Chapter 15
Formal Analysis of Multiple Coordinated HMI Systems

Guillaume Brat, Sébastien Combéfis, Dimitra Giannakopoulou, Charles Pecheur, Franco Raimondi and Neha Rungta

Abstract Several modern safety-critical environments involve multiple humans interacting not only with automation, but also between themselves in complex ways. For example, in handling the National Airspace, we have moved from the traditional single controller sitting in front of a display to multiple controllers interacting with their individual display, possibly each other's displays, and other more advanced automated systems. To evaluate safety in such contexts, it is imperative to include the role of one or multiple human operators in our analysis, as well as focus on properties of human automation interactions. This chapter discusses two novel frameworks developed at NASA for the design and analysis of human–machine interaction problems. The first framework supports modeling and analysis of automated systems from the point of view of their human operators and supports the specification and verification of HMI-specific properties such as mode confusion, controllability, or whether operator tasks are compatible with a particular system. The second framework captures the complexity of modern HMI systems by taking a multi-agent approach to modeling and analyzing multiple human agents interacting with each other as well as with automation.

G. Brat · D. Giannakopoulou · N. Rungta
NASA Ames Research Center, Moffett Field, CA, USA
e-mail: Guillaume.P.Brat@nasa.gov

D. Giannakopoulou
e-mail: Dimitra.Giannakopoulou@nasa.gov

N. Rungta
e-mail: Neha.S.Rungta@nasa.gov

S. Combéfis (✉)
École Centrale des Arts et Métiers (ECAM), Woluwé-Saint-Lambert, Belgium
e-mail: sebastien.combefis@uclouvain.be; s.combefis@ecam.be

C. Pecheur
Université catholique de Louvain, Louvain-la-Neuve, Belgium
e-mail: charles.pecheur@uclouvain.be

F. Raimondi
Middlesex University, London, UK
e-mail: f.raimondi@mdx.ac.uk

© Springer International Publishing AG 2017
B. Weyers et al. (eds.), *The Handbook of Formal Methods in Human-Computer Interaction*, Human-Computer Interaction Series,
DOI 10.1007/978-3-319-51838-1_15

15.1 Introduction

In a number of complex and safety-critical environments such as health care systems, autonomous automobiles, airplanes, and many others, the role of the human operator has shifted from manual control to "monitor and react". Human operators spend a large portion of their time monitoring automated systems and a smaller portion of their time performing actions based on the information provided by the system. There are several examples of such automated systems in use today including autopilots and collision avoidance systems in airplanes, systems that dispense prescribed dosage of medicine to patients, and systems used to perform trading in the financial markets. We expect this trend to continue with the introduction of autonomous cars, unmanned aerial vehicles, remote surgeries, and smart automation in our homes, among others.

To evaluate safety, it is imperative to consider the Human–Machine Interaction (HMI) system as a whole, by including the role of the human operator. Our research at NASA in this domain is targeted primarily on the National Airspace (NAS) in the context of civil aviation, but the techniques presented in this paper can be and have been applied to other domains as well.

The NAS is a complex environment in which humans and automated systems interact to enable safe transportation across the USA. The NAS has been designed to guarantee safety through human decision making with some support from automated systems. In other words, humans are at the core of the current air traffic management system. However, in order to accommodate increasing density of air traffic in the USA, there is a need to increase the use of automated systems that can assist humans such as air traffic controllers. Increased automation is also supported by FAA's Next Generation Air Transportation System (NextGen) program, where the FAA is the regulatory authority for civil aviation in the USA.

The NAS is turning into a complex, interconnected system, in which humans interact with automation extensively: we move from the traditional single controller sitting in front of a display to multiple controllers interacting with their individual display, possibly each other's displays, and other more advanced automated systems. Interactions can be as complex as information flowing from an automated system (possibly merging information coming from several automated systems) to another automated system with human controllers debating options to ensure safe and efficient traffic flow.

Despite the fact that increased automation is essential in assisting humans to problem solve in such a complex environment, infusion of a new technology is a very conservative process driven by the need to maintain current levels of safety. The current infusion process (from creating to certifying and fielding new automated systems) is extremely slow and limits our ability to cope with increasing traffic demands. This puts more pressure on the controllers and might result in pockets of reduced safety, which is not acceptable.

The main reason for the slow pace of infusion of new technology is our reliance on testing and human studies to assess the quality and consequences of fielding new automation. Human studies are very useful and provide many insights as to how controllers can react to new automation. However, they are always limited in scale and therefore fall short of accounting for off-nominal conditions and emergent behavior in this large, connected system that is the NAS. Clearly, we need better methods for analyzing the implications of fielding new automation.

This chapter discusses two novel frameworks developed at NASA for the design and analysis of human–machine interaction problems. The first framework supports modeling and analysis of automated systems from the point of view of their human operators in terms of observations made and commands issued. It supports the specification and verification of HMI-specific properties such as mode confusion, controllability, or whether operator tasks are compatible with a particular system.

The second framework aims at directly capturing the complexity of HMI systems involving multiple users interacting with each other as well as multiple automated systems. It provides a multi-agent approach to modeling and analyzing multiple human agents, e.g., pilots and controllers, interacting with each other as well as with automation. These models are created at a higher-level of abstraction that allows us to express actions, tasks, goals, and beliefs of the human such that experts in the domain of air traffic management and airspace system are able to understand the models. The multi-agent system framework provides support for modeling and scalable analysis to detect safety issues such as loss of separation, aircraft collision, or other user-specified properties in complex HMI systems. We describe the application of these two frameworks to realistic HMI systems.

15.2 From HMI Frameworks to Full-Blown Multi-agent Systems

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. Our work studies 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 (Combéfis et al. 2011). Both the domain and its mental model have labeled transition systems (LTS) semantics. In this context, we developed algorithms for automatically generating minimal mental models as well as checking full-control. We also proposed 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, implemented in the JavaPathfinder model checker (JavaPathfinder 2016), can be used for modeling 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.

Despite the flexibility of our framework in capturing and analyzing properties that are specific to HMI systems, such as mode confusion, controllability by human operators, or whether an operator task is supported by an automated system, it has been clear from early stages of our research that expressing HMI systems directly in terms of LTSs is unmanageable for realistic systems. For this reason, we bridged our analysis framework with environments that domain experts use to model and prototype their systems. In particular, our efforts have focused on the NASA Ames HMI prototyping tool ADEPT (Combéfis et al. 2016). We extended the models that our HMI analysis framework handles to allow adequate representation of ADEPT models. We also provided a property-preserving reduction from these extended models to LTSs, to enable application of our LTS-based formal analysis algorithms. We demonstrated our work on several examples, most notably on an ADEPT autopilot (Combéfis 2013).

However, as discussed, the scale of modern HMI systems goes beyond the modeling and analysis capabilities of such tools. To handle the full scale of the NAS, where multiple humans interact with multiple automated systems, one needs to support appropriate high-level modeling mechanisms as well as a variety of analysis capabilities that range from simulation to exhaustive verification techniques, when possible. To address this challenge, we have moved toward modeling HMI systems as multi-agent systems.

Multi-agent systems (MAS) offer a design abstraction and modeling approach for systems involving humans and automation. They provide the ability for predictive reasoning about various safety conditions such as expected behavior of the autonomy, situational awareness of humans, workload of the human operators, and the amount of time taken from the start to the end of a complete or partial operation or procedure.

Rational agents are commonly encoded using belief–desire–intention (BDI) architectures. BDI architectures, originally developed by Michael Bratman (1987), represent an agent's mental model of information, motivation, and deliberation. Beliefs represent what the agent believes to be true, desires are what the agent aims to achieve, and intentions are how the agent aims to achieve its desires based on its current beliefs. The BDI model is a popular model for representing rational agents, i.e., agents that decide their own actions in a rational and explainable way. This success is probably due to its intuitive representation of human practical reasoning processes.

Our multi-agent framework is an extension of Brahms—a BDI-based MAS framework. Brahms is a simulation and development environment originally designed to model the contextual situated activity behavior of groups of people in a real world context (Clancey et al. 1998; Sierhuis 2001). Brahms has now evolved to model humans, robots, automated systems, agents, and interactions between humans and automated systems. Our extension connects Brahms to a variety of simulation and formal verification tools to enable the safety analysis of NAS models.

15.3 Formal Design and Analysis Techniques for HMI Properties

In this section, we describe our work on developing an HMI-specific formal analysis framework. In our framework, HMI models are represented using an extension of labeled transition systems (LTS) and are analyzed to assert system controllability given the commands that can be issued and the observations that can be made by the human operator. We illustrate the capabilities of the framework with a simple example from the medical community. Even though LTSs are an ideal abstraction for the specification and analysis of HMI properties, it is hard to model large, complex HMI systems directly as LTSs. We therefore also describe our work on connecting our analysis algorithms to the ADEPT prototyping tool, which provides a more intuitive modeling environment for domain experts.

15.3.1 Extended LTS

Since we are interested in controllability properties and detection of automation surprises, the distinction between commands and observations matters (Heymann and Degani 2007; Javaux 2002). Our approach is based on two models: the *system model*, which describes the detailed behavior of the system, and the *mental model*, which represents an abstraction of the system for the human operator.[1]

Our formal models are expressed with enriched *labeled transition systems* (LTS) called HMI LTS. HMI LTSs are essentially graphs whose edges are labeled with actions. The difference with classical LTSs is that three kinds 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 behaviors of the system that is completely hidden to the user.

The detailed formalization is available in Combéfis and Pecheur (2009), Combéfis et al. (2011).

Formally, HMI LTS are tuples $\langle S, \mathscr{L}^c, \mathscr{L}^o, s_0, \delta \rangle$ where S is the set of states; $\mathscr{L}^c$ and $\mathscr{L}^o$ are the sets of commands and observations, respectively; s_0 is the initial state; and $\delta : S \times (\mathscr{L}^c \cup \mathscr{L}^o \cup \{\tau\}) \to 2^S$ 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

[1]The mental model is commonly referred to as *conceptual model* (Johnson and Henderson 2002) in the literature, that is, an abstraction of the system which outlines what the operator can do with the system and what she needs in order to interact with it.

observations and is denoted $\mathscr{L}^{co} = \mathscr{L}^c \cup \mathscr{L}^o$. In this chapter, 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 $\mathscr{L}^{co}$ 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 $\mathscr{M}$ is denoted $\mathbf{Tr}(\mathscr{M})$. 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.

15.3.2 Properties

Given a system model and a mental model, we are interested in the notion of *full controllability* that captures the fact that an operator has enough knowledge about the system in order to control it properly, i.e., at each instant of time, the operator must know exactly the set of commands that the system can receive, as well as the observations that can be made.

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

$$\begin{aligned} \forall \sigma \in \mathscr{L}^{co*} \text{ such that } s_{0_M} \overset{\sigma}{\Longrightarrow} s_M \text{ and } s_{0_U} \xrightarrow{\sigma} s_U \ : \\ A^c(s_M) = A^c(s_U) \text{ and } A^o(s_M) \subseteq A^o(s_U) \end{aligned} \tag{15.1}$$

In other words, it means that for every trace σ that can be executed on both the system and the mental model ($s_{0_M} \overset{\sigma}{\Longrightarrow} s_M$ and $s_{0_U} \xrightarrow{\sigma} s_U$), the set of commands ($A^c$) that are enabled on the system model 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 definition, the user must always know all the possible commands, which is a strong requirement. In the work presented here, we use a weaker variant, where a user may not always know all the possible commands but only those which are relevant for interacting with the system.

The full-control property characterizes a conceptual model for a given system, which we refer to as full-control mental model. All the behaviors of the system must be covered by the full-control mental model, and it should allow the user to interact

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

correctly with the system. This means that the user always knows what can be done on the system and what can be observed from it.

The full-control property captures the behavior of a given system that a user should know in order to operate it without errors. Given a system model $\mathscr{M}_M = \langle S_M, \mathscr{L}^c, \mathscr{C}^o, s_{0_M}, \delta_M \rangle$, a trace $\sigma \in \mathscr{L}^{co*}$ can be put into one of three different categories. In other words, the set of traces of $\mathscr{M}_M$ 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 behavior of the system of which the user must be aware.
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.

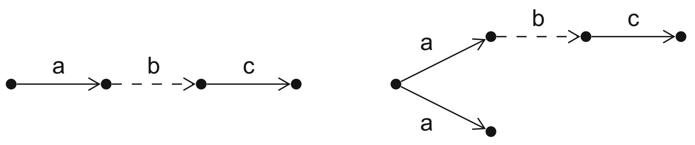
Traces from *Rej are forbidden* which means that they cannot be part of a full-control mental model. Traces from *Acc must be accepted* which means that any full-control mental model must contain those traces. Traces from *Dont may be accepted* which means that they can belong to a full-control mental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

15.3.3 Analysis

When interacting with a system, a user does not always need to know all the behavior of the system. Most of the time, users are interested in performing some *tasks* that only partially exercise 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 behavior covered by the tasks can be executed on the system. Such a system can of course have more behavior, as long as all the tasks are supported.

A user task can be expressed as an LTS $\mathscr{M}_T$. Trace inclusion between the system and the tasks ($\mathbf{Tr}(\mathscr{M}_T) \subseteq \mathbf{Tr}(\mathscr{M}_M)$) 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 Fig. 15.1, 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.



(a) User task model. (b) System model.

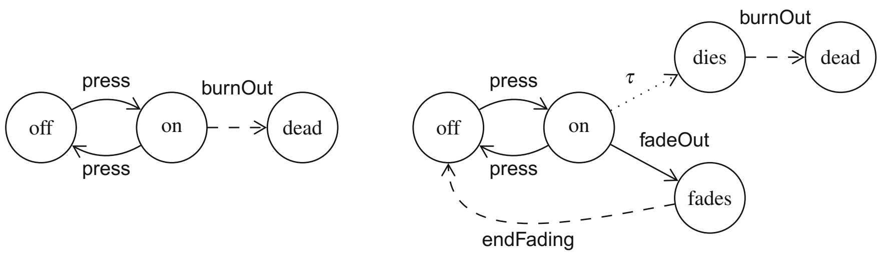
Fig. 15.1 A system (on the *right*) which can make the operator confused in some situations, when he wants to perform his tasks (on the *left*)

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 $\mathscr{M}_M = \langle S_M, \mathscr{L}^c, \mathscr{L}^o, s_{0_M}, \delta_M \rangle$ allows the operator to perform the tasks of the user task model $\mathscr{M}_T = \langle S_T, \mathscr{L}^c, \mathscr{L}^o, s_{0_T}, \delta_T \rangle$ if and only if:

$$\forall \sigma \in \mathscr{L}^{co*} \text{ such that } s_{0_T} \xrightarrow{\sigma} s_T \text{ and } s_{0_M} \overset{\sigma}{\Longrightarrow} s_M \; : \\ A^o(s_T) = A^o(s_M) \text{ and } A^c(s_T) \subseteq A^c(s_M) \qquad (15.2)$$

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 $\mathscr{M}_M$ and the user task model $\mathscr{M}_T$, interchanging commands and observations.

Figure 15.2 shows an example of a lamp illustrating that relation. The task model indicates that the user should be able to switch a lamp on and off with a press command. If the user observes a burnOut signal, then the lamp is dead and nothing else can be done. The proposed system model allows full-control of the task model (inverting the roles of commands and observations). There is an additional behavior which



(a) Task model. (b) System model.

Fig. 15.2 An example of a set of user task $\mathscr{M}_T$ for a simple lamp (on the *left*) and a system model $\mathscr{M}_M$ which allows full-control of it (on the *right*)

allows the lamp to be dimmed down all the way to off if the user presses the fadeOut command while in the on state. That additional behavior is not a problem since, according to the task model, it will never be executed.

15.3.4 Example

The Therac-25 (Leveson and Turner 1993) 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 accident occurred when patients were administered X-rays while the spreader was not in place.

The formal model described in Bolton et al. (2008) was represented in the JPF framework and had 110 states and 312 transitions, among which there are 194 commands, 66 observations, and 52 internal actions. The set of commands is {selectX, selectE, enter, fire, up}, and there is one observation corresponding to a timeout {8 second}. The model is illustrated as a statechart on Fig. 15.3.

The result of the analysis of the Therac-25 system is that it is well-behaved (i.e., it is full-control deterministic) 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 actually 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-

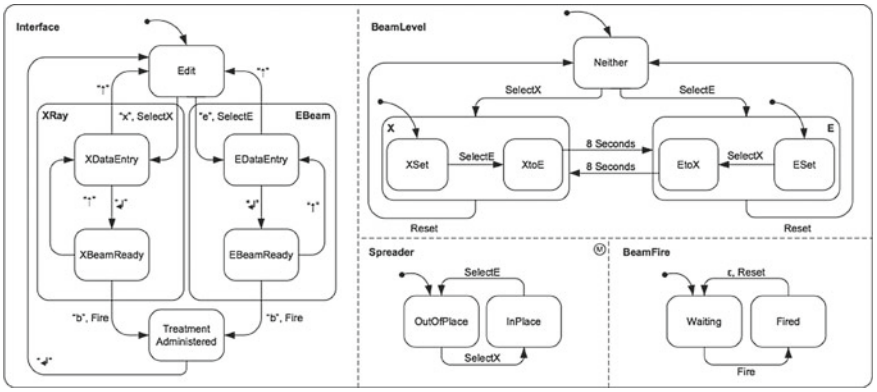


Fig. 15.3 Statechart model of the Therac-25 medical system (Leveson and Turner 1993). The enter command is represented with ↲ and the up command is represented with ↑

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.

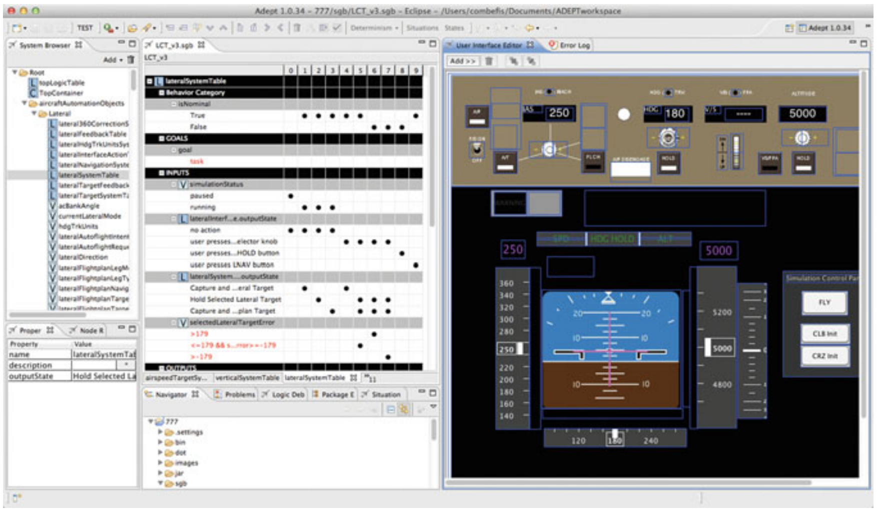
Analyzing that modified system leads to an error, because the system is no longer full-control deterministic. The counterexample produced by the framework is as follows: ⟨selectX, enter, fire, X-ray⟩. That trace corresponds to a trace that must be accepted and must be forbidden at the same time. Indeed, after selecting 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 behavior is due to an internal transition that occurs when the treatment has been administered, which represents the fact that the system is reset to its initial state. The 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.

There is another issue with that system. If the operator is not aware of the 8-second timer (or does not track the countdown), the issue described in Leveson and Turner (1993), Bolton et al. (2008) can also be found. It suffices to turn the observation 8 second 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 as follows: ⟨..., selectE, up, E-beam, selectX, E-beam⟩. 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.

15.3.5 Analysis of ADEPT Models

Even though HMI LTSs are an ideal abstraction for the specification and analysis of the HMI properties described above, it is hard to model large, complex HMI systems directly as LTSs. For this reason, we have worked on connecting our analysis algorithms to prototyping tools that are more familiar and intuitive for domain experts. In particular, we have worked closely with the developers of the ADEPT tool.

ADEPT (*Automatic Design and Evaluation Prototyping Toolset*) (Feary 2010) is a Java-based tool developed at NASA Ames, which supports designers in the early prototyping phases of the design of automation interfaces. The tool also offers a set of basic analyses that can be performed on the model under development. An ADEPT model is composed of two elements: a set of logic tables, coupled with an interactive user interface (UI). The logic tables describe the dynamics of the system as state changes in reaction to user actions or to environmental events. For example, Fig. 15.4 shows a screenshot of the autopilot model opened in ADEPT. The left part of the window shows one of the logic tables and the right part shows the user interface.



	0	1	2	3	4	5	6	7	8	9
lateralSystemTable										
Behavior Category										
isNominal										
True		●	●	●	●	●				●
False							●	●	●	
GOALS										
goal										
task										
INPUTS										
simulationStatus										
paused	●									
running		●	●	●						
lateralInterf...e.outputState										
no action	●	●	●	●						
user presses...elector knob					●	●	●	●		
user presses...HOLD button									●	
user presses LNAV button										●
lateralSystem....outputState										
Capture and ...eral Target		●			●					
Hold Selected Lateral Target			●			●	●	●		
Capture and ...plan Target				●		●	●	●		
selectedLateralTargetError										
>179							●			
<=179 && s...rror>=-179						●				
>-179								●		
OUTPUTS										

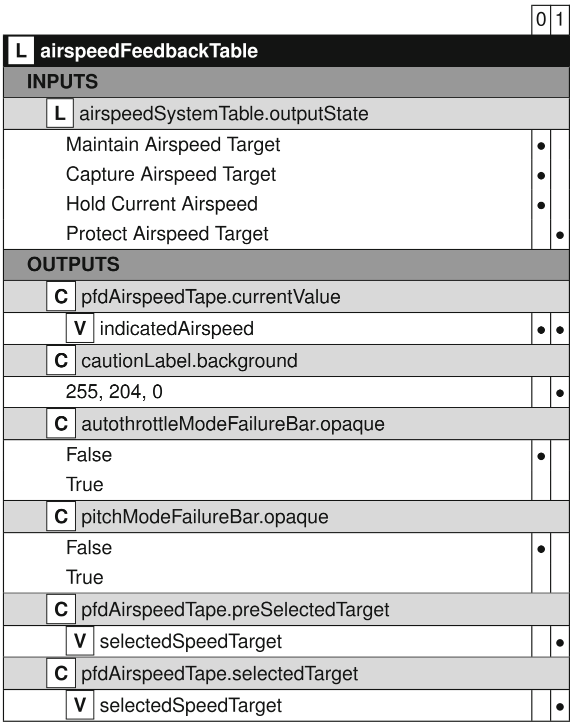
Fig. 15.4 The autopilot model opened in ADEPT, with one logic table in the *left* part of the window and the user interface on the *right* part

The UI is composed of a set of components that are encoded as Java objects representing graphical widgets. The logic tables can refer to the elements of the UI and to the other components through their Java instance variables, and interact with them through their methods, using Java syntax. In particular, UI events are seen as Boolean variables that are set to true when the event occurs.

Behind the scene, an ADEPT model is compiled into a Java program that can be executed in order to directly try the encoded behavior with the user interface. That tool is meant to be used as a rapid prototyping tool. The models not only can then be tested and simulated by the designers, but can also be analyzed by systematic and rigorous techniques. Possible analyses include validity checks on the structure of logic tables, for example. We have extended the analysis capabilities of ADEPT with our framework, which requires the translation of ADEPT tables into the models that our framework can handle. We describe our work with ADEPT through an autopilot example.

Figure 15.5 shows one of the logic tables of the autopilot model. The table example illustrates the way it can interact with elements of the UI. Each light gray line of the table corresponds to a variable of the system. The variables can be related to a component of the UI (such as pfdAirspeedTargetTape.currentValue), or they can be state variables of the model (such as indicatedAirSpeed) or they can relate to the internal logic of the system (airspeedSystemTable.outputState). The latter kind of variables can be seen as a description of the mode of a particular component of the system (the airspeed part in this example). For example, the two first lines of the output part of the logic table example mean that the value of the currentValue field of the pfdAirspeedTargetTape component of the UI is updated with the value of the indicatedAirspeed

Fig. 15.5 An example of a logic table: the airspeed feedback table of the autopilot model contains the logic related to the update of the UI for the airspeed part



	0	1
L airspeedFeedbackTable		
INPUTS		
L airspeedSystemTable.outputState		
Maintain Airspeed Target	●	
Capture Airspeed Target	●	
Hold Current Airspeed	●	
Protect Airspeed Target		●
OUTPUTS		
C pfdAirspeedTape.currentValue		
V indicatedAirspeed	●	●
C cautionLabel.background		
255, 204, 0		●
C autothrottleModeFailureBar.opaque		
False	●	
True		
C pitchModeFailureBar.opaque		
False	●	
True		
C pfdAirspeedTape.preSelectedTarget		
V selectedSpeedTarget		●
C pfdAirspeedTape.selectedTarget		
V selectedSpeedTarget		●

state variable. Moreover, each column of an ADEPT table corresponds to a transition scenario. From any state of the system that satisfies the condition described by the input part of the table, the system can move to the state of the system that results in applying the update instructions described by the output part of the table.

The ADEPT autopilot partially models the behavior of the autopilot of a Boeing 777 aircraft. The full autopilot ADEPT model has a total of 38 logic tables. Three major groups of tables can be identified in the model, namely one for the lateral aspect, one for the vertical aspect, and finally, one for the airspeed aspect. For each of these aspects, the logic tables are further partitioned into three groups: the action tables, the system tables, and the feedback tables, successively executed in that order. Action tables determine actions from UI events, system tables update the state of the system according to the performed action, and feedback tables reflect the state of the system to UI elements.

For a full account of our autopilot case study, we refer the interested reader to (Combéfis 2013). Here, we report briefly on the extensions that we applied to our formalisms as well as our experiences and some observations with analyzing autopilot system tables. First of all, given the fact that ADEPT models are state-based and in order to be able to easily support the automatic translation of ADEPT models into HMI LTS for the application of our analysis algorithms, we extended our HMI models with state information, as described in Combéfis et al. (2016). These models are named HMI state-Valued System models, or HVS. Our techniques cannot scale to the full size of such a large and complex model. Therefore, our analyses were performed on parts of the autopilot model, each analysis considering subsets of the system tables. As is typical with formal techniques, we additionally had to abstract the infinite data domains involved in the models.

Using the outputState as a mode indicator, we performed mode confusion analysis and detected a potential mode confusion on the airspeedSystemTable. This was identified during the minimal model generation phase, where the generation algorithm produced an error trace witnessing the fact that the system model was not fc-deterministic. By analyzing the error trace manually, we localized the erroneous behavior in the involved ADEPT tables. One of the models that was analyzed was an HVS with 7680 states and 66242 transitions, among which 57545 are labeled with commands and 8697 are internal τ-transitions. The obtained minimal mental model has 25 states and 180 transitions.

Our experiences described in this section lead us to the conclusion that in developing modeling and analysis techniques that can capture multiple aspects of the NAS, we would need to move toward formalisms that enable the intuitive and scalable modeling of multiple interacting agents (human and automation), with support for a variety of analyses, starting from more scalable simulations and toward not only more sophisticated, but also more resource-consuming exhaustive techniques such as the ones we presented. These observations lead to the work that is described in the next section.

15.4 Coordinating HMIs as Multi-agent Systems

This section describes the modeling analysis of multiple interacting HMIs specified in the Brahms modeling language (Clancey et al. 1998; Sierhuis 2001). The input to our framework is a Brahms model along with a Java implementation of its semantics. Brahms is a multi-agent simulation system in which people, tools, facilities, vehicles, and geography are modeled explicitly. The air transportation system of the NAS is modeled as a collection of distributed, interactive subsystems such as airports, air traffic control towers and personnel, aircraft, automated flight systems and air traffic tools, instruments, and flight crew. Each subsystem, whether a person or a tool such as the radar, is modeled independently with properties and contextual behaviors. Brahms facilitates modeling various configurable realistic scenarios that allows the analysis of the airspace in various conditions and reassignment of roles and responsibilities among human and automation.

15.4.1 The Brahms Language

Brahms is a full-fledged multi-agent, rule-based, activity programming language. It is based on a theory of work practice and situated cognition (Clancey et al. 1998; Sierhuis 2001). The Brahms language allows for the representation of situated activities of agents in a geographical model of the world. Situated activities are actions performed by the agent in some physical and social context for a specified period of time. The execution of actions is constrained (a) locally: by the reasoning capabilities of an agent and (b) globally by the agents beliefs of the external world, such as where the agent is located, the state of the world at that location and elsewhere, located artifacts, activities of other agents, and communication with other agents or artifacts. The objective of Brahms is to represent the interaction between people, off-task behaviors, multitasking, interrupted and resumed activities, informal interactions, and knowledge, while being located in some environment representative of the real world.

At each clock tick, the Brahms simulation engine inspects the model to update the state of the world, which includes all of the agents and all of the objects in the simulated world. Agents and objects have states (factual properties) and may have capabilities to model the world (e.g., radar display is modeled as beliefs, which are representations of the state of the aircraft). Agents and objects communicate with each other; the communication can represent verbal speech, reading, writing, etc. and may involve devices such as telephones, radios, and displays. Agents and objects may act to change their own state, beliefs, or other facts about the world.

15.4.2 MAS Formal Analysis

We use model checking-based techniques to systematically explore the various behaviors in Brahms scenarios, i.e., in our case study collision scenarios of the Überlingen model configuration. In Hunter et al. (2013) and Rungta et al. (2013), we present an extensible verification framework that takes as input a multi-agent system model and its semantics as input to some state space search engine (or a model checker). The search engine generates all possible behaviors of the model with respect to its semantics. The generated behaviors of the model are then encoded as a reachability graph $G = (N, E)$, where N is a set of nodes and E is a set of edges. This graph is automatically generated by the search engine. Each node $n \in N$ is labeled with the belief/facts values of the agents and objects. In Hunter et al. (2013), we generate the reachability graph using the Java PathFinder byte-code analysis framework. An edge between the nodes represents the updates to beliefs/facts and is also labeled with probabilities. The reachable states generated by JPF are mapped to the nodes in a reachability graph. This reachability graph is essentially an LTS.

The verification of safety properties and other reachability properties is performed on the fly as new states and transitions are generated. JPF is an explicit-state analysis engine that stores the generated model in memory. Capturing the state of all the agents and objects in Brahms including their workframes and thoughtframes can lead to large memory requirements. Additionally, for large systems, it is often intractable to generate and capture even just the intermediate representation in memory.

To overcome these limitations, we adopt a stateless model checking approach. Stateless model checking explores all possible behaviors of the program or model without storing the explored states in a visited set. The program or model is executed by a scheduler that tracks all the points of non-determinism in the program. The scheduler systematically explores all possible execution paths of the program obtained by the non-deterministic choices. Stateless model checking is particularly suited for exploring the state space of large models. In this work, we instrument the Brahms simulator to perform stateless model checking. The instrumented code within the Brahms engine generates all possible paths (each with different combinations of activity durations) in depth-first ordering. Stateless model checkers like VeriSoft (Godefroid 1997) do not in general store paths; however, in order to perform further analysis of the behaviors space, the Brahms stateless model checker can store all the generated paths in a database.

15.4.2.1 Non-determinism in Brahms

There are two main points of non-determinism in Brahms models. The first point of non-determinism is due to durations of primitive activities. The different primitive activities in Brahms have a duration in seconds associated with them. The duration of the primitive activity can either be fixed or can vary based on certain attributes of the primitive activities. When the random attribute of a primitive activity is set to true, the simulator randomly selects the primitive activity duration between the min and max durations specified for the activity. The second point of non-determinism arises from probabilistic updates to facts and beliefs of agents and objects. Updates to facts and beliefs are made using conclude statements in Brahms. Here is an example of a conclude statement:

$$\mathit{conclude}((\mathit{Pilot}.\mathit{checkStall} = \mathit{false}), \mathit{bc} : 70, \mathit{fc} : 70)$$

This states that the belief and fact, *checkStall*, in the *Pilot* agent will be updated to false with a probability of 70%. Here, *bc* represents belief certainty while *fc* represents fact certainty.

In the Überlingen model, currently there are only deterministic updates to facts or beliefs. The updates to facts and beliefs are asserted with a 100% probability. Nevertheless, there is a large degree of non-determinism due to variations in activity durations. The difference in minimum and maximum duration ranges from 2 s to a few 100 s. This can potentially lead to a large number of timing differences between the various events.

15.4.2.2 Behavior Space

The scheduler within the stateless Brahms model checker generates all possible paths through the different points of non-determinism in the Brahms model. Note that in describing the output of the Brahms stateless model checker, we use the terms path and trace interchangeably. Intuitively, a path (or trace) generated by the Brahms stateless model checker is equivalent to a single simulation run. More formally, a path or trace is a sequence of events executed by the simulator $< e0, e1, e2, ..., ei >$. Each event in the trace is a tuple, $< a, t, (u, val) >$ where a is the actor id, t is the Brahms clock time, and u is the fact or belief updated to the value val. For each trace, we generate a sequence of nodes in the intermediate representation $n_{init}, n_0, n_1, n_2, ..., n_i$. The initial node in the sequence, n_{init} is labeled with the initial values of belief/facts values for the various agents and objects. The event $e_0 := < a_0, t_0, (u_0, val_0) >$ is applied to the initial node n_{init} where the value assigned to u_0 is updated to val_0. Each event is applied in sequence to a node in the intermediate representation to generate $n_{init}, n_0, n_1, n_2, ..., n_i$.

15.4.3 Case Study: The Überlingen Collision

The Überlingen accident, (Überlingen 2004), involving the (automated) Traffic Collision Avoidance System (TCAS), is a good example to illustrate problems arising from multiple human operators interacting with multiple automated systems. TCAS has the ability to reconfigure the pilot and air traffic control center (ATCC) relationship, taking authority from the air traffic control officer (ATCO) and instructing the pilot.

TCAS is an onboard aircraft system that uses radar transponder signals to operate independently of ground-based equipment to provide advice to the pilot about conflicting aircraft that are equipped with the same transponder/TCAS equipment. The history of TCAS dates at least to the late 1950s. Motivated by a number of midair collisions over three decades, the United States Federal Aviation Administration (FAA) initiated the TCAS program in 1981. The system in use over Überlingen in 2002 was TCAS II v.7, which had been installed by US carriers since 1994: TCAS II issues the following types of aural annunciations:

- Traffic advisory (TA)
- Resolution advisory (RA)
- Clear of conflict

When a TA is issued, pilots are instructed to initiate a visual search, if possible, for the traffic causing the TA. In the cases when the traffic can be visually acquired, pilots are instructed to maintain visual separation from the traffic. When an RA is issued, pilots are expected to respond immediately to the RA unless doing so would jeopardize the safe operation of the flight. The separation timing, called TAU, provides the TA alert at about 48 s and the RA at 35 s prior to a predicted collision.

On July 1 2002, a midair collision between a Tupolev Tu-154M passenger jet traveling from Moscow to Barcelona, and a Boeing 757-23APF DHL cargo jet manned by two pilots, traveling from Bergamo to Brussels, occurred at 23:35 UTC over the town of Überlingen in southern Germany. The two flights were on a collision course. TCAS issued first a Traffic Advisory (TA) and then a Resolution Advisory (RA) for each plane. Just before TCAS issued an RA requesting that the Tupolev climb, the air traffic controller in charge of the sector issued a command to descend; the crew obeyed this command. Since TCAS had issued a Resolution Advisory to the Boeing crew to descend that they immediately followed, both planes were descending when they collided.

The decision of the Tupolev crew to follow the ATC's instructions rather than TCAS was the immediate cause of the accident. The regulations for the use of TCAS state that in the case of conflicting instructions from TCAS and ATCO, the pilot should follow the TCAS instructions. In this case study, the conflict arose because the loss of separation between the two planes was not detected or corrected by the ATCO. The loss of separation between airplanes are frequent occurrences; it is part of the normal work of air traffic control to detect and correct them accordingly.

There were a set of complex systemic problems at the Zurich air traffic control station that caused the ATCO to miss detecting the loss of separation between the two planes. Although two controllers were supposed to be on duty, one of the two was resting in the lounge: a common and accepted practice during the lower workload portion of night shift. On this particular evening, a scheduled maintenance procedure was being carried out on the main radar system, which meant that the controller had to use a less capable air traffic tracking system. The maintenance work also disconnected the phone system, which made it impossible for other air traffic control centers in the area to alert the Zurich controller to the problem. Finally, the controllers workload was increased by a late arriving plane. An A320 that was landing in Friedrichshafen required the ATCO's attention, who then failed to notice the potential separation infringement of the two planes.

The Überlingen collision proves that methods used for certifying TCAS II v7.0 did not adequately consider human–automation interactions. In particular, the certification method treated TCAS as if it were flight system automation, that is, a system that automatically controls the flight of the aircraft. Instead, TCAS is a system that tells pilot how to maneuver the aircraft, an instruction that implicitly removes and/or overrides the ATCs authority. Worldwide deployment of TCAS II v7.1 was still in process in 2012, a decade after the Überlingen collision.

15.4.3.1 Brahms' Model

In a Brahms model, the entire work system is modeled, including agents, groups to which they belong, facilities (buildings, rooms, offices, spaces in vehicles), tools (e.g., radio, radar display/workstation, telephone, vehicles), representational objects (e.g., a phone book, a control strip), and automated subsystems (e.g., TCAS), all located in an abstracted geography represented as areas and paths. Thus, the notion of human–system interaction in Brahms terms is more precisely an interaction between an agent and a subsystem in the model; both are behaving within the work system. A workframe in Brahms can model the interaction between an agent's beliefs, perception, and action in a dynamic environment, for example, these characteristics are leveraged when modeling how a pilot deploys the aircraft landing gear. A pilot uses the on-board landing control and then confirms that the landing gears are deployed while monitoring the aircraft trajectory on the Primary Flight Display. This is modeled in Brahms as follows: a pilot (e.g., the DHL pilot) is a member of the PilotGroup, which has a composite activity for managing aircraft energy configuration. A specific instance of a conceptual class is called a conceptual object. A particular flight (e.g., DHX611, a conceptual object) is operated by a particular airline and consists of a particular crew (a group) of pilots (agents) who file a particular flight plan document (an object), and so on. Each instance of an agent and object have possible actions defined by workframes where each workframe contains a set of activities that are ordered and often prioritized. Certain workframes are inherited from their group (for agents) or class (for objects). The set of possible actions are modeled at a general level and all members of a group/class have similar capabilities (represented as activities, workframes, and thoughtframes); however, at any time during the simulation, agent and object behaviors, beliefs, and facts about them will vary depending on their initial beliefs/facts and the environment with which they are interacting. The model incorporates organizational and regulatory aspects implicitly, manifest by how work practices relate roles, tools, and facilities.

A Brahms simulation model configuration consists of the modeled geography, agents, and objects, as well as their initial facts and beliefs of agents and objects. The different configurations allow us to perform a what-if analysis on the model. The time of departure for a flight might be an initial fact in a Brahms model. One can modify the model to assign a different time of departure for a flight in each simulation run. Another example of configurable initial facts may include work schedules for air traffic controllers. In one configuration of the work schedules, an air traffic controller may be working alone in the ATCC, while in another configuration, two controllers would be present in the ATCC. Initial beliefs of an agent might be broad preferences affecting behavior (e.g., TCAS should overrule the ATC), thus initial beliefs can be used as switches to easily specify alternative configurations of interest. Alternative configurations are conventionally called scenarios. Thus for example, a scenario might be a variation of the Überlingen collision in which two aircraft have flight times that put them on an intersecting path over Überlingen; the only other flight is a late arriving flight for Friedrichshafen and maintenance degrades the radar, but the telephones are in working order.

In general, a model is designed with sufficient flexibility to allow investigating scenarios of interest. The set of causal factors of interest (e.g., use of control strips when approving aircraft altitude changes, availability of telephones) constitute states of the world and behaviors that can be configured through initial facts and beliefs. The initial settings define a space of scenarios. Using Brahms to evaluate designs within this space, while using formal methods to help modelers understand its boundaries so they can refine the model to explore alternative scenarios constitutes the main research objective of this work.

The simulation engine determines the state of a modeled object (e.g., aircraft). It determines the state of its facts and beliefs. Some objects are not physical things in the world, but rather conceptual entities, called conceptual classes in the Brahms language. These represent processes, a set of people, physical objects, and locations (e.g., flights), and institutional systems (e.g., airlines) that people know about and refer to when organizing their work activities.

15.4.3.2 High-Level Structure of Model

The systems that are mentioned in the accident report and play a role in accident have been modeled in the Brahms Überlingen model. These include the pilots in each aircraft, two ATCOs at Zurich, the relevant airspace and airports, all the aircraft relevant to the accident, on-board automation, and other flight systems. The following key subsystems and conditions are modeled in the Brahms Überlingen model:

1. Interactions among Pilot, Flight Systems, and Aircraft for climb and cruise with European geography for one plane, the DHL flight plan.
2. BTC flight, flight plan (two versions: on-time and delayed with collision) and geography: this is independent of ATCO actions to confirm that simulation reproduces collision with flight paths actually flown.
3. Radar Systems and Displays with ATCOs, located in Control Centers, monitoring when flights are entering and exiting each European flight sector in flight plans.
4. Handover interactions between Pilot and ATCOs for each flight phase.
5. Two ATCOs in Zurich, Radar Planner (RP), and ARFA Radar Executive (RE), assigned to two workstations (RE has nothing to do under these conditions).
6. Add TCAS with capability to detect separation violations, generate Traffic Advisory (TA) and Resolution Advisory (RA). DHL and BTC are delayed (on collision course, which tests TCAS)
7. Pilots follow TCAS instructions
8. ATCO may intervene prior to alert depending on when ATCO notices conflict in Radar Displays since ATCO is busy communicating with other flights, moving between workstations, and trying to contact Friedrichshafen control tower on the phone.
9. AEF flight and flight plan so Zurich ARFA RE performs landing handoff to Friedrichshafen controller.
10. Third plane, the AEF flight, arrives late, requiring ATCO communications and handoff to Friedrichshafen: (a) Handled by ATCO in Zurich at right workstation (ARFA sector) and not left East and South sector workstation. (b) Phone communications for handovers, (c) Methods used by ATCO when phone contact does not work:

   a. Ask Controller Assistant (CA) to get another number (pass-nr); requires about 3 min for CA to return
   b. After pass-nr fails, discuss with CA other options about 30 sec
   c. When not busy handling other flights, try pass-nr again.
   d. When plane is at Top-Of-Descent waypoint, as specified in STAR, for landing at airport, within N nm of airport, method of last resort is to call pilots on radio and ask them to contact the tower directly

11. STCA added to ATCO workstations (modeling normal and fallback mode without optical alert). The ATCO responds to alert by advising Pilot to change flight level based on next flight segment of flight plan.
12. Reduce to one Zurich ATCO which triggers the sequence of variations from the nominal situation; now Zurich ATCO must operate flights from two workstations.

15.4.3.3 Analysis Results

The question that the analysis tries to answer, using both simulation and verification, is why under certain conditions, a collision is averted, while in others it is not? In the analysis, we try to gauge how the temporal sensitivity and variability of the interactions among ATCO, TCAS, and the pilots impacts the potential loss of separation and collision of the planes. Concretely, the questions that we ask during the analysis are as follows:

- Given that the arrival of the AEF flight is disrupting the ATCOs monitoring of the larger airspace (e.g., if it arrives sufficiently late, no collision occurs), what is the period (relative to the BTC and DHL flights paths) when AEF's arrival can cause collision?
- During this period, does a collision always occur or are there variations of how the AEF handoff occurs, such that sometimes the separation infringement is averted?
- Is there evidence that high-priority activities such as monitoring the sector are repeatedly interrupted or deferred, implying the ATCO is unable to cope with the workload?

The Brahms Überlingen Model defines a space of work systems (e.g., is STCA optical functioning? are there two ATCOs?) and events (e.g., the aircraft and flights). Every model configuration, which involves configuring initial facts, beliefs, and agent/object relations, constitutes a scenario that can be simulated and will itself produce many different outcomes (chronology of events), because of non-deterministic timings of agent and object behaviors. The model was developed and tested with a variety of scenarios (e.g., varying additional flights in the sector; all subsystems are working properly). The Überlingen accident is of special interest, in which systems are configured as they were at the time of the accident and the DHL and BTC planes are on intersecting routes.

The key events that occur during simulation are logged chronologically in a file that constitutes a readable trace of the interactions among the ATCO, pilots, and automated systems. The log includes information about the following:

(a) ATCO–pilot interaction regarding a route change, including flight level and climb/descend instruction,
(b) Separation violation events detected by TCAS, including TAU value,
(c) Closest aircraft and separation detected by ATCO when monitoring radar,
(d) STCA optical or aural alerts, including separation detected,
(e) Agent movements (e.g., ATCO shifting between workstations),
(f) Aircraft movements, including departure, entering and exiting sectors, waypoint arrival, landing, collision, airspeeds, and vertical,
(g) Aircraft control changes (e.g., autopilot disengaged),
(h) Radio calls, including communicated beliefs, and
(i) Phone calls that fail to complete.

The outcome of ten simulation runs of Brahms Überlingen model configured for the collision scenario are shown in Table 15.1. In simulation runs 1, 2, and 3, the ATCO intervenes before TCAS TA, but planes have not separated sufficiently, TCAS will take BTCs descent into account, advising DHL to climb. In the simulation runs 4, 5, 7, 8, and 9, the ATCO intervenes between TA and RA. In these runs, whether the planes collide depends on timing. As shown in Table 15.1 two of the five runs results in a collision. Note that in our model, a collision is defined as occurring when the vertical separation between the planes is less than a 100 feet. Finally, in the simulation runs 6 and 10, the ATCO intervenes about 10 s after the TCAS RA, which the BTC pilots ignore (or might be imagined as discussing for a long time); therefore, BTC continues flying level while DHL descends and they miss each other, separated by more than 600 ft at the crossing point. In other runs, we have also observed that ATCO intervenes so late, he actually takes the pilots' report about TCAS RA instructions into account.

When ATCO intervenes in the period between the TA and RA in runs 4, 5, 7, 8, and 9, a collision is possible, like what happened at Überlingen: ATCO has to intervene before the TA advising BTC to descend so that BTC can respond before TCAS advises DHL to climb. In runs 4 and 7, collision is narrowly averted because BTC begins to descend 4 or 5 s after the TCAS RA, which is sufficient for a narrow miss (just over 100 feet). In run 9, the BTC descent begins 5 s before the RA, hence the aircraft miss by more than 200 feet). Runs 5 and 8 lead to a collision because the TCAS RA and BTC AP disengage occur at the same time, like what happened at Überlingen.

Because the model uses the Überlingen descent tables to control the BTC and DHL aircraft during the emergency descent, simulation matches the paths of the aircraft at Überlingen guaranteeing a collision (within defined range of error). In both cases, TCAS did not instruct DHL to climb because BTC was above DHL at that time and of course had not begun its descent.

When ATCO intervenes after the RA, the BTC pilots in the simulations ignore the RA advice and continue level flight, which itself averts the collision, even though

Table 15.1 Outcomes of ten simulation runs of Überlingen scenario. Bold indicates greatest potential for collision (ATCO intervenes between TA and RA; both aircraft descending)

Run	Collide?	Explanation	ATCO-BTC	TCAS RA-DHL	ATCO relative TA/RA
1	No	TCAS detects BTC plane descending due to ATCO; so advises DHL to Climb	Descend	Climb	Before
2	No	TCAS detects BTC plane descending due to ATCO; so advises DHL to Climb	Descend	Climb	Before
3	No	TCAS detects BTC plane descending due to ATCO; so advises DHL to Climb. AEF flight arrives very late after TCAS TA	Descend	Climb	Before
4	No	DHL TCAS Descend; BTC above. Planes crossed >100 ft vertical separation	Descend	Descent	During
5	YES	DHL TCAS Descend; BTC above. BTC AP turned off at DHL RA. Planes crossed <20 ft vertical separation	Descend	Descent	During
6	No	DHL TCAS Descend; BTC above. ATCO later than RA, so BTC level. Planes crossed >600 ft vertical separation	Descend	Descent	After
7	No	DHL TCAS Descend; BTC above. DHL turned off 2 s before BTC. Planes crossed >100 ft vertical separation	Descend	Descent	During
8	YES	DHL TCAS Descend; BTC above. Planes crossed <50 ft vertical separation	Descend	Descent	During
9	No	DHL TCAS Descend; BTC above. Planes crossed >200 ft vertical separation	Descend	Descent	During
10	No	DHL TCAS Descend; BTC above. ATCO later than RA, so BTC level. Planes crossed >600 ft vertical separation	Descend	Descent	After

ATCO advises BTC to descend (which implies ignoring that DHL is below them). We of course do not know what the BTC pilots would have done if ATCO had not intervened. With more than one pilot interpreting TCAS correctly, it appears possible the BTC would have climbed.

The final AEF handoff (directing the pilots to contact the tower) always occurs in the simulation after the TCAS RA; at Überlingen, it occurred prior to the TA. This discrepancy raises many questions about what variability is desirable. In the verification of the system, we were able to find certain cases where the final AEF handoff occurs before the TCAS TA and the planes collide.

15.5 Related Work

Several research groups have worked on modeling and analysis of HMI systems and properties and have taken a variety of approaches to the problem. There has been been work on fomalising user models such as CCT and PUMS since the 1980s, but most of these modeling approaches do not focus on the verification aspect Young et al. (1989), Butterworth and Blandford (1997), Bovair et al. (1990).

Campos and Harrison (2008, 2011) propose a framework for analyzing HMI using model checking. They define a set of generic usability properties (Campos and Harrison 2008), such as the possibility to undo an action. These properties can be expressed in a modal logic called MAL and checked with a model checker on the system. This approach targets specific and precise usability properties, whereas our approaches revolve around the higher-level "full-control" characteristic that we defined for an HMI system and as such is complementary to their analysis.

Thimbleby and Gow (2007), Thimbleby (2010) use graphs to represent models. They study usability properties of the system by analyzing structural properties of graphs such as 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. (2007) 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. (2001), Bastide et al. (2003) also developed a framework to analyze interactive systems. Their focus is on the combination of user task models and system models, which we have also explored in the context of our work (Combéfis 2009).

Bolton et al. (2008, 2011), Bolton and Bass (2010) developed a framework used to help predicting human errors and system failures. Models of the system are analyzed against erroneous human behavior models. The analysis is based on task-analytic models and taxonomies of erroneous human behavior. All those models are merged into one model which is then analyzed by a model checker to prove that some safety properties are satisfied. Modeling human error is something that can be incorporated into our Brahms models and taken into account in our analysis.

Bredereke and Lankenau (2002, 2005) 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 analyze systems modeled as Input-Output Labeled Transition Systems (IOTS) (Tretmans 2008). 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.

The works discussed above all focus on analysis of HMI properties but none of them have looked at the issue of facilitating the modeling and analysis of complex multi-agent systems. In a way, this is an orthogonal concern, and the techniques discussed above can be incorporated with a framework like Brahms.

There has been a large body of work in the verification safety-critical systems in the domain of civil aviation in the US as well as in Europe. The DO-178B titled *Software considerations in airborne systems and equipment certification* is the official guideline for certifying avionics software. Several model checking and formal verification techniques have been employed to verify avionic *software* in Miller et al. (2010), Ait Ameur et al. (2010) in accorrdance with the DO-178B. Recent work describes how changes in aircraft systems and in the air traffic system pose new challenges for certification, due to the increased interaction and integration (Rushby 2011). The certification is defined for the deployed software.

For the verification of model-based development, in Miller et al. (2010), the authors present a framework that supports multiple input formalisms to *model* avionic software: these include MATLAB Simulink/Stateflow and SCADE. These formalisms are then translated into an intermediate representation using Luster, a standard modeling language employed to model reactive systems with applications in avionics. Finally, Lustre models are translated to the input language of various model checkers, including NuSMV, PVS, and SAL. These models, however, do not account for the behavior of the human operators.

The work of Yasmeen and Gunter (2011) deals with the verification of the behavior of human operators to check the robustness of mixed systems. In this approach, the authors employ concurrent game structures as the modeling language and translate the verification problem to a model checking instance using SPIN. Our approach is different in that we do not perform syntactic translations, and we reason explicitly about probabilities and beliefs of the agents in the model.

15.6 Conclusions and Future Work

NASA has engaged in research to develop new analytical techniques that can model human–machine interactions and represent the real complexity of the NAS. These techniques represent and analyze scenarios with single users interacting with a single piece of automation as well as multiple humans interacting with multiple automated systems. In this chapter, we described our efforts in developing HMI-specific formal analysis algorithms, connecting them to models that domain experts are familiar with, and finally moving toward multi-agent modeling formalisms that are able to capture in a more scalable fashion the intricate interactions between agents in the NAS. We illustrated our work with a variety of relevant case studies.

In the future, we want to build a fast time simulation and analysis framework based on these technologies to evaluate design concepts for Air Traffic Management operations. Our ultimate goal is to provide algorithms and tool support for the quantitative analysis of safety, performance, and workload for human operators early in the design process. This will allow airspace designers to evaluate various design concepts before implementing them.

To this end, in recent ongoing work, we have developed Brahms models for arrivals and departures at the La Guardia airport based on the work on Departure Sensitive Arrival Scheduling (DSAS) concept developed by the Airspace Operations Lab (AOL) at NASA Ames (Lee et al. 2015). The DSAS concept provides the ability to maximize departure throughput at the LaGuardia Airport in the New York metroplex without impacting the flow of the arrival traffic; it was part of a research effort to explore solutions to delays incurred by departures in the New York metroplex. We are able to successfully model DSAS scenarios that consist of approximately 1.5 h real flight time. During this time, there are between 130 and 150 airplanes being managed by four enroute controllers, three TRACON controllers, and one tower controller at LGA who is responsible for departures and arrivals. The planes are landing at approximately 36–40 planes an hour (Rungta et al. 2016). On this model, airspace designers can evaluate different design candidates in terms of the safety, performance, and workload. Our goal is to then turn the modeled constructs into templates for a general framework that would allow airspace designers to extend components based on their needs.

References

Ameur YA, Boniol F, Wiels V (2010) Toward a wider use of formal methods for aerospace systems design and verification. Int J Softw Tools Technol Transf 12(1):1–7. ISSN 1433-2779. doi:10.1007/s10009-009-0131-4

Bastide R, Navarre D, Palanque P (2003) A tool-supported design framework for safety critical interactive systems. Interact Comput 15(3):309–328

Bolton M, Siminiceanu R, Bass E (2011) A systematic approach to model checking human-automation interaction using task analytic models. IEEE Trans Syst Man Cybern Part A: Syst Hum 41(5):961–976

Bolton M, Bass E (2010) 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, pp 992–996

Bolton M, Bass E, Siminiceanu R (2008) Using formal methods to predict human error and system failures. In: Proceedings of the second international conference on applied human factors and ergonomics (AHFE 2008)

Bovair Susan, Kieras DE, Polson PG (1990) The acquisition and performance of text-editing skill: a cognitive complexity analysis. Hum Comput Interact 5(1):1–48

Bratman M (1987) Intention, plans, and practical reason

Bredereke J, Lankenau A (2002) A rigorous view of mode confusion. In: Proceedings of the 21st international conference on computer safety, reliability and security (SAFECOMP 2002). Springer, pp 19–31, Sept 2002

Bredereke J, Lankenau A (2005) Safety-relevant mode confusions–modelling and reducing them. Reliab Eng Syst Saf 88(3):229–245

Butterworth R, Blandford A (1997) Programmable user models: the story so far. Middlesex University, London

Campos JC, Harrison MD (2008) Systematic analysis of control panel interfaces using formal tools. In: Nicholas Graham TC, Palanque PA (eds) Proceedings of the 15th international workshop on design, specification and verification of interactive systems (DSV-IS 2008), vol 5136. Springer, Lecture notes in computer science, pp 72–85

Campos JC, Harrison MD (2011) Model checking interactor specifications. Autom Softw Eng 8(3):275–310

Clancey WJ, Sachs P, Sierhuis M, Van Hoof R (1998) Brahms: simulating practice for work systems design. Int J Hum Comput Stud 49(6):831–865

Combéfis S (2009) Operational model: integrating user tasks and environment information with system model. In: Proceedings of the 3rd international workshop on formal methods for interactive systems, pp 83–86

Combéfis S (2013) A formal framework for the analysis of human-machine interactions. PhD thesis, Université catholique de Louvain

Combéfis S, Giannakopoulou D, Pecheur C (2016) Automatic detection of potential automation surprises for ADEPT models. IEEE Trans Hum Mach Syst Spec Issue Syst Approaches Hum Mach Interface Improv Resil Robust Stab 46(2):

Combéfis S, Giannakopoulou D, Pecheur C, Feary M (2011) A formal framework for design and analysis of human-machine interaction. In: Proceedings of the IEEE international conference on systems, man and cybernetics, Anchorage, Alaska, USA, 9–12 Oct 2011. IEEE, pp 1801–1808. ISBN 978-1-4577-0652-3. doi:10.1109/ICSMC.2011.6083933

Combéfis S, Pecheur C (2009) A bisimulation-based approach to the analysis of human-computer interaction. In: Calvary G, Nicholas Graham TC, Gray P (eds) Proceedings of the ACM SIGCHI symposium on engineering interactive computing systems (EICS'09)

Curzon P, Rukšėnas R, Blandford A (2007) An approach to formal verification of human-computer interaction. Formal Aspects Comput 19(4):513–550

Feary MS (2010) A toolset for supporting iterative human—automation interaction in design. Technical Report 20100012861, NASA Ames Research Center, March 2010

Godefroid P (1997) Model checking for programming languages using verisoft. In: Proceedings of the 24th ACM SIGPLAN-SIGACT symposium on principles of programming languages. ACM, pp 174–186

Heymann M, Degani A (2007) Formal analysis and automatic generation of user interfaces: approach, methodology, and an algorithm. Hum Factors: J Hum Factors Ergon Soc 49(2):311–330

Hunter J, Raimondi F, Rungta N, Stocker R (2013) A synergistic and extensible framework for multi-agent system verification. In: Proceedings of the 2013 international conference on autonomous agents and multi-agent systems. International Foundation for Autonomous Agents and Multiagent Systems, pp 869–876

JavaPathfinder (JPF) (2016). http://babelfish.arc.nasa.gov/trac/jpf/

Javaux D (2002) A method for predicting errors when interacting with finite state systems. How implicit learning shapes the user's knowledge of a system. Reliab Eng Syst Saf 75:147–165

Johnson J, Henderson A (2002) Conceptual models: begin by designing what to design. Interactions 9:25–32

Lee PU, Smith NM, Homola J, Brasil C, Buckley N, Cabrall C, Chevalley E, Parke B, Yoo HS (2015) Reducing departure delays at laguardia airport with departure-sensitive arrival spacing (dsas) operations. In: Eleventh USA/Europe air traffic management research and development seminar (ATM)

Leveson NG, Turner CS (1993) Investigation of the therac-25 accidents. IEEE Comput 26(7):18–41

Miller SP, Whalen MW, Cofer DD (2010) Software model checking takes off. Commun ACM 53(2):58–64. ISSN 0001-0782. doi:10.1145/1646353.1646372

Navarre D, Palanque P, Bastide R (2001) Engineering interactive systems through formal methods for both tasks and system models. In Proceedings of the RTO Human Factors and Medicine Panel (HFM) specialists' meeting, pp 20.1–20.17, June 2001

Rungta N, Brat G, Clancey WJ, Linde C, Raimondi F, Seah C, Shafto M (2013) Aviation safety: modeling and analyzing complex interactions between humans and automated systems. In: Proceedings of the 3rd international conference on application and theory of automation in command and control systems. ACM, pp 27–37

Rungta N, Mercer EG, Raimondi F, Krantz BC, Stocker R, Wallace A (2016) Modeling complex air traffic management systems. In: Proceedings of the 8th international workshop on modeling in software engineering. ACM, pp 41–47

Rushby JM (2011) New challenges in certification for aircraft software. In: Chakraborty S, Jerraya A, Baruah SK, Fischmeister S (eds) EMSOFT. ACM, pp 211–218. ISBN 978-1-4503-0714-7

Sierhuis M (2001) Modeling and simulating work practice. BRAHMS: a multiagent modeling and simulation language for work system analysis and design. PhD thesis, Social Science and Informatics (SWI), University of Amsterdam, SIKS Dissertation Series No. 2001-10, Amsterdam, The Netherlands

Thimbleby H (2010) Press on: principles of interaction programming. The MIT Press, Jan 2010. ISBN 0262514230

Thimbleby H, Gow J (2007) Applying graph theory to interaction design. In Gulliksen J, Harning MB, Palanque P, van der Veer G, Wesson J (eds) Proceedings of the engineering interactive systems joint working conferences EHCI, DSV-IS, HCSE (EIS 2007). Lecture notes in computer science, vol 4940, pp 501–519. Springer, Mar 2007

Tretmans J (2008) Model based testing with labelled transition systems. In: Hierons R, Bowen J, Harman M (eds) Formal methods and testing. Lecture notes in computer science, vol 4949. Springer, pp 1–38

uberlingeng (2004) Investigation Report AX001-1-2/02. Technical report, German Federal Bureau of Aircraft Accidents Investigation

Yasmeen A, Gunter EL (2011) Automated framework for formal operator task analysis. In Dwyer MB, Tip F (eds) ISSTA. ACM, pp 78–88

Young RM, Green TRG, Simon T (1989) Programmable user models for predictive evaluation of interface designs. In: ACM SIGCHI bulletin, vol 20. ACM, pp 15–19