V&V of Lexical, Syntactic and Semantic Properties for Interactive Systems through Model Checking of Formal Description of Dialog

Guillaume Brat[1], Célia Martinie[2], and Philippe Palanque[2]

[1] NASA Ames Research Center, MS-269-1 Moffett Field, California, USA
guillaume.p.brat@nasa.gov
[2] IRIT, Université Paul Sabatier, 118, route de Narbonne
31062 Toulouse Cedex 9, France
{martinie,palanque}@irit.fr

Abstract. During early phases of the development of an interactive system, future system properties are identified (through interaction with end users in the brainstorming and prototyping phase of the application, or by other stakeholders) imposing requirements on the final system. They can be specific to the application under development or generic to all applications such as usability principles. Instances of specific properties include visibility of the aircraft altitude, speed… in the cockpit and the continuous possibility of disengaging the autopilot in whatever state the aircraft is. Instances of generic properties include availability of undo (for undoable functions) and availability of a progression bar for functions lasting more than four seconds. While behavioral models of interactive systems using formal description techniques provide complete and unambiguous descriptions of states and state changes, it does not provide explicit representation of the absence or presence of properties. Assessing that the system that has been built is the right system remains a challenge usually met through extensive use and acceptance tests. By the explicit representation of properties and the availability of tools to support checking these properties, it becomes possible to provide developers with means for systematic exploration of the behavioral models and assessment of the presence or absence of these properties. This paper proposes the synergistic use two tools for checking both generic and specific properties of interactive applications: Petshop and Java PathFinder. Petshop is dedicated to the description of interactive system behavior. Java PathFinder is dedicated to the runtime verification of Java applications and as an extension dedicated to User Interfaces. This approach is exemplified on a safety critical application in the area of interactive cockpits for large civil aircrafts.

1 Introduction

Nowadays interactive applications are more and more required to handle the complexity of command and control systems for safety critical applications. Formalisms, processes and tools are then required to bring together several properties such as reliability, dependability and operability. In addition to standard properties of computer systems (such as safety or liveness), interaction properties have been identified. Properties related to the usage of an interactive system are called external properties [2] [9] and characterize the capacity of the system to provide support for its users to accomplish their tasks and goals, potentially in several ways, and prevent or help to recover from errors. Although all types of properties are not always completely independent one from each other, external properties are related to the user's point of view and usability factor, whereas internal properties are related to the design and development process of the system itself (modifiability, run time efficiency). Interactive systems have to support both types of properties and dedicated techniques and approaches have been studied for this purpose, amongst them are formal methods. Formal languages have proven their value in several domains and are a necessary condition to understand, design and develop systems and check their properties.

Formal methods are studied since several years in the field of HCI as a mean to analyze in a complete and unambiguous way interactions between a user and a system. Several types of approaches have been developed [8], which encompass contributions about formal description of an interactive system and/or formal verification of its properties. Amongst these approaches, ICO description technique and associated Petshop CASE tool, provide augmented support for describing the conceptual model of the system but also for analysis and validation at earlier stage in the process [10]. This kind of approaches provide support for describing exhaustively interactive systems and their behavior, as well as prototyping, testing and verifying synchronously certain types of properties. However, as existing notations to produce executable models are quite expressive, models cannot be verified until they have been translated into more abstract models in order to perform properties verification. This paper proposes to associate Petshop to JPF, a framework for runtime verification of Java programs. This association provides support for complete and non-ambiguous description of an interactive application (with Petshop) as well as formal verification of the ICO specification based interactive application (with JPF). Next section is dedicated to the presentation of the tool suite. Third section illustrates the use of this tool suite for the example of the Weather Radar aircraft cockpit application. Fourth section is dedicated to related work.

2 A Tool Suite for the Validation and Verification of Interactive Systems

This tool suite is composed of two software tools: Petshop and JavaPathFinder. Petshop [3] is dedicated to the description of interactive system behavior. JavaPathFinder [4] is dedicated to the runtime verification of Java applications and as an extension dedicated to User Interfaces.

2.1 Petshop

The CASE tool Petshop[1] is Java based and provides support for editing ICO models. Interactive Cooperative Object (ICO) [16] is an object-oriented formal notation dedicated to interactive systems. It provides support for describing: the events to which the application can react, the set of functions it can perform and the implicit set of states in which the system can be. This formalism encompasses both the "input" aspects of the interaction (i.e. how user actions impact on the inner state of the application, and which actions are enabled at any given time) and its "output" aspects (i.e. when and how the application displays information relevant to the user). ICO notation is based on Petri nets and Petshop tool then allows classical manipulations on the Petri nets add/remove/modify (Petri net items, marking, code within transitions, etc.) and offers classical editing services (copy/cut/paste, undo/redo, navigation amongst the models, etc.). Petshop also enables to execute simultaneously the interactive application as well as its underlying models. Furthermore, it is possible to modify the models while the modeled interactive application is running. This list of features enables to formally specify, test and validate an interactive application in early stages of the development process, using Petshop as a high-fidelity prototyping tool [19]. However, properties verification activities require transformation of the high-fidelity prototype. Fig. 1 represents diagrammatically the current existing process for verifying properties of interactive applications running in the Petshop environment.

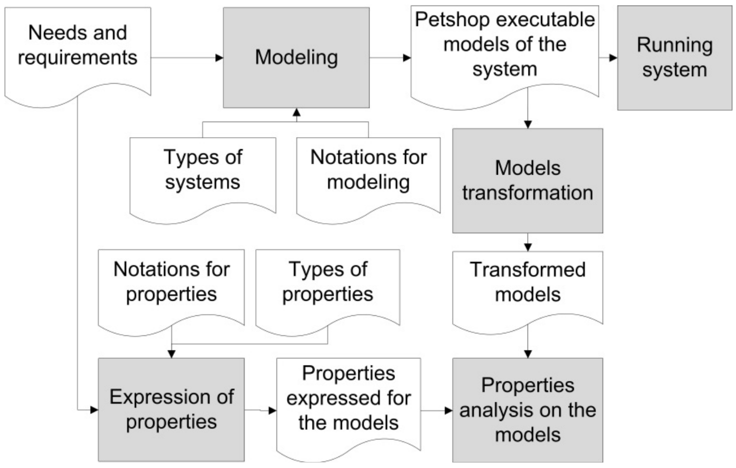


Fig. 1. Existing process for verifying properties of Petshop running interactive applications

The analysis performed at runtime within Petshop environment is executed on translated version of ICO models. The translated models are low-level Petri nets and the Petshop analysis module performs an invariant Analysis (P/T invariants) of the underlying Petri net. Previous work has shown that it is also possible to perform formal manual analysis of interactive system properties [17] as well as ergonomic rules

[1] `http://www.irit.fr/recherches/ICS/softwares/petshop/`

[18], both using ACTL notation for expressing properties. However, as previously described, ICO notation is quite expressive and ICO models cannot be verified until they have been translated into more abstract models in order to perform properties verification.

2.2 JPF

Java PathFinder[2] (JPF) [4, 23] is a framework for the runtime verification of Java programs. It can be used as an explicit-state model checker that works directly on Java bytecode. JPF specializes in finding deadlocks, verifying assertions, and checking temporal logic specifications through the use of listeners, which monitors the exploration of all possible paths in a Java program and continuously check that the specifications are met. JPF explores all possible interleavings in multi-threaded programs as well as all possible choice points (both in terms of control logic and data values) corresponding to a specific environment.

JPF has many extensions, which can be used to process various languages (e.g., UML-style Statecharts, Scala), the use of specific Java libraries (e.g., network communciations, java.awt or javax.swing), or, to switch from model checking to a less exhaustive form of verification (e.g., symbolic execution, concolic execution or runtime analysis). In our case, we are using mostly jpf-awt, which provides convenient abstractions of user interface libraries (awt or swing) and means to model interacting users through scripts [24].

Complex functional properties, corresponding to LTL [21] safety properties, can be expressed and checked using JPF listeners. Listeners are Java programs that run in parallel with the execution done by JPF, monitor the states of the application under test at every step, and check them against the specified formal property. JPF listeners are commonly referred to as observers in the model checking community. So, using listeners does not require any modification of the original application code; it does require knowledge of that code though. Listeners can easily encode LTL safety properties such as the Property *P* described above. However, they cannot express liveness properties (in layman's terms, asserting that something good eventually happens), e.g., stating that a value is eventually displayed. Liveness properties can be checked by JPF only using the jpf-ltl extension, which has not been tested extensively. This capability can be useful to check properties such as checking that a progression bar keeps moving or that a button eventually gets reset.

2.3 Process for Formal Description and Verification of Interactive Applications

Fig. 2 summarizes the proposed process. Functional as well as non-functional needs and requirements are identified at the beginning of the process (top left part in Fig. 2).

[2] `http://babelfish.arc.nasa.gov/trac/jpf/`

Interactive application is then formally described using Petshop tool. In parallel, properties that have to be verified by the application are represented using temporal logic notations. JPF is then used to verify properties directly on the running interactive application.

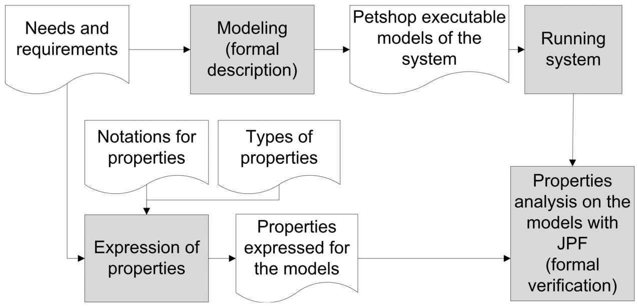

Needs and requirements
Modeling (formal description)
Petshop executable models of the system
Running system
Notations for properties
Types of properties
Expression of properties
Properties expressed for the models
Properties analysis on the models with JPF (formal verification)

Fig. 2. Proposed verification process integrating Petshop and JPF

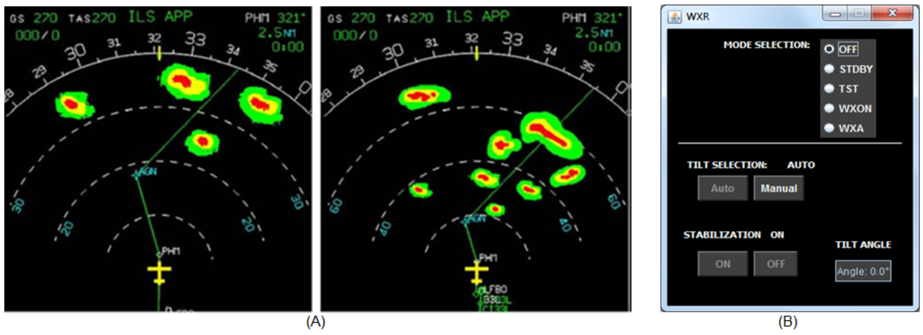
Following section provides an example of applying the tool suite for the implementation and verification of a civil aircraft cockpit interactive application.

3 Illustrative Example: WXR Application

Weather Radar application (also named WXR) has been modeled taking into account ARINC 661 [1] standard, which aims at defining software interfaces to the Cockpit Display System (CDS) used in all types of aircraft installations.

3.1 Weather Radar Presentation

Weather radar is an application currently deployed in many cockpits of commercial aircrafts. It provides support to pilot's activities by increasing their awareness of meteorological phenomena during the flight journey, allowing them to determine if they may have to request for a trajectory change, in order to avoid storms or precipitations for example. Fig. 3A shows screenshots of weather radar displays. Fig. 3B presents a screenshot of the weather radar control panel. This panel provides two functionalities to the crew members. The first one is dedicated to the mode selection of weather radar and provides information about status of the radar, in order to ensure that the weather radar can be set up correctly. The second functionality, available in the lower part of the window, is dedicated to the adjustment of the weather radar orientation (Tilt angle). This can be done in an automatic way or manually. Additionally, a stabilization function aims at keeping the radar beam stable even in case of turbulences. It shall not be possible to manually edit the Tilt angle if the application is in automatic mode or if the stabilization functionality is on.



WXR
MODE SELECTION: [✓ OFF] STDBY TST WXON WXA
TILT SELECTION: AUTO
Auto Manual
STABILIZATION ON
ON OFF
TILT ANGLE
Angle: 0.0°

Fig. 3. A - Screenshot of weather radar displays, B - Screenshot of the weather radar control panel

3.2 Formal Specification of WXR Application with Petshop

Fig. 4 presents a diagrammatic overview of the WXR application running in the Petshop environment. The following parts of the application are represented distinctively: the presentation part of the application (User Interface), the behavioral part of the application (Cooperative Objects models) and the functions that bind the presentation part to the behavioral part: activation and rendering functions.

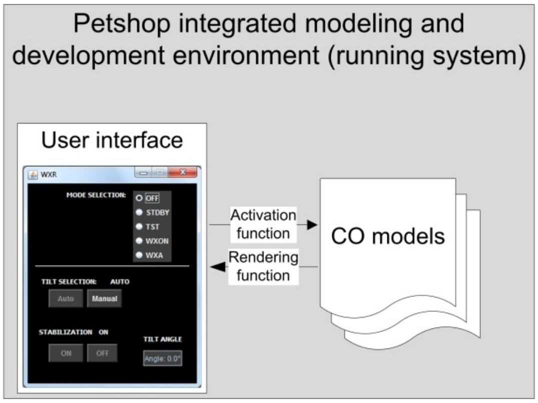


Fig. 4. Overview of the WXR application running in the Petshop environment

The weather radar control panel application is composed of the presentation part (shown in Fig. 3B), the Cooperative Object (CO) model (shown in Fig. 5), and the activation and rendering functions. Activation and rendering functions are not presented in this article and the interested reader can find example in [16]. The Cooperative Object shown in Fig. 5 is the formal description of the WXR application's behavior. This formal description is used as part of the specification for developing the final application running on the targeted system. JPF tool is then used to verify properties against the final application as described in the following paragraphs.

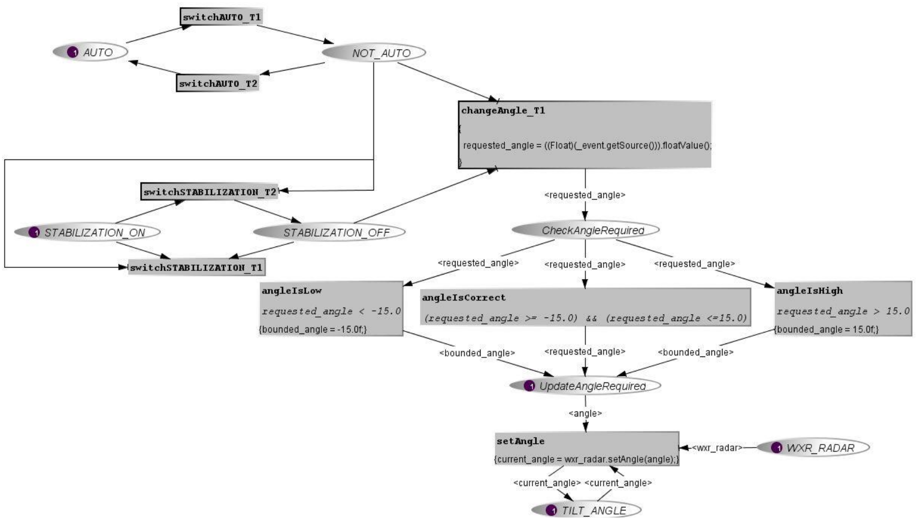


Fig. 5. Cooperative Object (CO) model of the weather radar control panel

3.3 Formal Verification of WXR Application with JPF

Our goal is to demonstrate that we can automate the exhaustive verification of formal properties on interactive systems using the WRX application. For that we use the JPF model checker, and more precisely jpf-awt, a JPF extension for model checking applications making use of the java.awt and java.swing libraries [24]. This extension provides means of modeling user inputs (e.g., pressing buttons, entering text, selecting items) and understanding awt (or swing) method calls through abstractions.

The first step in using JPF to model-checked interactive applications consists of identifying user input scenarios of interest and capturing them using scripts that are passed to *jpf-awt*. This allows us to "close the system" (in model checking jargon), which in this case means that the verification can be done without human interaction. The scripts use a simple scripting language to describe input sequences of interest; in some ways, they describe ranges of possible interactions with the application. In the case of WRX, we can define a range of scenarios starting with some mode selection and then allowing a user to play freely with tilt selection, then stabilization and finally setting a range of tilt angle. This corresponds to the following script, in which the ANY keyword indicates a random choice between different options:

```
ANY {$MODE_SELECTION.select()}
ANY {NONE | $MANUAL.doClick() | $AUTO.doClick()}
ANY {NONE | $ON.doClick() | OFF.doClick()}
$TILT_ANLGE:input.setText("whatever")
```

The second step consists of capturing a formal property representing functional or non-functional requirements. Non-functional, systemic, requirements such as requiring the absence of deadlocks or other "Java language issues" are checked automatically by JPF. Functional requirements can be expressed, and thus checked, by various means. Assertions are simple instructions, which can be inserted at any point in the user interface code. They simply monitor the value of complex conditions over the values or states of any user interface object. For example, the property, say *P*, stating that "it shall not be possible to manually edit the Tilt angle if the application is in automatic mode or if the stabilization functionality is on" can be checked by placing the assertion based on the following condition

```
(lbl_Selection.getText()!="AUTO") && (lbl_Stabilization.getText()!="ON)
```

in the method that displays a new tilt angle. If *P* can be violated by some of sequence allowed by the input script, then JPF reports the violation and demonstrates it by displaying a trace expressed in terms of the elements described in the script. Assertions are also very useful to check that an entered numerical value is within a specific range, e.g., checking that a title angle value is indeed between 0^o and 360^o. Assertions are also convenient for expressing some simple safety (in layman's term, asserting that nothing bad happens) properties. However using assertions requires instrumenting the code of the application, which is not always desirable.

The third step consists of running JPF and waiting for its report. If the property is verified, JPF returns some statistics about the analysis time and the number of states and threads explored by the analysis. If the property can be violated by the application, JPF returns a counter-example showing a possible violation sequence. This counter-example is produced as a trace of events corresponding to the events used in the script, thus facilitating the understanding of the counter-example by the developer; displaying a trace as a full Java execution trace would be overwhelming to the developer and would be hard to relate to the application.

4 Related Work

Paterno and Santoro [20] proposed an approach based on formal model-checking (with CADP[3] toolset) of LOTOS specifications of dialogue between the user and the system. Another set of approaches are based on the formal verification of state charts. Campos and Harrison [5] proposed an approach based on SMV [15] model-checking of Interactor specifications. Kamel and Ait Ameur [13] also propose an approach to verify properties for multimodal interactions with SMV model checker. Combéfis et al. [7] propose to translate state chart models into Java programs which can then be verified using Java PathFinder model checker. All of these approaches based on state chart models of interactive application behavior do not provide support for complete and non-ambiguous description of concurrent events driven applications (such as multimodal interactive systems). Furthermore, they do not provide support for simultaneous execution of the application prototype from the models, as actually provided

[3] `http://cadp.inria.fr/`

by Petshop. Approaches based on the executability of models provide augmented support for describing the conceptual model of the system and for analysis and verification at earlier stage in the process [10]. This kind of approaches provides support for describing exhaustively interactive systems and their behavior, as well as prototyping, testing and verifying synchronously certain types of properties. APEX-CPN Tools [22] is a framework based on colored Petri nets [22], which provides support for rapid prototyping of ubiquitous environments and a predefined set of algorithms for properties verification. However the underlying notation of this framework is less expressive than the ICO one.

5 Conclusion and Future Work

We presented an approach for supporting validation and verification of interactive applications throughout the whole development process. This approach relies on the synergistic use of Petshop tool for producing formal specification of the application and of JPF tool for formal verification of the developed application. This framework provides support for validation and verification of internal and external properties of an interactive application. Petshop tool is used to produce formal specifications of complex interactive critical applications. JPF tool is then used to verify that the final application built from the specification meets the properties requirements.

The presented work will be followed by an investigation on how to verify properties on ICO models. As they are the formal specification of the interactive application and they are also used as the source code of the application prototype, they could be used as the deployed interactive application itself. Future work is to investigate to which extent JPF tool can be used to directly perform model-checking on ICO running models.

References

1. ARINC 661 specification: Cockpit Display System Interfaces to User Systems, Prepared by AEEC. Published by Aeronautical Radio, Inc. (April 22, 2002)
2. Bass, L., John, B., Juristo Juzgado, N., Sánchez Segura, M.I.: Usability-Supporting Architectural Patterns. In: ICSE 2004, pp. 716–717 (2004)
3. Bastide, R., Navarre, D., Palanque, P.: A Tool-Supported Design Framework for Safety Critical Interactive Systems. Interacting with Computers 15(3), 309–328 (2003)
4. Brat, G., Drusinsky, D., Giannakopoulou, D., Goldberg, A., Havelund, K., Lowry, M., Pasareanu, C., Venet, A., Washington, R., Visser, W.: Experimental Evaluation of Verification and Validation Tools on Martian Rover Software. Journal on Formal Methods in Systems Design 25(2-3) (September 2004)
5. Campos, J.C., Harrison, M.D.: Model Checking Interactor Specifications. Journal of Automated Software Engineering 8(3-4), 275–310 (2001)
6. Clarke, E.M., Emerson, E.A., Sistla, A.P.: Automatic verification of finite-state concurrent systems using temporal logic specifications. ACM Transactions on Programming Languages and Systems 8(2), 244–263 (1986)

7. Combéfis, S., Giannakopoulou, D., Pecheur, C., Feary, M.: A Formal Framework for Design and Analysis of Human-Machine Interaction. In: Proceedings of IEEE System, Man and Cybernetics (SMC), Anchorage, USA, pp. 1801–1808 (2011)
8. Dix, A.: Upside down As and algorithms – computational formalisms and theory. In: Carroll, J. (ed.) HCI Models Theories and Frameworks: Toward a Multidisciplinary Science, ch. 14, pp. 381–429. Morgan Kaufmann, San Francisco (2003)
9. Gram, C., Cockton, G.: Design principles for Interactive Software. Chapman & Hall, London (1996)
10. Fuchs, N.E.: Specifications are (preferably) executable. Journal on Software Engineering 7(5), 323–334 (1992)
11. Hewelt, M., Wagner, T., Cabac, L.: Integrating verification into the PAOSE approach. In: Proceedings of the Petri Nets and Software Engineering. International Workshop PNSE 2011, Newcastle upon Tyne, UK, pp. 124–135 (June 2011)
12. Jensen, K., Kristensen, L.M., Wells, L.: Coloured Petri Nets and CPN Tools for modelling and validation of concurrent systems. International Journal on Software Tools for Technology Transfer 9(3-4), 213–254 (2007)
13. Kamel, N., Ait Ameur, Y.: A Formal Model for CARE Usability Properties Verification in Multimodal HCI. In: Proceeding of IEEE International Conference on Pervasive Services, Istanbul, Turkey, July 15-20, pp. 341–348 (2007)
14. Mascheroni, M., Wagner, T., Wüstenberg, L.: Verifying reference nets by means of hypernets: A plugin for Renew. In: Proceedings of the International Workshop on Petri Nets and Software Engineering, PNSE 2010, Braga, Portugal, pp. 39–54 (2010)
15. McMillan, K.L.: Symbolic Model Checking. Kluwer Academic Publishers (1993)
16. Navarre, D., Palanque, P., Ladry, J.-F., Barboni, E.: ICOs: a Model-Based User Interface Description Technique dedicated to Interactive Systems Addressing Usability, Reliability and Scalability. Transactions on Computer-Human Interaction, ACM SIGCHI 16(4), 1–56 (2009)
17. Palanque, P., Bastide, R.: Verification of an Interactive Software by analysis of its formal specification. In: Proceedings of the IFIP TC13 Interact 1995 Conference, Lillehammer, Norway, June 27-29, pp. 191–197 (1995)
18. Palanque, P., Farenc, C.: Embedding Ergonomic Rules as Generic Requirements in a Formal Development Process of Interactive Software. In: Proceedings of IFIP TC 13 Interact 1999 Conference, Edinburg, Scotland, September 1-4 (1999)
19. Palanque, P., Ladry, J.-F., Navarre, D., Barboni, E.: High-Fidelity Prototyping of Interactive Systems Can Be Formal Too. In: Jacko, J.A. (ed.) HCI International 2009, Part I. LNCS, vol. 5610, pp. 667–676. Springer, Heidelberg (2009)
20. Paternó, F., Santoro, C.: Integrating model checking and HCI tools to help designers verify user interface properties. In: Paternó, F. (ed.) DSV-IS 2000. LNCS, vol. 1946, pp. 135–150. Springer, Heidelberg (2001)
21. Pnueli: The temporal logic of programs. In: Proceedings of the 18th IEEE Symposium on Foundation of Computer Science, pp. 46–57 (1977)
22. Silva, J.L., Campos, J.C., Harrison, M.D.: Formal Analysis of Ubiquitous Computing Environments through the APEX Framework. In: EICS 2012: Proceedings of the 4th ACM SIGCHI Symposium on Engineering Interactive Computing Systems, pp. 131–140 (2012)
23. Visser, W., Havelund, K., Brat, G., Park, S.: Model Checking Programs. In: Proceedings of the 15th IEEE International Conference on Automated Software Engineering (ASE 2000). IEEE Computer Society, Washington, DC (2000)
24. Mehlitz, P.C., Tkachuk, O., Ujma, M.: JPF-AWT: Model checking GUI applications. In: ASE 2011, pp. 584–587 (2011)