internal actions, so when comparing task to device models these are hidden.
7. B: Some task actions are directly supported by the device. While it is possible that at different interaction contexts a single task action is mapped to different device actions, it is desirable that task actions are mapped to device actions consistently over time. Problems will arise if that is not the case.
8. C: This subset includes task actions which concern directly the operation of the device rather than the performance of the task independently of the device. Their existence depends on the modelling approach taken: does the task describe how to operate the device or does it abstract away from device related issues?
9. D: Some device actions are not observable to the user. They represent non determinism resulting from internal system behaviour, e.g., communication between different architectural components, communication with a network of computers, etc.



A: internal task actions D: internal device actions
 B: task actions served by the device E: device actions supporting task actions
 C: device specific task actions F: observable device actions not related to the task

Figure 1. The correspondence of task actions to device actions.

10. E: These device actions are directly relevant to the performance of the task in question.
11. F: These device actions are observable to the user but they do not concern the specific task. They are not observed as non-determinism but rather as non relevant detail.
12. The constitution of TA and DA embodies significant modelling decisions. Actions in set A may model aspects of cognitive activity, e.g., decisions, recall, etc. Alternatively they may model externalised behaviour of the user which the device has no way of perceiving, e.g., the user is distracted by environmental stimuli, the operation of the device is situated within a wider context which is not in the scope of the device model, etc. Modelling these aspects of interaction can be approached from a range of perspectives which are not excluded by the simple model discussed here. Contrasting sets A and F illustrates the non symmetric treatment of the task and device models. While the task model describes a particular aspect of user behaviour it is implicitly assumed in the description above that the user's perception of the device behaviour extends beyond the scope of the task. Finally, the choice of actions identifies an abstraction level for the description of the task and interface models. In a task based design (Wilson and Johnson 1996) this level of abstraction is determined by a task analysis. In the present discussion a consistent abstraction level for task and device actions is pre-supposed: it must either be the result of design or, in the case where existing systems are modelled, it is a necessary quality of the models constructed.
13. **Example 1.** Consider the task of editing a graph using a simple drawing tool. The graph is constructed by connecting lines and nodes. The tool is general purpose so when moving a node, all the lines connected to it have to be adjusted manually so that they connect to it also at its new position. This suggests a mismatch between the levels of abstraction of the task and that supported by the device. A purpose-specific tool that supports the task of graph editing will allow the user to move the node with its edges following.
14. ■
15. **Example 2.** Consider the task of film editing and suppose a tool is provided for selecting which parts of the film will be played, e.g., by some hierarchical organisation of data. This specifies pointers to the beginning and end of film-segments to be played. If the film-editor wishes to remove a particular sequence from the film they have to find to specify this indirectly. They must determine where the previous segment played ends and where playing will resume. In this case the level of abstraction is appropriate but the choice of actions supported is not.
16. ■
17. ■

18. **Example 3.** Consider the task of writing a document on a word processor. The task action could be to move a paragraph to a position in the text, to delete it or to insert a copy of a paragraph. These interface can support all these actions by 'cut' and 'paste' operations. The difference in the abstraction level of task actions and device actions is overcome by the user who translates the task actions 'move', 'delete', and 'copy' to appropriate sequences of 'cut' and 'paste' operations. In this case the user adapts their activity to what is supported by the device. A more accurate task model would include the elementary task actions of 'cut' and 'paste' instead of the composite task action 'move'.
19. ■
20. The examples above illustrate how the correspondence between task actions and device actions can capture the idea of designing at the appropriate level of abstraction and can gauge whether the appropriate 'vocabulary' of actions is provided. Engineering this correspondence will help design a system which will support the task described by the task model, say using a task based design approach (Wilson and Johnson 1996). Such a correspondence may need to be defined explicitly in order to assess an existing system.
21. The correspondence of task actions to device actions is now discussed more closely. For both models the actions in sets A and D are significant. However, in modelling their interaction these two subsets must be considered internal (hidden) transitions and they can be modelled by the silent transition τ . This means that they induce non deterministic behaviour. A fine point has to be made here. The non determinism resulting from hiding task actions may be observed at 'run' time, as opposed to being a 'don't care non-determinism'. This latter type of non-determinism is present in the specification of software or of international standards where the specifier leaves open the option to the implementers to support their own choice of some specified behaviours. Non-determinism and its interpretation are significant for modelling the relationship between the two models.
22. The modeller may choose to model some components of the task knowledge which are specific to the manipulation of the interaction device and which enable the users to perform their tasks (then $C \neq \emptyset$). Alternatively, the modeller can describe an idealised task model which does not describe such components (then $C = \emptyset$). In such a case they are considered as internal detail and so should the corresponding device actions, so the comparison with the task model will reflect only on the feasibility of performing the idealised task ignoring how complicated the interaction may be. Alternatively, they could all be modelled, but considered as internal detail. If such actions are modelled by the task model then the corresponding interactions must be considered to belong to the subset E of DA. A more accurate model of interaction results if task actions C are explicitly mapped to device actions E. Including such task actions and their corresponding interaction in the formal representations and their comparisons gives a better handle for comparing the two and a stronger task requirement.
23. Consider now the case where set A is empty. This would mean that all task actions are observable by the device. Unless an extraordinarily perceptive/affectionate machine is assumed this would imply that the task simply describes how to operate the device.
24. **Example 4.** Consider a device model of an electronic chess board. If $A = \emptyset$ the task can not be a description of how to play chess and win the game. At best it will concern the rather less interesting task of how to make legal moves.
25. ■
26. The cases where B or E are empty are contrived. If D is empty this means that the system is considered deterministic. A finite state machine could be a sufficient model of the device rather than a LTS. This implication of $D = \emptyset$ is discussed further below; in the general case though it will be assumed that all sets A-F are non-empty.
27. Consider now the relationship between task actions and device actions. This can be modelled by
28. $R = TA \leftrightarrow DA$
29. where \leftrightarrow denotes a binary relation. R models a competent user who can relate task actions to the interactions supported by the device. This competence can not always be assumed, it is however desirable to support it by design. In the literature of human-computer interaction the issues embodied by relation R have been discussed by Moran (1983) as the mapping of external tasks to internal tasks. Wilson and Johnson (1996) suggest that interactions should be supported at a level of abstraction defined by the task model. In the context of the comparison of the device and task models which is discussed here, the validity of the analysis relies on the success of the designer in making the relation R explicit to the user. An explicit specification of R may indeed be a helpful guide to the design activity, but this possibility is not discussed further. The properties of R itself can give interesting insights into

the properties of the design.

30. $\bullet R: B \cup C \rightarrow E \cup F$ is a function. The device supports fully all task actions.
31. $\bullet R: B \cup C \rightarrow E \cup F$ is surjective (or according to the definitions above $F = \emptyset$). The device does not support task actions outside those of the task model.
32. $\bullet R: B \cup C \rightarrow E \cup F$ is injective. This implies that there is no redundancy; each task action is supported by distinct device actions.
33. **Example 5.** Consider a word processor and the task of typesetting. In this case R is a function if all actions for the typesetting task are supported by the system. Normally, R is not surjective as there will be functionality that the user is not aware of or has not yet discovered. R is normally not injective since there are multiple ways of achieving each task action. For example, setting the margins of a document can be done by a dialogue box or by direct manipulation of the on-screen representation of borders. In general, most interactive systems support several alternatives for most task actions, e.g., menus, keyboard shortcuts. This provides added flexibility for task performance.

34. ■

35. Specifying the relation R

36. Let $TM[TA]$ be short for a LTS $(TM, TA, \xrightarrow{\mu}, tm_0)$ and let $DM[DA]$ for $(DM, DA, \xrightarrow{\mu}, dm_0)$. R can be specified as a two-step renaming of TA and DA to a set of labels L as follows:
 $TM_H = TM[H(TA)]$ where $H: TA \rightarrow L$
37. $DM_G = DM[G(DA)]$ where $G: DA \rightarrow L$
 $H(B \cup C) = G(E), H(A) = \{x\}, G(D) = \emptyset, G(F) = F$
38. The correspondence between task and device actions is specified in two stages as $R = H; G^{-1}$ where $;$ denotes the composition of binary relations. Combined, the two mappings of labels from TA to a set of labels L and then to a set of labels DA have the effect of matching task actions to device actions. All actions in set A are mapped to a reserved label x , so that they can be treated uniformly in further analysis.

4. Comparison of user and device behaviours

39. Testing is a formal framework for comparing systems whose internal structure is not known, with respect to their responses to sequences of actions offered by their external environment. A wide range of formal relationships between processes modelled by LTS have been interpreted within a testing framework (De Nicola 1989). A long standing debate has tried to establish which is the most appropriate way to model the relationship of an implementation to its specification. This issue is discussed here in the context of relating task and device models.

4.1 Trace pre-order and equivalence

A fundamental relation between systems is the trace pre-order (and equivalence). A trace of a process describes a sequence of observable interactions with its environment. It is a prefix-closed set. Trace pre-order and equivalence correspond to the inclusion and equality of two formal languages. Comparing two processes with respect to their traces ignores their non-determinism and is too weak a relationship for comparing the dynamic behaviour of most interactive systems. A classic example from process algebra literature is illustrated in figure 2. The computation tree to the left represents a behaviour which after an action a offers a choice between two interactions b and c . The tree to the right describes a system which on offering interaction a commits itself to a subsequent behaviour that will offer b or c only. These two systems are trace equivalent. However, if they are considered to model two interactive devices, a reasonable observer would easily distinguish between two. This is not so clear-cut in the case of modelling users. This is illustrated with the example that follows.

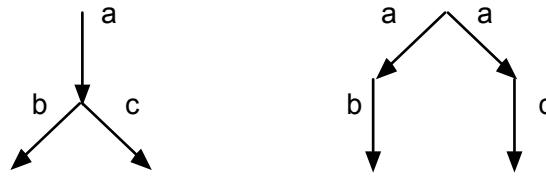


Figure 2. Two trace equivalent behaviours which are not behaviour equivalent.

Example 6. Consider an interaction model of a text editor with the behaviour modelled by the computation tree to the left of figure 2, where *a* represents the action 'start the spell checker', *b* stands for 'type-in correction', and *c* stands for 'accept recommended spelling change'. After being started the spell checker will let the user choose whether to accept a recommendation or type-in their own correction. On being started, the spell-checker modelled by the computation tree to the right, will leave no choice whether to accept a spelling change or type the correction, but would internally decide which action to offer to the user. Clearly, the comparison between LTSs modelling the device should reveal this difference. However, the two computation trees of figure 2 illustrate string equivalent behaviours.

Consider now that the two behaviours illustrated in figure 2, describe the task of using a spell checker and that the indicated actions have the same meaning as above. The first behaviour is clear. The user will start the spell-checker and then decide what to do: type-in a change or accept a recommendation. The second behaviour introduces a non-deterministic choice. This can be interpreted in two ways: the user will have decided whether or not to accept the recommendation of the spell checker before starting it, or, the user does not care which type of interaction is offered as long as there is at least one available. This type of 'don't care non determinism' sets weaker requirements upon the device.

Early work in representing tasks formally relied on grammars to represent task action sequencing (CLG and TAG). This type of linguistic model identifies trace equivalent interactive systems. Further, as is for example the case with CLG, and similar top down development methods, it has been suggested that a device model is a more concrete specification of the syntax of interaction. In such a modelling approach, the relationship between task and device models is that of an abstract to a concrete syntax. However, the example above suggests that trace pre-order is not appropriate for comparing interactive device specifications and task specifications. The reason for this is that comparing traces of LTS is not sensitive to the non-determinism that, as was illustrated above, characterises the behaviour of both the user and the device. Further we note that the meaning associated to the LTS is radically different for the device and the task models.

4.2 Bisimulation pre-order and equivalence

Another plausible approach to comparing behaviours is bisimulation (Milner 1989).

Definition. A relation $R \subseteq Q \times Q$ is a (strong) bisimulation pre-order \leq_{bis} on a set of states Q of a LTS iff $\forall (P, S) \in R, \forall \mu \in A \cup \{\tau\}$ the following holds:

$P \xrightarrow{\mu} P' \Rightarrow \exists S' \mid (P', S') \in R \bullet S \xrightarrow{\mu} S' \mid R$ is a bisimulation equivalence if $P \leq_{\text{bis}} R$ and $R \leq_{\text{bis}} P$. Two LTSs are called strong bisimulation equivalent if there exists a strong bisimulation relation R relating their initial states, i.e. $(p_0, s_0) \in R$. By substituting \Rightarrow^{μ} in the place of $\xrightarrow{\mu}$ in the definition above we obtain the weak bisimulation relation and the weak bisimulation preorder, denoted as $P \leq_{\text{weak}} Q$.

The practical advantage of strong and weak bisimulation, and the range of its variants that have been proposed in the literature, is that they can be verified by model checking, e.g., the CAESAR/ALDEBARAN toolset (Fernandez et al 1992). However bisimulation can be too discriminating. The following example is adapted from De Nicola (1988), here set in the context of device and task models.

Example 5.

Consider the two execution trees illustrated in figure 3. These trees represent behaviours which are not bisimulation equivalent. Suppose they model a spell checker where *a* stands for 'start the spell checker', *b* for 'find spelling error', *c* for 'accept recommended change' and *d* for 'type correct spelling'. The system to the left will 'decide' non-deterministically after being started whether it will let the user accept the

Error! Reference source not found.

Error! Reference source not found.

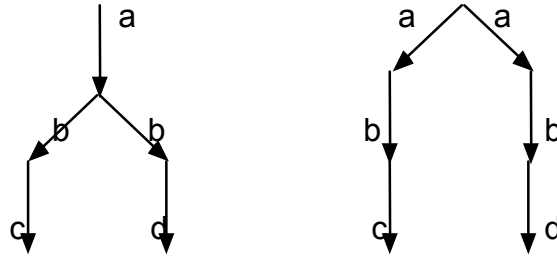


Figure 3. These behaviours are not bisimulation equivalent, but they cannot be distinguished by an external environment interacting with them.

spelling change or type-in the correct spelling. The system to the right will have committed non-deterministically to one of the two when it is started. An external user will find it impossible to discriminate between the two systems.

Consider now a user whose task is to use the spell-checking program and let a , c and d be as above, and let b now stand for ‘read indicated spelling error’. Both task descriptions will have the user committed to a decision of accepting a spelling change or of typing in the correct spelling either immediately before or after starting the spell-checker. For the machine though it is impossible to distinguish between the two users.

■

4.3 Testing pre-order and equivalence

For a large class of systems (finite, non-divergent systems) a large class of pre-orders and their corresponding equivalences coincide with the notion of the *testing pre-order* (res. *equivalence*) which is discussed in this section. This result (De Nicola 1988) shows that testing pre-order is a fundamental relation, so it is worth a closer look. The definitions by De Nicola are adapted below.

Definitions

Let P, Q range over the set of all LTS \mathcal{P} over an alphabet $A \cup \{\tau\}$ of elementary actions. Consider also a set of observers \mathcal{E} modelled by a set of LTSs over an alphabet $A \cup \{\tau, w\}$ where w is a reserved success event that is not in A .

Given two transition systems $P \in \mathcal{P}$ and $E \in \mathcal{E}$ with initial states p_0 and e_0 respectively, a *computation* from $\langle p_0, e_0 \rangle$ is a finite or infinite sequence of pairs of states $\langle p_n, e_n \rangle$ where:

- i. $\langle p_n, e_n \rangle \xrightarrow{\tau} \langle p_{n+1}, e_{n+1} \rangle$ if $(p_n \xrightarrow{\tau} p_{n+1} \text{ and } e_n = e_{n+1})$ or $(e_n \xrightarrow{\tau} e_{n+1} \text{ and } p_n = p_{n+1})$
- ii. $\langle p_n, e_n \rangle \xrightarrow{\alpha} \langle p_{n+1}, e_{n+1} \rangle$ if $p_n \xrightarrow{\alpha} p_{n+1}$ and $e_n \xrightarrow{\alpha} e_{n+1}$

and if the sequence is finite and $\langle p_k, e_k \rangle$ is its final element.

We can now define the following predicates:

P *may* satisfy E if there exists $s \in A^*$ such that $\langle p_0, e_0 \rangle \xRightarrow{s} \langle p_n, e_n \rangle$ and $w \in \text{out}(e_n)$

P *must* satisfy E if for every computation $\langle p_0, e_0 \rangle \xrightarrow{\mu_1} \langle p_1, e_1 \rangle \xrightarrow{\mu_2} \langle p_2, e_2 \rangle \rightarrow \dots$ there exists $n \geq 0$ such that $w \in \text{out}(e_n)$.

These definitions help define the following pre-orders (from which we can define the corresponding equivalences).

Testing pre-orders.

$P \leq_{\text{may}} Q$ if for all $E \in \mathcal{E}$ P may satisfy E implies Q may satisfy E

$P \leq_{\text{must}} Q$ if for all $E \in \mathcal{E}$ P must satisfy E implies Q must satisfy E

$P \leq_{\text{test}} Q$ if $P \leq_{\text{may}} Q$ and $P \leq_{\text{must}} Q$

■

De Nicola (1987) also shows that: $P \leq_{\text{may}} Q$ iff $\text{Tr}(P) \subseteq \text{Tr}(Q)$ which relates testing to the trace pre-order discussed before. Testing pre-order and equivalence are weaker requirement than weak bisimulation

defined earlier and they are implied by it. Unfortunately, model checking tools do not support the verification of testing pre-orders or equivalences, or of other relations that have the same discriminating power between LTSs. The definition of testing pre-order suggests that a testing methodology could be developed to test, in our context, the relationship of the device and the task models. A generic practical testing approach is outlined by Quemada, Azcorra and Pavón (1993). An important concern for choosing the relationship to check is the test generation problem: can a set of test cases be derived from the specification? In the general case there is not a finite algorithmic solution to this problem for testing pre-order. For this reason, research on formal testing of system behaviours has focused on a weaker but more versatile relation for which the test generation problem has been solved. This relation is called the conformance relation and it is discussed in the following section.

4.4 The conformance relationship

Definition. Let P and Q be processes (modelled by LTSs) and let \mathcal{L} be the set of all possible labels for all LTSs: Then

$P \text{ conf } Q$ if

$$\forall \sigma \in \text{Tr}(Q), \forall A \subseteq \mathcal{L} \bullet \text{if } \exists P' | \forall \mu \in A \bullet P \xrightarrow{\sigma} P' \not\xrightarrow{\mu} \text{ then } Q \xrightarrow{\sigma} Q' \not\xrightarrow{\mu}$$

This formula is read as follows: If P can perform some trace s and then behave like a process P' and if Q can perform the same trace s and then behave like Q' , then the following conditions are required: whenever P' refuses to perform any action from set A , then Q' must also refuse every action in A . Thus, $P \text{ conf } Q$ means that testing the traces of Q against the process P will not lead to deadlocks that could not occur with the same test performed with Q itself.

■

The following points are interesting to note about the conformance relationship:

- Conformance is not transitive and therefore as an implementation relationship it cannot be used as the basis of a refinement process (Brinksma 1989).
- Conformance is the difference between trace and failure pre-order (Lotosphere 1990).

$$DM \leq_{\text{trace}} TM \text{ and } DM \text{ conf } TM \Leftrightarrow DM \leq_{\text{test}} TM$$

- Conformance is testable (Brinksma 1989).

The last statement means that from a given specification Q it is always possible to derive a set of tests that will succeed with a given implementation iff that conforms to Q . This set of tests is specified by a process which is called the *canonical tester* of Q . More formally, let Q be a process and \mathcal{Q} be the set of all possible processes. The canonical tester of Q is a process $T(Q)$ such that

$$\text{Tr}(T(Q)) = \text{Tr}(Q) \wedge \forall P \in \mathcal{Q} | P \text{ conf } Q \text{ iff } \text{Succ}(T(Q), P)$$

The predicate $\text{Succ}(T(Q), P)$ means that a testing environment whose behaviour is specified by $T(Q)$ will always test P successfully. It has been shown (Brinksma 1989) that for all LOTOS process specifications there exists a canonical tester. The generation of the canonical tester is supported by LOTOS tools, as for example the COOPER component of the LITE tool-set (Mañas 1995).

A formal expression of the relationship between task and device behaviours can now be specified as follows:

$$DM_H \text{ conf } TM_G$$

This expression means that a user interacting with a device, that behaves as DM , will not reach an impasse when performing a task, described by TM , provided there is a correspondence of task actions and device actions as discussed earlier. DM may specify behaviours which are not specified in TM . In other words, the task model is seen only a partial (incomplete) specification for the device. The conformance requirement formalises an intuitive definition of the preservation of task sequencing which is used as a heuristic guideline in task based design (Wilson and Johnson 1996). There it is suggested that in progressing from a task model to a device model it should be ensured that all task action sequences specified in the task model are possible, although alternative sequencing can be introduced in the interface which can also support other tasks than those specified in the task model. When modelled formally this requirement is predicated

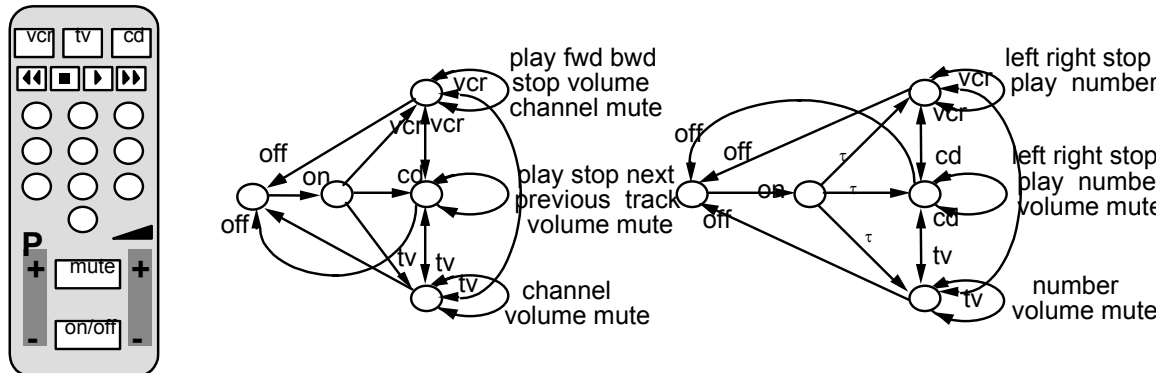


Figure 4. An ‘all-in one’ remote control for a VCR, a CD and a TV. Imaginary task and device models are modelled by the LTSs illustrated. Note, the different actions offered by each and that ‘mute’ is required in all ‘modes’ of the task model but it is not available on the VCR mode of the device model.

by the correspondence of actions described by relation R of the previous section.

A practical approach for testing a device model against a task model can now be outlined:

- Specify TM and DM and the relation R.
- Produce the ‘canonical tester’ $T(TM_G)$, for the modified task model TM_G . The specification of the tests for the device is given by the behaviour :

$$CT = \text{hide } \{x\} \text{ in } T(TM_G)$$

Here, all actions of the task model which do not correspond to interactions with the user device are hidden.

- From CT produce a set of finite tests T_i , with $i:1..n$.
- Test the device against T_i , with $i:1..n$.

A device model can be tested against the tests T_i using a test-bed tool. In earlier work we used the LOTOS specification language to specify device and task models which were tested with the LOLA component of the TOPO tool-set (Quemada, Pavón and Fernandez 1989). Alternatively the tests can be performed at varying levels of realism using the implemented software; this is an informal testing. The advantage of testing is that the abstract tests can be refined gradually to shadow the development of a system so that the analysis is not tied to an abstract level remote from the development activity. However, there are serious limitations to the practicality of this approach. The problem of generating a canonical tester from a specification has been solved only for basic LOTOS processes (LOTOSPHERE 1990). The algorithm that computes the canonical tester may not terminate when the specification contains infinite recursion. So for practical reasons it may be better to restrict the problem to modelling and testing finite tasks, i.e. task specifications without recursion. Even if the algorithm terminates producing a process specification of the canonical tester, this may be too large to be used in practice as the basis of testing. Also, given any non-deterministic system testing cannot be exhaustive by its definition. Generating a useful collection of finite tests is thus an important practical problem that the specifier is currently expected to solve ‘manually’.

5. Example

This example is adapted from Andre and Degani (1997) who discuss the notion of modes in interactive systems. The example concerns a remote control that can operate three systems: the TV, the VCR and a CD player. For the sake of the example, let the LTS to the left represent the task model of operating the system and the one to the right the device model. In this task model the user selects one of these systems directly after switching on the device. Further the user expects to change the volume or to ‘mute’ the output

while operating any of the devices. When it is switched on it will non-deterministically choose one device to operate (e.g., the last device that was operated before switching it on or it could always be the same device - this LTS leaves this open to interpretation). This device is 'moded'. This remote control does not allow manipulation of the volume or muting when in the VCR 'mode'. Rather the user must switch to the TV to adjust/mute the volume. The example here does not intend to suggest a design solution for making the user aware of the operating mode, but rather to illustrate the mappings and the relationships discussed in the paper.

Figure 4 illustrates how different task actions are mapped to the same interface action. For example, moving the video tape forward, and moving to the next track on a CD are supported by the same interface action. The numeric pad may support choosing a track on the CD and also selecting the channel on the TV or the VCR. The sets of actions discussed in section 2 will now be as follows with the obvious mappings between B and E:

- $A = \emptyset$. So other activities of the user or how they decide which device to switch on are not modelled in the task.
- $B = \{\text{on, off, vcr, play, fwd, bwd, stop, volume, channel, mute, next, previous, track}\}$.
- $C = \emptyset$. A very good fit of the device to the task is assumed here. A counter example would be if there were extra operations, say to switch on the remote controller separately from the device.
- $D = \emptyset$. The internal workings of the device are not modelled, e.g., the communication with the systems, or how it determines which system to activate after it is switched on.
- $E = \{\text{on, off, left, right, stop, play, number, volume, mute}\}$.
- $F = \emptyset$. The device does not support any more tasks. A counter example would be to support operations the user does not know how to do, e.g., tuning the TV.

For these two systems the IM does not conform to the task model. This can be established by testing the interface against the following test (the operator ; denotes action succession).

$t = \text{on; vcr; mute; success; stop}$

This test must fail at all runs of the test. However, if the system was modified so that mute and volume are offered in all modes we would have successful verdict in this test (must succeed). The interface would then conform to the task model. This modification would also ensure weak bisimulation pre-order between the two models. TM and DM would still not be weak bisimulation equivalent, since when the system is switched on the TM does not assume a device to be selected by default.

6. Conclusions

This paper has discussed the comparison of labelled transition systems representing task and device models. The term task model and device model were used here as place holders for a description of the temporal behaviour of users and systems. These models can be populated by appropriate task and system modelling approaches. As a modelling framework labelled transition systems is quite appropriate for modelling the temporal behaviour of interactive systems. The labelled transition systems can be produced by the interpretation of more abstract formal languages and specifications which embody theoretical models of users or architectural considerations for the device. Clearly, a host of interactive systems design problems are not covered by modelling actions and their sequencing. Further, the validity of task modelling approaches, i.e., whether they do predict the activity of users has been questioned (see for example, Draper 1993).

However, even this simple model of interaction helps make explicit some interesting design issues. The first is the correspondence of task actions to interface actions. This mapping cannot be considered a constant; rather it is an essential component of interactive system design. The definition of the sets of actions for users and devices is an important modelling issue on its own right which was not touched upon in this paper. While it may not always be cost effective to enumerate all task and device actions and then

make this mapping explicit, the discussion shows that the mapping itself, the choice of actions and abstraction level at which to model tasks and devices are salient design problems.

The presence of non-determinism in both task and device models was identified. This non-determinism suggests that simple comparison of the traces of two LTS representations is not sufficient to compare them. Testing was suggested as a viable means of comparing interactive devices and tasks in the cases where some component of their temporal behaviour can be considered to be non-deterministic to the other. Conformance testing was proposed as a practical relationship to establish in this context. The example illustrated how conformance gives a handle of at least one class of interaction design issues which broadly concern the modedness (conversely the flexibility) of interaction. This paper has extended an earlier discussion (Markopoulos, Johnson and Rowson 1997) on task conformance presented earlier in the formal framework of LOTOS and which related specifications of user tasks in the framework of TKS (Johnson 1992) and interactor based specifications of the system (Markopoulos 1997).

7. Acknowledgements

This research is funded by EPSRC, grant GR/K79796. Acknowledgement is due to CNRS/IMAG for the CADP toolset, to UPM University of Madrid, for the TOPO/LOLA toolset and to ITA, University of Twente, for the LITE toolset.

8. References

- Accot, J., Chatty, S. & Palanque, P. (1996) A formal description of low level interaction and its application to multimodal interactive systems. In Bodart, F. & Vanderdonck, J. (Eds.) *Design, Specification and Verification of Interactive Systems '96*, Springer (Wien), pp. 92-104.
- Andre, A. & Degani, A. (1997) Do you know what mode you're in? An analysis of mode error in everyday things. In M.Mouloua & Koonce, J.M., (Eds.) *Human-automation interaction: Research & Practice*, Mahwah, NJ:Lawrence Erlbaum, pp.19-28.
- Breedvelt-Schouten I.M., Paterno, F.D. & Severijns, C.A. (1997) Reusable structures in task models. In Harrison MD & Torres JC (Eds.), *DSV-IS'97: Design Specification and Verification of Interactive Systems*, Springer Wien, pp. 225-240.
- Brinksma E (1989) A theory for the derivation of tests. In van Eijk PHJ, Vissers CA & Diaz M (Eds.) *The Formal Description Technique Lotos*, Elsevier (North-Holland), pp. 235-247.
- Card, S.K., Moran T.P. & Newell, A. (1983) *The Psychology of Human Computer Interaction*. Lawrence Erlbaum Associates, New Jersey.
- De Nicola R (1989) Extensional Equivalences for Transition Systems, *Acta Informatica* Vol. 24, 211-237.
- Dix, A.J. & Abowd, G. (1996). Modelling status and event behaviour of interactive systems, *Software Engineering Journal*, 11, 6, pp.334-346.
- Draper, S (1993) The notion of Task in HCI. *InterCHI 93, Adjunct Proceedings*, pp 207-208.
- Fernandez, J.C., Caravel, H., Mounier, L., Rasse, A., Rodríguez, C. & Sifakis, J. (1992). A toolbox for the verification of LOTOS Programs. *ICSE '92, 14th Int. Conference on Software Engineering*, Melbourne, May 1992.
- Gerrit C. van der Veer, Diederik Broos, Keneth Donau, Mark J. Fokke, Felix Yap, "ETAG - Some Applications of a formal representation of the user interface", *Human Computer Interaction, INTERACT '90*.

- Green M (1985) Report on Dialogue Specification Tools. In Pfaff GE, User Device Management Systems, Springer-Verlag, pp. 9-20.
- Hoare CAR (1985) Communicating Sequential Processes. Prentice Hall International.
- ISO (1989). Information Processing Systems-Open Systems Interconnection-LOTOS-A Formal Description Technique based on the Temporal Ordering of Observational Behaviour. ISO/IEC 8807, International Organisation for Standardisation, Geneva.
- Johnson CW (1995) The application of Petri-Nets to reason about human factors problems during accident analysis. In Palanque, P. and Bastide, R. (Eds.) Design, Specification, Verification of Interactive Systems '95, Springer Wien, pp. 93-112.
- Johnson, P. (1992). Human-Computer Interaction: Psychology, Task Analysis and Software Engineering, McGraw-Hill, London, 1992.
- Lotosphere (1990) A theoretical and methodological framework to conformance testing. Burmeister J & de Meer J (Eds.) project deliverable Lo/WP1.T1.3/N0006/V5, ESPRIT ref. 2304.
- Mañas, J.A. (1995). Getting to use the LOTOSphere Integrated Tool Environment (LITE). In Bolognesi T., Van De Lagemaat J. & Vissers C., Eds., LOTOSphere: Software Development with LOTOS, Kluwer, Netherlands, pp. 87-107.
- Mañas, J.A. (1995). Getting To Use The Lotosphere Integrated Tool Environment (Lite). In Bolognesi T., Van De Lagemaat J. & Vissers C., Eds., Lotosphere: Software Development With Lotos, Kluwer, Netherlands, Pp. 87-107.
- Markopoulos P, Johnson P & Rowson J (1997) Formal Aspects of Task Based Design. In Harrison MD & Torres JC (eds.), DSV-IS'97: Design Specification and Verification of Interactive Systems, Springer Wien, pp. 209-224.
- Markopoulos, P. (1997). A compositional model for the formal specification of user interface software, Ph.D. Thesis, Department of Computer Science, Queen Mary and Westfield College, University of London, March 1997.
- Milner, R., (1989). Communication and Concurrency, Prentice Hall International.
- Moran, T. (1981). The command language grammar: a representation for the user interface of interactive computer systems, International Journal of Man Machine Studies, 15, 3-50.
- Moran, T. (1983). Getting into the system: External and Internal Task Mapping Analysis, In Janda, A., Ed., 2nd Conference on Human Factors in Computing Systems, CHI '83, ACM Press, New York.
- Paternó, F. & Faconti, G.P. (1992). On the use of LOTOS to describe graphical interaction. In MONK A., Diaper, D. & Harrison, M., Eds, People and Computers VII, Human Computer Interaction, BCS-HCI Conference Proceedings, Cambridge University Press, 155-173.
- Quemada, J., Pavón, S. & Fernandez, A. (1989) State exploration by transformation with LOLA, Workshop on Automatic Verification Methods for Finite State Systems, Grenoble, June 1989.
- Quemada, J, Azcorra, A, & Pavón, S. (1993) Software Development with LOTOS. In Turner K (Ed.) Using formal description techniques - an introduction to Estelle, LOTOS and SDL, Wiley, pp. 345-373.

Error! Reference source not found.

Error! Reference source not found.

Wilson, S. & Johnson, P. (1996). Bridging the generation gap: From work tasks to user interface designs. In Vanderdonckt, J., Ed., 2nd International Workshop on Computer-Aided Design of User Interfaces, CADUI'96, Presses Universitaires de Namur, 77-94.

Error! Reference source not found.

Error! Reference source not found.

Author(s): **Error! Reference source not found.**

Title: **Error! Reference source not found.**
Error! Reference source not found.

Distribution

Director:	E.P.C. van Utteren	WL-01
Department Head	R.P.G. Collier	WL-11

Full report

Paul Shrubsole	Nat.Lab.	WL-01
Panos Markopoulos		
Aart Matsinger		
Armin Kohlrausch		
Berry Eggen		
Christian Bakker		
Duco Das		
Erik Moll		
Gerrit-Jan Bloem		
Joyce Westerink		
Hok Kong Tang		
Luc Geurts		
Maddy Janse		
Paul Kaufholz		
Reinder Haakma		
Rene Collier		
Vincent Buil		
Richard van de Sluis		
John de Vet		
Boris de Ruyter		
Herman ter Horst		
Leon van Stuivenberg		