Detection of Pilot Errors in Data by combining Task Modeling and Model Checking

Florian Frische, Tomasz Mistrzyk and Andreas Lüdtke

OFFIS e.V., Escherweg 2,Oldenburg, Germany
{Florian.Frische, Tomasz.Mistrzyk, Andreas.Luedtke}@offis.de

Abstract. In this paper we show a consistent approach of using Hierarchical Task Analysis together with model checking to identify pilot errors during the interaction with cockpit automation systems in aircraft. Task analysis is used to model flight procedures which describe how to operate a specific system in a particular situation. Afterwards model checking is used to identify deviations from these procedures in empirical simulator data. We envision applying this method to automatically detect pilot errors during flight tests or pilot training.

Keywords: Hierarchical Task Analysis, Model Checking, Error Analysis

1 Hierarchical Tasks Analysis of Normative Pilot Activities

Our goal is to identify aircraft pilot errors (e.g. omission of actions) during the interaction with cockpit automation systems while performing specific flight tasks. This is done based on two inputs: (1) Information about how the flight task has to be performed normatively and (2) data on the actual performance (recorded in flight simulations) of the task. We define pilot errors as deviations from normative activities. To get the first input we perform a hierarchical task analysis to produce a hierarchical task model.

There are various approaches supporting analysis and modeling of tasks. Semi-formal task analysis and modeling approaches such as [5, 6] primarily concentrate on the hierarchical decomposition of tasks into subtasks and their temporal ordering. The models are semi-formal in the sense that they formally structure (hierarchy, temporal relations) informal elements (task descriptions in natural language). Formal task modeling approaches define a granular formal structure for the task descriptions, e.g. in form of Goals, Operators, Methods and Selection Rules (like in GOMS [4]). The result of a task analysis is a hierarchical task model. Task models are typically used in the early phases of system design [8] or user interface design [4] to get valuable information about task performance. They are also increasingly applied for the analysis of workplace situations, especially for human error analysis [1].

In our approach we implement a tool chain which combines semi-formal and formal task analysis and modeling. The tool chain begins with AMBOSS [5], a tool for the semi-formal level of task analysis and modeling. This modeling environment provides different abstraction levels in a hierarchical manner, represented graphically in a tree-like format. AMBOSS offers different features for task inspection like task order specification, task duration and task criticality as well as the description of the task environment. Similar to other approaches [7] AMBOSS also provides simulation of semi-formal task models to support the experts in the analysis of the plausibility of such models. To go from semi-formal to formal task modeling AMBOSS supports the export of the task model as an XML output file which can be imported by PED.

PED (Procedure Editor) is a modeling environment for creating formal task models. With the formal task modeling the designer gets a highly precise description of the environment that the operator interacts with. To achieve such a fine granular level we developed a rule based language which is based on the well-known GOMS [4] language. Using PED we formalize procedures as a set of rules. Each rule contains a left- and a right-hand side. The left-hand side defines the conditional part. Every rule is assigned to exactly one goal and is applicable if the conditional part evaluates to true. Conditions contain system variables (e.g. current auto pilot mode) or environmental variables (e.g. current altitude of the aircraft). The right-hand side specifies the method, which is a sequence of different types of actions. Actions are functional entities distinguished in percept, motor, vocal, memory retrieval and memory store actions. This language allows us to organize a flight procedure in a tree-like structure of goals and sub-goals, methods and operators to achieve a certain goal on a fine granular level.

After we have analyzed and modeled the normative pilot activities we have to produce data on actual performance of pilots during a flight task. The normative activities and the data are then used to find deviations from normative activities and to derive actual pilot errors. These steps are described in the next section.

2 Identifying Actual Pilot Errors based on Actual and Normative Pilot Activities

Data on actual task performance is recorded during experiments with pilots in flight simulations. The recorded data contain aircraft states (e.g. current speed), pilot actions (e.g. eye-movements) and environment states (e.g. weather conditions). The flight simulations and data recording will be performed by DLR (German Aerospace Center) with whom we cooperate in the European project HUMAN.

Based on the raw data we produce an activity trace. The activity trace consists of time stamped data rows. Each row in the activity trace represents a pilot action or the beginning / the ending of a task with the corresponding aircraft and environment states at a given time. Deviations from normative activities are detected by automatically comparing normative activities (rules) with actual activities (activity trace). These deviations are then classified in two types: (1) additional normative activities that have not been identified during the tasks analysis before - in this case the normative activities have to be extended - or (2) true errors. Experts have to decide whether deviations are the former or the latter.

Deviations of type (2) are classified with regard to error types (ETs), e.g. omission of actions or goals. The allocation of errors to an ET is done by interpreting an erroneous action in context of the task that a pilot is performing and environmental / aircraft conditions.

Model Checking is a powerful approach for automatic system verification. Meanwhile there are approaches which try to use model checking on user behaviour with the intention to enhance human-machine interfaces [2] and to analyze erroneous actions [3]. We are currently developing a method based on formal automata and model checking to perform the detection of pilot errors and ET classification automatically. This requires a transformation of the normative and actual activities into automata. In addition, we have chosen a set of ETs that have to be available in automata as well. These three types of automata are formally integrated into a comprehensive model in order to apply model checking techniques for the automatic detection of pilot errors. We use the model checking tool UPPAAL [9]. Fig. 1 depicts our approach.

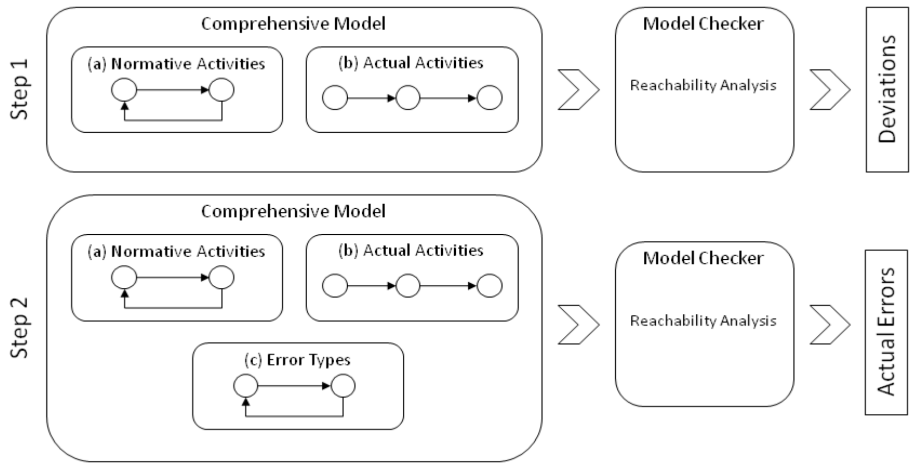

Fig. 1. Automatic detection of pilot errors by analyzing a comprehensive automata model of normative activities, actual activities and error types using a model checker.

We perform the error detection in 2 steps: First, we only integrate the automata for the normative activities and a pilot activity trace (a and b in Fig. 1). We apply the model checker to test if the actual activity automaton can be successfully synchronized with the normative activity automata. This is done by a reachability analysis for the final state of the actual activity automaton. The synchronization between the automata is modeled by communication variables (channels in UPPAAL) for each pilot action. Every transition in the activity trace has a channel which sends a message to the normative automata. A transition is only possible if at least one normative activity automata can understand the message and perform a transition itself. In this way the reachability analysis will only be successful if the actions in the activity trace are normative.

The model checker denotes deviations from normative activities as deadlocks. In a second step we subsequently add the ET automata (c in Fig. 1) to the integrated model and run the query again (individually for each ET automaton). The ET automata include communication variables similar to those in the normative activity automata. A true pilot error has been identified if one of the ET automata solves the deadlock.

4 Conclusion

In this paper we presented a comprehensive approach to analyze and model normative flight procedures in a stepwise approach from semi-formal to formal. The resulting formal task model is used to automatically identify pilot errors in data that has been recorded in flight simulations. Preliminary tests with the model checking tool UPPAAL showed that the method produces results for some normative activities. But, we found that the current construction of the automata is not powerful enough to cover all aspects of normative flight procedures: information about tasks on a low level and some types of temporal relations between tasks could not be handled. We tackle these problems in the next steps by extending our task language as well as the automata structure. We envision applying our method for the automatic detection of pilot errors during flight tests in a simulator or during pilot training. In the first case the automatic identification of pilot errors can support highlighting weaknesses of the cockpit design to drive subsequent design improvements. In the second case the analysis of the trainees' performance and the design of individual training sessions can be supported.

The work described here is funded by the European Commission in the 7th Framework Programme within the project HUMAN (FP7-211988, www.human.aero).

References

1. Basnyat, S., Bastide B.: Error Patterns: Systematic Investigation of Deviations in Task Models, In Proceedings of the 5th International Workshop, TAMODIA 2006, Hasselt, Belgium, pp. 109-121 Springer, Heidelberg (2006)
2. Basuki T.A., Cerone A., Griesmayer A., Schlatte R.: Model-checking user behaviour using interacting components, In: Journal of Formal Aspects of Computing (FACS), Springer (2009)
3. Fields R.E.: Analysis of erroneous actions in the design of critical systems, University of York (2001)
4. John, B. E., Kieras, D. E.: Using GOMS for user interface design and evaluation: which technique?, *ACM Trans. Comput.-Hum. Interact.* 3, pp. 287-319 (1996)
5. Giese, M., Mistrzyk, T., Pfau, A., Szwillus, G., von Detten M.: AMBOSS: A Task Modeling Approach for Safety-Critical, In Proceedings of 7th International Workshop on Task Models and Diagrams, TAMODIA 2008, Pisa, Italy, pp. 98-108 Springer, (2008)
6. Lu, S., Paris, C., Vander Linden, K.: Tamot: Towards a Flexible Task Modeling Tool. In: The Proceedings of Human Factors, Melbourne, Australia, pp. 25-27 (2002)
7. Mori, G., Paternò, F., Santoro, C.: CTTE: support for developing and analyzing task models for interactive-system design. In: IEEE Transactions on, Software Engineering, Bde. Volume 28, Issue 8, IEEE Press, pp. 797-813 (2002)
8. Navarre, D., Palanque, P.A., Barboni, E., Mistrzyk, T.: On the Benefit of Synergistic Model-Based Approach for Safety Critical Interactive System Testing. In: Proceedings of the 6th Internationals Workshop, TAMODIA 2007, Toulouse, France, LNCS, vol. 4849, pp. 140-154 Springer, Heidelberg (2007)
9. UPPAAL. Project homepage http://www.uppaal.com