A Formal Framework for Design and Analysis of Human-Machine Interaction

Sébastien Combéfis*, Dimitra Giannakopoulou†, Charles Pecheur* and Michael Feary†
*Computer Science and Engineering Department
ICT, Electronics and Applied Mathematics Institute
Université catholique de Louvain, Louvain-la-Neuve, Belgium
Email: {Sebastien.Combefis, Charles.Pecheur}@uclouvain.be
†NASA Ames Research Center
Moffett Field, CA 94035, USA
Email: {Dimitra.Giannakopoulou, Michael.S.Feary}@nasa.gov

***Abstract*—Automated systems are increasingly complex, making it hard to design interfaces for human operators. Human-machine interaction (HMI) errors like automation surprises are more likely to appear and lead to system failures or accidents. In previous work, we studied the problem of generating system abstractions, called mental models, that facilitate system understanding while allowing proper control of the system by operators as defined by the *full-control* property. Both the domain and its mental model have Labelled Transition Systems (LTS) semantics, and we proposed algorithms for automatically generating *minimal* mental models as well as checking full-control.**

This paper presents a *methodology* and an associated *framework* for using the above and other formal method based algorithms to support the design of HMI systems. The framework can be used for modelling HMI systems and analysing models against HMI vulnerabilities. The analysis can be used for validation purposes or for generating artifacts such as mental models, manuals and recovery procedures. The framework is implemented in the JavaPathfinder model checker. Our methodology is demonstrated on two examples, an existing benchmark of a medical device, and a model generated from the ADEPT toolset developed at NASA Ames. Guidelines about how ADEPT models can be translated automatically into JavaPathfinder models are also discussed.

***Index Terms*—Formal methods, HCI, Learning**

I. Introduction

Automated systems are increasingly complex, making it hard to design interfaces for human operators. Human-machine interaction (HMI) errors like automation surprises are more likely to appear and lead to system failures or accidents [1]–[3].

Human-machine interaction has been extensively studied for several years by researchers in psychology, human factors and ergonomics. Since the mid-1980s, researchers are investigating the use of formal methods to analyse behavioural aspects of HMI. The first results were focused on specific applications and on systems and their properties [4], [5]. The field then moved to more generic results using theories like graph theory, model-checking or theorem proving [6]–[8].

Recently, Degani et al. [9] formulated the problem of generating a user mental model for a system described as a finite state automaton, and presented an approach for addressing this problem. In this context, a mental model is not meant to capture a human cognitive model; rather, it is meant to capture the implicit and intended model of operation according to which the system developer designs the system. Like others, we make the assumption that such models are meant to be relatively simple, even though their corresponding systems may be large and complex. Complex systems are often accompanied with procedures that describe sequences of steps that need to be followed when performing some tasks. Procedures therefore represent explicit characterizations of specific parts of a mental model.

Automatic generation of mental models needs to be driven by the intended characteristics of the resulting models. The full-control property [10] formalizes the following notion of a correct mental model: a user following a full-control mental model will know at any point how to command or observe the system to achieve a goal, based on the history of previous commands and observations performed. Note that full-control requires a distinction between commands executed by the user on the system (inputs) and observations controlled by the system and just observed by the operator (outputs). Our previous work in this domain [10], [11] provided algorithms for checking the full-control property and automatically generating minimal full-control mental models.

The focus of our previous work was purely on the definition of a notion of correctness of mental models based on the full-control property, as well as the development of algorithms for automatic generation of minimal full-control mental models. In contrast, the current paper presents a *methodology* and an associated *framework* for using such algorithms in a practical setting to support the design and analysis of HMI systems. The proposed framework can be used for modelling HMI systems and analyzing models against HMI vulnerabilities. The analysis can be used for validation purposes or for generating artifacts such as mental models, manuals and recovery procedures; it can also be used to help redesign or update a system model to avoid detected vulnerabilities. Finally, the proposed algorithms for checking the full-control property can be slightly varied in order to analyse whether a system model appropriately supports the user tasks associated with it.

The design and analysis capabilities supported by our methodology are demonstrated on two examples, an existing benchmark of a medical device, and a model generated from the ADEPT toolset. The associated framework is implemented in the JavaPathfinder (JPF) model checker [12]. We discuss our early attempts at connecting our framework to the ADEPT [13] toolset. ADEPT is an HMI design environment that supports the high level design of HMI systems in a tabular fashion, as well as the automated generation of prototypes of a described system and its interface for experimentation and simulation. ADEPT supports some limited forms of analysis such as completeness and determinism of the specification. By connecting JPF and ADEPT we will extend the analysis capabilities provided by the ADEPT toolset. Both ADEPT and JPF have been developed at the NASA Ames Research Center (by different teams), which justifies our choice of these tools. However, the methodology and framework that we present could also be connected to other environments for the design and analysis of HMI systems.

A. Related Work

Campos et al. [5], [7] proposes a framework based on model checking to analyse HMI. The framework is based on systems modelled with interactors and properties of interest are described with the MAL logic. They define a set of generic usability properties [5], such as the possibility to undo. Their approach is thus based on properties which can be expressed with the MAL logic and checked with a model checker on the system and are specific and targeted to a precise usability property. Our approach uses a more generic definition of good systems, and is complementary to their analysis.

Thimbleby et al. [6], [14] use graphs to represent models. They study usability properties of the system by analyzing structural properties of graphs like the maximum degree and the value of centrality measures. In their approach, there is no distinction among actions and there is little focus on the dynamic aspects of the interaction.

Curzon et al. [8] use a framework based on defining systems with modal logic. Properties of the model are checked using a theorem prover. Similarly to Campos et al., properties of interest are more targeted to a specific usability property while our approach is more generic.

Navarre et al. [15] also developed a framework to analyse interactive systems. Their focus is on the combination of user task models and system models. We focus mainly on the system although in this paper we also present an approach for checking whether a user task is supported by a system model.

Bolton et al. [16]–[18] developed a framework used to help predicting human errors and system failures. Models of the system are analyzed against erroneous human behaviour models. The analysis is based on task-analytic models and taxonomies of erroneous human behaviour. All those models are merged into one model which is then analyzed by a model checker to prove that some safety properties are satisfied.

Bredereke et al. [19], [20] formalized mode confusions and developed a framework to reduce them. The formalization is based on a specification/implementation refinement relation. Their work is targeted on mode confusion while the work presented here is targeted to more general controllability issues.

Model-based testing has been used to analyse systems modelled as Input-Output Labelled Transition Systems (IOTS) [21]. The IO conformance relation (IOCO) is defined to describe the relationship between implementations and specifications. The IOCO relation states that the outputs produced by an implementation must, at any point, be a subset of the corresponding outputs in the specification. This is triggered by the fact that IOCO is used in the context of testing implementations. Outputs are similar to observations in our context. The full-control property defined in our work needs to consider commands (inputs) in addition to observations.

B. Motivation

Generating a minimal full-control mental model from a given system model helps to get a better understanding of the system. The full-control property captures the knowledge an operator needs to have about a system to be able to control it properly. Such a mental model can be used to build training materials such as user manuals [22]. Providing a system that the user can learn, minimizing her memory load, and allowing her to operate it without error is a desirable usability property [23].

In previous work, we proposed algorithms for automatically generating minimal full-control mental models. When a full-control mental model does not exist for a system, our algorithms provide a counterexample that exhibits a violation of the full-control property. In this paper, we discuss how such counterexamples can be analysed to understand the problematic behaviour of the system, and we propose ways of correcting the system to address such problems. More generally, the contribution of this paper is a methodology to perform analysis of dynamic aspects of human-machine interaction, based on our existing approaches for automatic generation of full-control mental models and checking of the full-control property. This work targets the system designer with a focus on demonstrating several uses of our algorithms and their outputs in a practical setting of analysing system models as well as their associated task models.

The remainder of the paper is organised as follows: Section II presents the modelling approach with all the necessary background. Section III is the core of the paper and presents the proposed interaction analysis formal methodology and framework. Section IV describes the implementation of the prototype tool. Section V demonstrates the methodology on two realistic examples. Section VI concludes the paper and provides some perspectives.

II. Modelling HMI

This section provides the background necessary for the work presented in this paper. The detailed formalization is presented in [10], [11]. The approach used is based on models as motivated in [24]. There are two models of interest in the proposed approach: the *system model* describes the detailed behaviour of the system and the *mental model* represents an abstraction of the system for the human operator[1].

A. System and mental models

System and mental models are modelled with enriched *labelled transition systems* (LTS) called HMI LTS, that are essentially graphs whose edges are labelled with actions. The difference with classical LTSs is that three kind of actions are defined:

1) *Commands* are actions triggered by the user on the system; they are also referred to as inputs to the system;
2) *Observations* are actions autonomously triggered by the system but that the user can observe; they are also referred to as outputs from the system;
3) *Internal actions* are neither controlled nor observed by the user; they correspond to internal behaviour of the system that is completely hidden to the user.

When interested in controllability properties of systems, and to avoid automation surprise errors, the distinction between commands and observations matters [9], [26].

Formally, HMI LTS are tuples $\langle S, \mathcal{L}^c, \mathcal{L}^o, s_0, \delta \rangle$ where S is the set of states, $\mathcal{L}^c$ and $\mathcal{L}^o$ are the sets of commands and observations respectively, s_0 is the initial state and $\delta : S \times (\mathcal{L}^c \cup \mathcal{L}^o \cup \{\tau\}) \to 2^S$ is the transition function. Internal actions cannot be distinguished by the user and are thus denoted with the same symbol τ, called the internal action. The set of observable actions comprises commands and observations and is denoted $\mathcal{L}^{co} = \mathcal{L}^c \cup \mathcal{L}^o$. In this paper, HMI LTS will simply be referred to as LTS.

When a transition exists between states s and s' with action a, that is $\delta(s, a) = s'$, we say that the action a is enabled in state s and we write $s \xrightarrow{a} s'$. A *trace* $\sigma = \langle \sigma_1, \sigma_2, \cdots, \sigma_n \rangle$ is a sequence of observable actions in $\mathcal{L}^{co}$ that can be executed on the system, that is $s_0 \xrightarrow{\sigma_1} s_1 \xrightarrow{\sigma_2} \cdots \xrightarrow{\sigma_n} s_n$. The set of traces of an LTS $\mathcal{M}$ is denoted $\mathbf{Tr}(\mathcal{M})$. Internal actions can also occur between actions of a trace, which is written $s \overset{\sigma}{\Longrightarrow} s'$ and corresponds to $s \xrightarrow{\tau^* \sigma_1 \tau^* \cdots \tau^* \sigma_n \tau^*} s'$, where τ^* means zero, one or more occurrences of τ. The set of commands that are enabled in a state s, denoted $A^c(s)$, corresponds to actions a such that there exists a state s' with $s \overset{a}{\Longrightarrow} s'$. The set of enabled observations, denoted $A^o(s)$, is defined similarly.

B. Full-control property

The *full-control property* captures that an operator has enough knowledge about a given system in order to be able to control it properly. That is, at each time during the interaction between the user and the system, she must know exactly what are the available commands on the system and must be aware of at least the observations that can occur.

Formally, a mental model $\mathcal{M}_U = \langle S_U, \mathcal{L}^c, \mathcal{L}^o, s_{0_U}, \delta_U \rangle$ allows the full-control of a given system $\mathcal{M}_M = \langle S_M, \mathcal{L}^c, \mathcal{L}^o, s_{0_M}, \delta_M \rangle$[2] if and only if:

$$\forall \sigma \in \mathcal{L}^{co*} \text{ such that } s_{0_M} \overset{\sigma}{\Longrightarrow} s_M \text{ and } s_{0_U} \xrightarrow{\sigma} s_U : \\ A^c(s_M) = A^c(s_U) \text{ and } A^o(s_M) \subseteq A^o(s_U) \qquad (1)$$

Intuitively, it means that for every trace σ, which can be executed both on the system and the mental model, after executing them ($s_{0_M} \overset{\sigma}{\Longrightarrow} s_M$ and $s_{0_U} \xrightarrow{\sigma} s_U$), the set of commands ($A^c$) that are enabled on the system and on the mental model are exactly the same and the set of observations (A^o) enabled according to the mental model contains at least the observations enabled on the system model.

With this approach, the user must always know all the possible commands which is a strong requirement. A weaker variant, where a user may not always know all the possible commands but only those which are relevant for the interaction she wants to have with the system, is considered in Section III-C below.

The full-control property indeed characterizes a conceptual model for a given system. All the behaviour of the system must be covered by the full-control mental model, and it should allow the operator to interact correctly with the system. That is ensured by the fact that the operator always knows what he can do on the system and what he expects to observe from it.

III. Interaction Analysis

Based on the full-control property, this paper proposes a framework and methodology to analyse systems from an HMI standpoint.

Two algorithms were developed in [10], [11], which are focused on the automatic generation of a minimal full-control mental model for a given system. The first is based on the definition of a bisimulation-based relation between the states of the system, stating which of them can be merged together because they can be handled similarly from the standpoint of the operator. The second uses a learning algorithm which iteratively builds mental model guesses. The algorithm relies on a teacher to answer whether proposed execution sequences must, may or cannot be part of the mental model. The teacher uses the system model to answer such queries.

As the main input is a model of the system, the analysis can be performed and used in different steps of the HMI design process. In the evaluation phase, the analysis gives feedback regarding whether the model of the system is controllable by an operator. If it is not controllable, the analysis provides an example of a problematic interaction. The analysis can also be used at the end of the design process, once the system has been validated, in order to build artifacts such as user manuals, trainings, etc.

[1]The mental model is commonly referred to as *conceptual model* [25] in the literature, that is, an abstraction of the system which outlines what the operator can do with the system and what she needs to interact with it.

[2]The subscript M for the system refers to *Machine* and the subscript U for the mental model refers to *User*.

A. Categorizing behaviour

The full-control property captures the behaviour of a given system that a user should know in order to be able to operate it without errors. Given a system model $\mathcal{M}_M = \langle S_M, \mathcal{L}^c, \mathcal{C}^o, s_{0_M}, \delta_M \rangle$, a trace $\sigma \in \mathcal{L}^{co*}$ can be put into one of three different categories. In other words, the set of traces of $\mathcal{M}_M$ can be partitioned into three sets: Acc (Accepted), Rej (Rejected) and $Dont$ (Don't care).

Let σ be a trace and a an action (command or observation):

1) $\sigma a \in Rej$ if (i) $\sigma \in Rej$ or (ii) $\sigma \in Acc$, a is a command and there exists an execution of σ where a is not enabled in the reached state. This first category highlights the fact that the user must always know exactly the available commands.
2) $\sigma a \in Acc$ if (i) $\sigma \in Acc$ and either a is a command which is enabled for all the states that are reached after the execution of σ or (ii) a is an observation that is enabled in at least one state reached after the execution of σ. This second category contains the behaviour of the system that the user must be aware of.
3) In the other cases, $\sigma a \in Dont$. This corresponds to two cases: (i) either $\sigma \in Dont$ or (ii) $\sigma \in Acc$, a is an observation and a is not enabled in any state reachable by σ. That last category reflects the fact that the user may expect an observation that will not necessarily occur in the system.

Traces from Rej *are forbidden* which means that they cannot be part of a full-control mental model. Traces from Acc *must be accepted* which means that any full-control mental model must contain those traces. Traces from $Dont$ *may be accepted* which means that they can belong to a full-control mental model for the system, or not.

The learning-based mental model generation algorithm proposed in [11] uses that categorization to learn a minimal full-control mental model for a given system. Figure 1 shows the partition of the set of traces. That partition represents all the possible mental models that allow full-control of a given system. Such mental models accept all the traces from Acc, reject all the traces from Rej and may accept any subset of $Dont$. The learning-based algorithm incrementally computes this partition. The algorithm then builds, from that partition, a minimal mental model, that is, one with the smallest number of states.

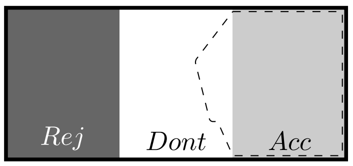

Fig. 1. The set of full-control mental models for a given system, characterized by the categorization of traces into Acc, Rej and $Dont$ sets. The set showed with a dashed line represents one possible mental model. The traces from Acc, $Dont$ and Rej respectively must, may and cannot be part of a full-control mental model.

B. Evaluating the system model during the design process

The full-control mental model generation algorithm can be used in the design process as a validation step of the system. Applying the mental model generation algorithm on a system model can lead to two different outcomes:

1) the system is not well-behaved and does not satisfy the full-control deterministic criterion;
2) or the system does have a minimal full-control mental model.

In the first case, the system is such that it is not possible to generate a full-control mental model for it. That means that there exists some sequence σ so that $s_0 \overset{\sigma}{\Longrightarrow} s$, $s_0 \overset{\sigma}{\Longrightarrow} s'$ and $A^c(s) \neq A^c(s')$. In words, there exists a trace that can lead to different states having different sets of enabled commands. Such a system is said to be *non full-control deterministic*. The operator can thus not know exactly the available commands after that trace, which is in contradiction with the full-control property. In that case, the generation algorithm will provide a trace that it cannot categorize because it must both be accepted and forbidden. That trace can be used to adapt the model of the system for the next iteration of the design process. One way of addressing non-full-control-determinism in a system would be, for example, to add observable behaviour as discussed below.

Figure 2 shows a small example to illustrate this situation; suppose it represents a TV decoder. The decoder is turned on by pressing the on button. Then, if the decoder gets connected to the network, one can select the first channel with the channel1 command. But if the decoder does not have the access to the Internet, one cannot do anything. The generation algorithm will fail and output $\langle \text{on} \rangle$ as a problematic trace. The user can be surprised after executing $\langle \text{on} \rangle$ on such a system. A possible modification of the system would be to add an observation online in the case when the decoder gets connected to the internet. That system is well-behaved and a full-control mental model does exist for it.

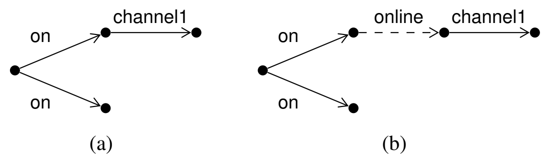

Fig. 2. A not well-behaved system model (on the left) and a possible adaptation of it (on the right).

In the second case, a minimal full-control mental model is produced. It means that the system can be controlled and that automation surprises can be avoided if the operator knows and follows exactly that mental model. However, the larger the mental model, the harder it is for a human operator to memorize and use. The size of the generated mental model can be used in a metric that characterizes good mental models. A too large mental model means that the system is too complex and cannot be simplified to a level that a human operator can follow. The system may need to be redesigned.

C. Checking a system model against user tasks

When interacting with a system, a user does not always need to know all the behaviour of the system. Most of the time, a category of users is only interested in performing some *tasks*, which only partially exercice the capabilities of the system. Given the model of a system and a set of user tasks, the system allows the operator to perform all the user tasks if all the behaviour covered by the tasks can be executed on the system. Such a system can of course have more behaviour, as long as all the tasks are supported.

A user task can be expressed as an LTS $\mathcal{M}_T$. Trace inclusion between the system and the tasks ($\mathbf{Tr}(\mathcal{M}_T) \subseteq \mathbf{Tr}(\mathcal{M}_M)$) can be used to ensure that all traces of the task are supported by the system. However trace inclusion is not a satisfactory criterion for this type of problem as illustrated by the following example.

In Figure 3, where solid lines correspond to commands and dashed lines to observations, the set of traces of the user task is a subset of the set of traces of the system. But there is a situation where the user can be surprised. After performing an a command, the system can transition to a state where the b observation will never occur, resulting in the user not being able to complete the task.

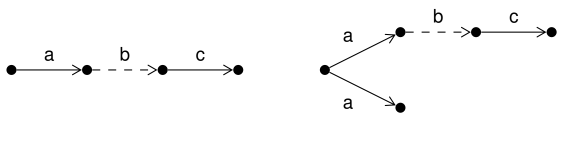

Fig. 3. A system (on the right) which can make the operator confused in some situations, when he wants to perform his tasks (on the left).

The full-control property can be used to achieve a more relevant check between a system model and a user task. Formally, a system model $\mathcal{M}_M = \langle S_M, \mathcal{L}^c, \mathcal{L}^o, s_{0_M}, \delta_M \rangle$ allows the operator to perform the tasks of the user task model $\mathcal{M}_T = \langle S_T, \mathcal{L}^c, \mathcal{L}^o, s_{0_T}, \delta_T \rangle$ if and only if:

$$\forall \sigma \in \mathcal{L}^{co*} \text{ such that } s_{0_T} \xrightarrow{\sigma} s_T \text{ and } s_{0_M} \overset{\sigma}{\Longrightarrow} s_M : \\ A^o(s_T) = A^o(s_M) \text{ and } A^c(s_T) \subseteq A^c(s_M) \qquad (2)$$

For a system to support a user task, the following must hold. At any point during the execution of the task, if the user needs to issue a command, then that command should be included in the commands available in the system at that point. Moreover, at any point during the execution of the task, the task should be prepared to receive exactly those observations available in the system at that point. Therefore the full-control check can be applied between the system model $\mathcal{M}_M$ and the user task model $\mathcal{M}_T$, interchanging commands and observations.

Figure 4 shows an example of a lamp illustrating that relation. The task model indicates that the user should be able to switch between the lamp turned on and turned off with a press command. If the user observes a burnOut, she knows that the lamp is dead and that she cannot do anything more. The proposed system model allows full-control of the task model (inverting the roles of commands and observations). There is indeed an additional behaviour which allows the lamp to be turned off with a fading if the user presses the fadeOut command while in the on state. That additional behaviour is not a problem since, according to the task model, it will never be executed.

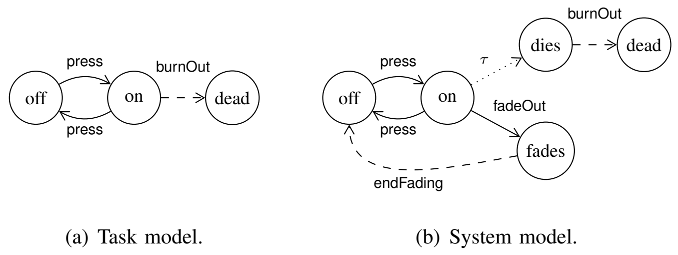

Fig. 4. An example of a set of user task $\mathcal{M}_T$ for a simple lamp (on the left) and a system model $\mathcal{M}_M$ which allows full-control of it (on the right).

IV. Framework Implementation

This section presents how the framework is implemented. The first part of the section describes how the JavaPathfinder model-checker and its statecharts extension are used. The second part introduces the ADEPT toolset and how it can be used with our framework.

A. JPF and statecharts

The proposed framework has been implemented in Java and uses the *Java Pathfinder* (JPF) model-checker [12]. Figure 5 shows a global overview of the developed framework. The first part of the framework aims at providing tools for encoding models using statecharts [27], a widespread graphical notation to model systems. The statecharts can be designed in any existing tool which supports export in XMI file, e.g. ArgoUML [3]. The XMIParser tool [4] converts the statechart into a Java program encoding it, following the conventions of the JPF statecharts extension [28]. With that extension, the resulting Java program can be explored and used by the JPF model checker, for example to check temporal logic properties. The SC2LTS (Statecharts to LTS) tool uses JPF with the JPF statecharts extension to explore all the transitions of the complete behaviour of the system and builds the full expanded LTS. For convenience, LTSs can also be loaded and saved from a plain text format (LTSLoader).

The second part of the framework consists of the analysis and mental model generation part. The FCCheck tool checks whether a mental model allows full-control of a given system. The tool takes two LTSs as input (a system and a mental model) and outputs true if the mental model allows full-control of the system and false otherwise. The MMGen tool takes a model of a system as input and generates a minimal full-control mental model (if such a model exists). Both algorithms from [10], [11] (Bisim and Learning) can be used. The tool produces an LTS corresponding to one minimal full-control mental model or says that no such model exists providing a problematic sequence from the system.

[3]http://argouml.tigris.org/.

[4]The XMIParser does not yet support all the possibilities of the XMI specification.

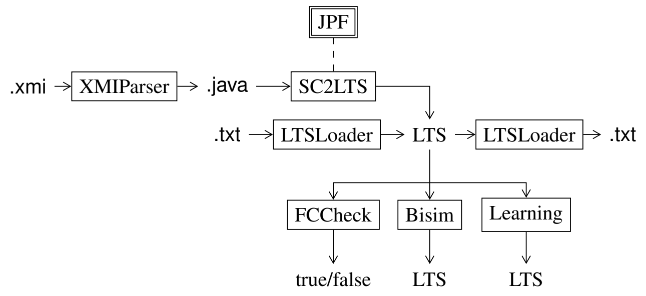

Fig. 5. Global overview of the proposed HMI analysis framework. The elements in rectangle correspond to algorithms developed. The XMI parser builds a Java program encoding the statechart contained in the XMI file. The statechart to LTS converter (SC2LTS) uses JPF to convert a statechart encoded as a Java program into a full expanded LTS. The full-control check (FCCheck) takes two LTS as input (a system and a mental model) and checks if the mental model allows full-control of the system. The mental model generator (MMGen) takes one LTS as input (a system) and computes a minimal full-control mental model.

The benefit of using JPF is that it is a model checker. It can therefore be used to perform additional types of analysis on the statechart model, for example application-specific safety properties as supported by the JPF framework.

B. The ADEPT toolset

The ADEPT toolset [13] aims at helping designers to identify HMI vulnerabilities early in the design process, focusing on cognitive behavioural aspects of the user and the system. The behaviour of the system is described with tables consisting of two parts: inputs and outputs. Each of these parts is decomposed into a set of variables. Each column of those tables represents a transition case which can be triggered for specified values of the variables (input) and has, as effect, to assign new values to those variables (output). Also, one particular class of inputs corresponds to commands, i.e. actions executed by the user. Figure 6 shows an example of an ADEPT model for the system of Figure 4(b). For example, column 0 states that if the lamp is turned on, if its life is still positive and if the press command is executed, then the lamp is turned off and column 3 states that whenever the lamp is turned on and its life is positive, the remaining life is decremented.

	0	1	2	3	4	5
Inputs						
State						
on	●		●	●	●	
fading					●	●
off		●				
Life						
>0	●	●	●	●		
=0					●	
Actions						
press	●	●				
fadeOut			●			
Outputs						
State						
on		●				
fading			●			
off	●				●	●
Life						
-=1				●		

Fig. 6. Example of a model described as an ADEPT table, corresponding to the lamp whose LTS is given in Figure 4(b).

ADEPT models can be translated into statecharts and then be run in our framework. For the experiments presented hereafter, this translation was performed in a manual but systematic way, which we intend to automate in the future. ADEPT models are well suited to be directly translated into JPF statechart Java programs. An automatic translation method is currently being developed. While commands are clearly identified, observations are not. Several models may be derived from one ADEPT model, depending on which variables are considered as observations. For example, for the model of Figure 6, the value of the bulb's remaining life can be made visible to the user. Either the precise value in seconds, or a more abstract information like a life level (for example: full, alive, low and dead).

V. Evaluation

The approach presented in this paper has been applied to many examples [11], two of which are illustrated here. The first one is the Therac-25 medical system and the second one is a video-cassette recorder.

A. The Therac-25 example

The Therac-25 [2] is a medical system which was subject to an accident due to an operator manipulation error which took place during the interaction with the machine. The machine has the ability to treat patients by administering X-ray or electron beams. For the first treatment, a spreader has to be put in place so that the patient does not receive too much radiation. The incident that occurred was that patients were administered X-rays while the spreader was not in place.

The formal model described in [16] has been used to get a JPF statechart Java program. It gives a model with 110 states and 312 transitions, among which there are 194 commands, 66 observations and 52 internal actions. The set of commands is $\{\mathsf{selectX}, \mathsf{selectE}, \mathsf{enter}, \mathsf{fire}, \mathsf{up}\}$ and there is one observation corresponding to a timeout $\{\mathsf{8seconds}\}$. The model is illustrated as a statechart on Figure 7.

The result of the analysis of the Therac-25 system is that it is well-behaved (full-control deterministic as defined in Section III-B) and that it cannot be reduced. The minimal full-control model is thus exactly the same as the system model, without the τ transitions. The potential error with that system cannot be captured with the model as it has been described.

In fact, the error is due to *mode confusion*: the operator believes that the system was in the electron beam mode while it is in fact in the X-ray mode. That kind of error can be found with our framework, by enriching the system model with mode information. Loops are added on all the states where a mode is active with either X-ray or E-beam, according to the mode the machine is in. These new labels are treated as commands, reflecting the fact that the operator must know exactly which mode the machine is in.

Analysing that modified system leads to an error because the system is no longer full-control deterministic. The counterexample produced by the framework is: $\langle \mathsf{selectX}, \mathsf{enter}, \mathsf{fire}, \mathsf{X\text{-}ray} \rangle$. That trace corresponds to a trace that must be accepted and must be forbidden at the same time. Indeed, after selecting

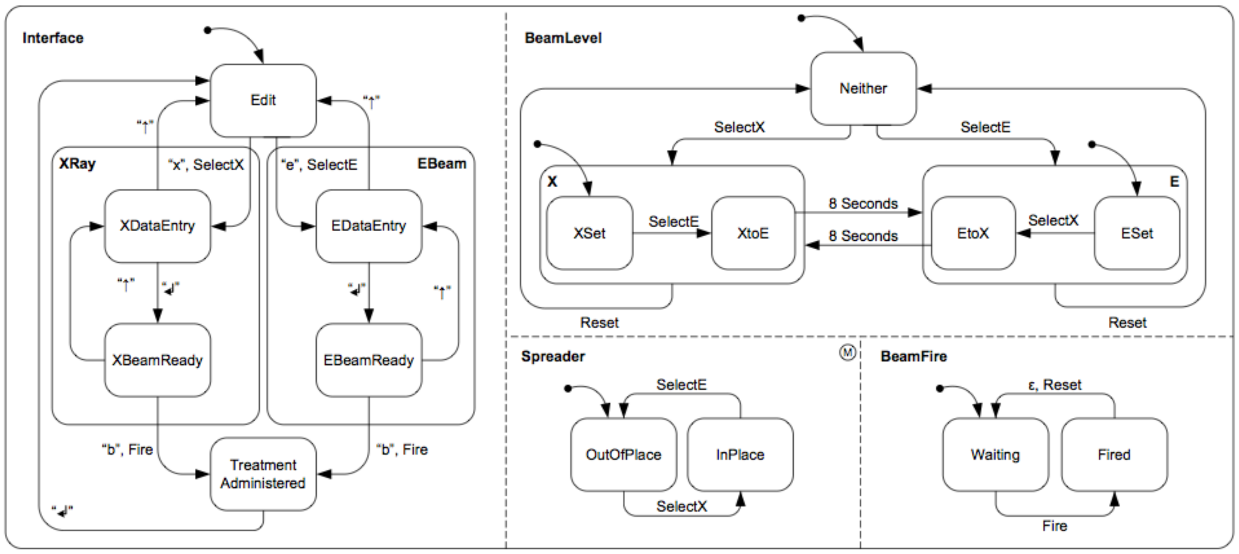

Fig. 7. Statechart model of the Therac-25 medical system [16]. The enter command is represented with ↲ and the up command is represented with ↑.

X-ray beam (selectX), validating it (enter) and administering the treatment (fire), the X-ray command may or may not be available depending on the execution followed in the system. This means that the system may end up either in the X-ray or in the E-beam mode, non-deterministically and with no observable difference. That behaviour is due to an internal transition which occurs when the treatment has been administered, which represents the fact that the system is reset to its initial state. The revealed controllability issue indicates that there should be an observation informing the user when the system is reset. Adding a reset observation makes the system full-controllable and a minimal full-control mental model for it can be generated, with 24 states.

The other issue with that system, the well-known one, can also be detected with our framework, after fixing the other issues in the model. If the operator is not aware of the 8-seconds timer (or does not track the countdown), the issue described in [2], [16] can also be found. It suffices to turn the observation 8seconds into an internal transition τ and to make the system being reset when the operator presses enter after the treatment has been administered. The last actions of the returned counterexample (the most relevant part) is: $\langle \ldots, \mathsf{selectE}, \mathsf{up}, \mathsf{E\text{-}beam}, \mathsf{selectX}, \mathsf{E\text{-}beam} \rangle$. That corresponds to the user selecting the electron beam mode, then changing his mind by pushing on the up button and selecting the X-ray mode. After that, the system may either be in E-beam or X-ray mode.

B. The Video Cassette Recorder example

The second example is a model of a video-cassette recorder (VCR). That model has been obtained from an ADEPT model which has been translated into a JPF statechart Java program, manually. The system has six main operating modes: play, stopped, fast forward, rewind, pause and record. There is one command to activate each of those modes and one additional command to turn the system on or off. Moreover, there are several speeds for the fast forward and rewind modes. Those speeds are automatically chosen by the system according to the remaining tape length. Finally, the system enters automatically into the rewind mode when reaching the end of the tape. The system model has 1088 states and 3740 transitions. There are seven commands (one to activate each mode and a power button) and two observations (tape moving forward and backward).

Analyzing the system against the full-control relation highlights some bad interaction that may happen and lead to confusion by the operator. The issue comes from the [ON, STOP, 0.01] state which corresponds to the VCR being turned on, the tape being stopped and the remaining length of tape being 0.01. In this state, when the user presses the play command, the system can go to one of the two following states: [ON, PLAY, 0.01] or [ON, REWIND_FULL, 0.00]. In the first case, the system just switches in the play mode. In the second case, after the user presses the play button, the tape has just moved forward until it remains zero, in which case the system automatically goes to rewind mode. The controllability issue pointed by the framework is that the pause command is not available in the two states that are reachable from [ON, STOP, 0.01] after executing the action play.

Such kinds of error can be harmful. For the VCR example, suppose that the operator wants to press the play button and then, someone calls him and he presses the pause button as, for him, it is an authorized operation. When he comes back to his VCR, he observes that the tape was rewinded up to the beginning. That situation occurs because after pressing the play button, the system can either transition into a state where the pause button is enabled or in another state where it is not. For critical systems, such an error can have more serious consequences.

VI. Conclusion

This work describes a formal framework for the analysis of human-machine interactions, with a focus on controllability aspects of the system based on a distinction between commands and observations. The analysis is based on a formal characterization of an adequate control of the system by the user. That characterization, captured by the full-control property, is used as a validation criterion for system models during the design process cycle. The full-control property is a desirable property since it helps to prevent the operator from being surprised when interacting with a system. The framework has been implemented in Java within the JPF model checker environment.

The paper demonstrated the use of the proposed methodology and framework for the analysis on two realistic examples. These examples show that our framework can detect problems in system models and provides feedback that helps the designer to identify problematic interaction scenarios and to redesign the system.

Future work includes testing the framework with larger and more realistic examples. It also includes developing an automatic translator of ADEPT models into JPF statecharts. That would make it possible to test our framework with more examples and also to get an opportunity to merge the two tools, that is, to integrate the analysis and generation capabilities of our framework into ADEPT. We also intend to continue developing the framework to increase the support for the process of redesigning the system as a result of identified problems.

There are also perspectives of extension of this work, by considering other kinds of properties to be checked between the system and a mental model, like those checked in [5], [16] for example: *"can the effect of some commands be undone?"* or *"does every action provide a visible feedback?"*. Such analysis could be done by exploiting the JPF model checker and more classical model checking.

Acknowledgement

This work was supported by the Human System Solutions element of NASA's System-wide Safety Assurance Technologies project and by the project MoVES under the Interuniversity Attraction Poles Programme — Belgian State — Belgian Science Policy.

References

[1] A. Degani, *Taming HAL: Designing Interfaces Beyond 2001*. Palgrave Macmillan, Jan. 2004.

[2] N. G. Leveson and C. S. Turner, "Investigation of the therac-25 accidents," *IEEE Computer*, vol. 26, no. 7, pp. 18–41, Jul. 1993.

[3] E. Palmer, "Oops, it didn't arm. — a case study of two automation surprises," in *Proceedings of the 8th International Symposium on Aviation Psychology*, 1996, pp. 227–232.

[4] J. Rushby, "Using model checking to help discover mode confusions and other automation surprises," *Reliability Engineering and System Safety*, vol. 75, no. 2, pp. 167–177, Feb. 2002.

[5] J. C. Campos and M. D. Harrison, "Systematic analysis of control panel interfaces using formal tools," in *Proceedings of the 15th International Workshop on the Design, Verification and Specification of Interactive Systems*, ser. Lecture Notes in Computer Science, no. 5136. Springer-Verlag, Jul. 2008, pp. 72–85.

[6] H. Thimbleby and J. Gow, "Applying graph theory to interaction design," in *Engineering Interactive Systems 2007/DSVIS 2007*, ser. Lecture Notes in Computer Science, J. Gulliksen, Ed., no. 4940. Springer-Verlag, 2008, pp. 501–518.

[7] J. C. Campos and M. D. Harrison, "Model checking interactor specifications," *Automated Software Engineering*, vol. 8, no. 3–4, pp. 275–310, 2001.

[8] P. Curzon, R. Rukšėnas, and A. Blandford, "An approach to formal verification of human-computer interaction," *Formal Aspects of Computing*, vol. 19, no. 4, pp. 513–550, Nov. 2007.

[9] M. Heymann and A. Degani, "Formal analysis and automatic generation of user interfaces: Approach, methodology, and an algorithm," *Human Factors: The Journal of the Human Factors and Ergonomics Society*, vol. 49, no. 2, pp. 311–330, Apr. 2007.

[10] S. Combéfis and C. Pecheur, "A bisimulation-based approach to the analysis of human-computer interaction," in *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS'09)*, G. Calvary, T. N. Graham, and P. Gray, Eds. New York, NY, USA: ACM, 2009, pp. 101–110.

[11] S. Combéfis, D. Giannakopoulou, C. Pecheur, and M. S. Feary, "Learning system abstractions for human-machine interactions," in *(under submission)*.

[12] "JavaPathfinder (JPF)," http://babelfish.arc.nasa.gov/trac/jpf/.

[13] M. S. Feary, "A toolset for supporting iterative human – automation interaction in design," NASA Ames Research Center, Tech. Rep. 20100012861, Mar. 2010.

[14] H. Thimbleby, *Press On: Principles of Interaction Programming*. The MIT Press, Nov. 2007.

[15] D. Navarre, P. Palanque, and R. Bastide, "Engineering interactive systems through formal methods for both tasks and system models," in *Proceedings of RTO Human Factors and Medicine Panel (HFM) Specialists' Meeting*, no. RTO-MP-077, May 2002.

[16] M. L. Bolton, E. J. Bass, and R. I. Siminiceanu, "Using formal methods to predict human error and system failures," in *Proceedings of the 2nd Applied Human Factors and Ergonomics International Conference*, Jul. 2008, pp. 14–17.

[17] M. L. Bolton, R. I. Siminiceanu, and E. J. Bass, "A systematic approach to model checking human-automation interaction using task analytic models," *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 2011.

[18] M. L. Bolton and E. J. Bass, "Using task analytic models and phenotypes of erroneous human behavior to discover system failures using model checking," in *Proceedings of the 54th Annual Meeting of the Human Factors and Ergonomics Society*, vol. 54, Oct. 2010, pp. 992–996.

[19] J. Bredereke and A. Lankenau, "A rigorous view of mode confusion," in *Proceedings of the 21st International Conference on Computer Safety, Reliability and Security (SAFECOMP'02)*, vol. 2434. London, UK: Springer-Verlag, Sep. 2002, pp. 19–31.

[20] ——, "Safety-relevant mode confusions—modelling and reducing them," *Reliability Engineering and System Safety*, vol. 88, no. 3, pp. 229–245, 2005.

[21] J. Tretmans, "Model based testing with labelled transition systems," in *Formal Methods and Testing*, ser. LNCS, R. M. Hierons, J. P. Bowen, and M. Harman, Eds., vol. 4949. Springer, 2008, pp. 1–38.

[22] H. Thimbleby, "Creating user manuals for using in collaborative design," in *Proceedings of the Conference Companion on Human Factors in Computing Systems*. New York, NY, USA: ACM, 1996, pp. 279–280.

[23] J. Nielsen, "The usability engineering life cycle," *Computer*, vol. 25, pp. 12–22, Mar. 1992.

[24] F. Paternò, *Model-Based Design and Evaluation of Interactive Applications*. Springer-Verlag, Jan. 2000.

[25] J. Johnson and A. Henderson, "Conceptual models: Begin by designing what to design," *Interactions*, vol. 9, pp. 25–32, Jan. 2002.

[26] D. Javaux, "A method for predicting errors when interacting with finite state systems. How implicit learning shapes the user's knowledge of a system," *Reliability Engineering and System Safety*, vol. 75, pp. 147–165, Feb. 2002.

[27] D. Harel, "Statecharts: A visual formalism for complex systems," *Science of Computer Programming*, vol. 8, pp. 231–274, Jun. 1987.

[28] P. C. Mehlitz, "Trust your model - verifying aerospace system models with JavaPathfinder," in *Aerospace Conference, 2008 IEEE*, Mar. 2008, pp. 1–11.