



HAL
open science

TOUCAN: An IDE Supporting the Development of Effective Interactive Java Applications

Célia Martinie, David Navarre, Philippe Palanque, Eric Barboni, Alexandre Canny

► **To cite this version:**

Célia Martinie, David Navarre, Philippe Palanque, Eric Barboni, Alexandre Canny. TOUCAN: An IDE Supporting the Development of Effective Interactive Java Applications. ACM SIGCHI conference Engineering Interactive Computing Systems (EICS 2018), Jun 2018, Paris, France. pp.1-7, 10.1145/3220134.3220136 . hal-03623020

HAL Id: hal-03623020

<https://hal.science/hal-03623020>

Submitted on 29 Mar 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Open Archive Toulouse Archive Ouverte

OATAO is an open access repository that collects the work of Toulouse researchers and makes it freely available over the web where possible

This is an author's version published in:

<http://oatao.univ-toulouse.fr/22536>

Official URL

DOI : <https://doi.org/10.1145/3220134.3220136>

To cite this version: Martinie De Almeida, Celia and Navarre, David and Palanque, Philippe and Barboni, Eric and Canny, Alexandre *TOUCAN: An IDE Supporting the Development of Effective Interactive Java Applications*. (2018) In: ACM SIGCHI conference Engineering Interactive Computing Systems (EICS 2018), 19 June 2018 - 22 June 2018 (Paris, France).

Any correspondence concerning this service should be sent to the repository administrator: tech-oatao@listes-diff.inp-toulouse.fr

TOUCAN: An IDE Supporting the Development of Effective Interactive Java Applications

Célia Martinie, David Navarre, Philippe Palanque, Eric Barboni, Alexandre Canny
ICS-IRIT, University Toulouse 3
Toulouse, France
{martinie, navarre, palanque, barboni, acanny}@irit.fr

ABSTRACT

The TOUCAN IDE (Integrated Development Environment) provides support for building effective interactive applications programmed in Java Swing or in JavaFX. Taking into account the effectiveness factor of usability requires from developers to guarantee that the application allows users to reach their goals and to complete their tasks. This means that users' goals and tasks have been analyzed and that their explicit description is available. By providing support for mapping and co-execution of task descriptions with interactive application software, TOUCAN IDE enables to program an interactive application and, at the same time, to support its effectiveness. In this article we highlight the main features of TOUCAN, as well as its underlying principles for enabling the mapping and co-execution of an interactive application with its associated task models.

KEYWORDS

Interactive application programming, task models, co-execution.

1 INTRODUCTION

Ensuring that an interactive application allows users to perform their activities and to reach their goals is essential to the overall usability of the interactive application. The effectiveness factor of usability [3] directly refers to this capability. The interactive application shall implement all of the needed functionalities and these functionalities shall be accessible when needed (e.g. made available to the user in an order that is compatible with the one needed to reach the goals). Ensuring the effectiveness of an interactive application thus requires an explicit description of users' goals and tasks. Task models is one of the very few means for explicitly and exhaustively describing user tasks at design time. However, at the same time, task models are considered as cumbersome, expensive to build and mainly useful in the early phases of the development process, where they provide high-level descriptions for the identification of the main functions of the interactive application. But, ensuring the effectiveness of an interactive application also requires to verify that each user actions are feasible with the interactive application (e.g. being able to click on a button that is enabled at the appropriate time) and this verification should be possible along the whole development process. The TOUCAN IDE (Integrated Development Environment) provides support for such verification. TOUCAN stands for Tasks Objects and Users Connected to Applications Natively. TOUCAN IDE provides support for programming a Java application (Java Swing and Java FX) while ensuring its effectiveness thanks to the integration of the application with its corresponding task models. In this article, we highlight how TOUCAN provides support for:

- checking the effectiveness factor of usability using task models and interactive software application,
- connecting a description of user interactive task in a task model to lines of codes in an interactive software application,
- the co-development of an interactive application and its associated task models.

Next section describes the positioning of TOUCAN with related work on task models based IDE that support the development of interactive applications. The third section presents the TOUCAN IDE and illustrates its main

features with the example of the Automated Teller Machine (ATM) application. The fourth section presents how the underlying parts of TOUCAN contribute to the mapping and co-execution of an interactive application with its associated task models. The last section concludes on the benefits and perspectives for the usage of TOUCAN.

2 BACKGROUND AND POSITIONING WITH REGARDS TO TASK-MODEL BASED IDE

This section aims at highlighting the scope of the TOUCAN IDE compared to the scope of other task models based development environments. Task models are one of the main artefacts in the CAMELEON framework [2]. This framework assumes that task models are the preliminary source of information and that it is possible to generate an interactive application from such information (while adding other ingredients such as UI guidelines for instance). The main claim is that with such an approach it is possible to generate user interfaces for different platforms thus reducing the development costs. The main drawbacks are that it is difficult to integrate design and craft knowledge in such processes. A tool suite based on CTT can help to overcome these drawbacks by integrating reverse engineering tools such as ReverseCTT and ReverseMARIA [4] in order to automatically produce Abstract UI models from concrete UI and to produce task models from Abstract UI models. The TOUCAN approach is different in the way that the software programmer connects the presentation and dialog of the application manually with the task models. The connection between task models and an interactive application at runtime has first been proposed for fully model-based approaches [1]. Such Integrated Development Environments provides support the co-execution between task and system models and enables to produce and to connect task models and formal models of the application at design time and at runtime. This type of IDE can be used for ensuring consistency between task and system models. However, they require using a formal description technique able to encompass all the elements of the interactive application including input device information, interaction techniques as well as the non-interactive part (functional core) of the application. This may lead to high development costs for the construction of the application and interaction models. The TOUCAN approach is different in the way that the software programmer can use a generic development language to develop the interactive software. The TOUCAN IDE is based on previous work on how development processes may support the validation of the effectiveness of an interactive application [5]. The work

presented in [5] focuses on the process. It also discusses the possible different types of approaches for the mapping of task models with interactive applications and presents a proof of concept CASE tool to support this mapping. From this proof of concept version of the tool, we have built a standalone IDE that aims at being publicly released to software developers. This article focuses on this new standalone IDE itself featuring additional capabilities such as the co-execution of JavaFX applications with task models, as well as the programming of adapters for enabling the mapping between task models and custom widget that the developers may have to program.

3 THE DEVELOPMENT OF AN EFFECTIVE INTERACTIVE APPLICATION WITH TOUCAN

The TOUCAN IDE (Integrated Development Environment) enables developers and HCI specialists to code Java Swing or JavaFX applications and to validate their effectiveness using task models as the description of the user tasks. TOUCAN addresses the following challenges: to integrate task modeling common features (edition, simulation) and Java software development main features (programming, build, execution) in an IDE, to provide means to map lines of codes with user actions, to provide means to synchronize and control simulation of task models with java software under execution. To address these challenges, the main features of the TOUCAN IDE are:

- TOUCAN project creation and management (composed of task models, java classes and libraries, as well as correspondence configuration files)
- Java application programming and execution
- Task models editing and simulation
- Automatic extraction of interactive input and output tasks in the HAMSTERS task models
- Automatic extraction of event sources and renderers from annotated applications using Java technology
- Edition of the correspondences between interactive tasks and event sources and renderers
- Co-execution of the Java application with the task models

Figure 1 depicts the screenshot of the TOUCAN IDE while an interactive application is co executing with its associated task model. The TOUCAN IDE is composed of: a project panel (left side panel in Figure 1) presenting the structure of a TOUCAN project, including task models, java software and correspondence files, and a main modeling and programming area for the editing of java applications and task models (main central part in Figure 1).

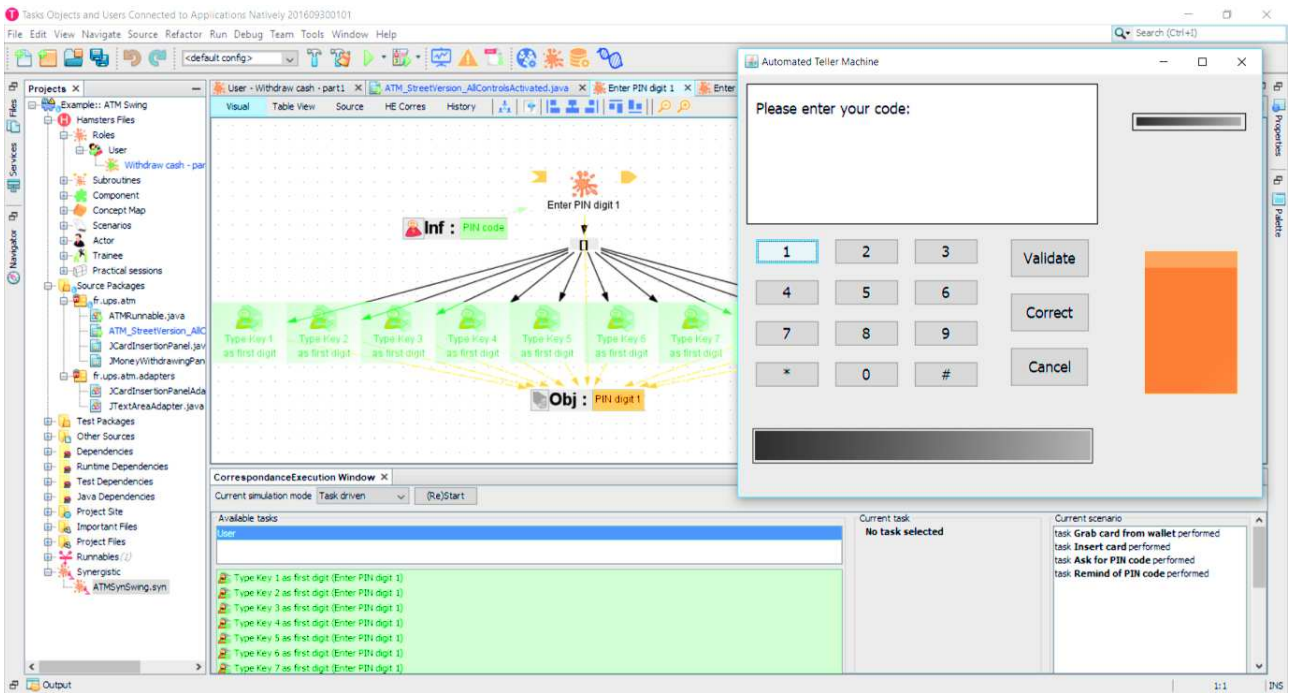


Figure 1. The co-execution of the task model with the application (task driven)

3.1 Illustrative example of the Automated Teller Machine

The ATM is an interactive system that is widely known and used regularly by most of us. This article focuses on the usage of the TOUCAN tool for this simple but realistic example of interactive application.

3.2 Describe user tasks

The TOUCAN tool provides support to edit and simulate

HAMSTERS task models [5]. In the case of the ATM application, the main user goal is to “Withdraw money”. Figure 2 presents the task model that describes the tasks that need to be performed to reach that goal. At the higher abstraction level, the main goal can be decomposed into sub-goals, which can in turn be decomposed into activities. Output of this decomposition is a graphical tree of nodes. Nodes can be tasks or temporal operators. At the second level of the hierarchical decomposition of the goal “Withdraw money” in Figure 2, the following sub-goals have to be reached in a sequential manner: “Identify”,

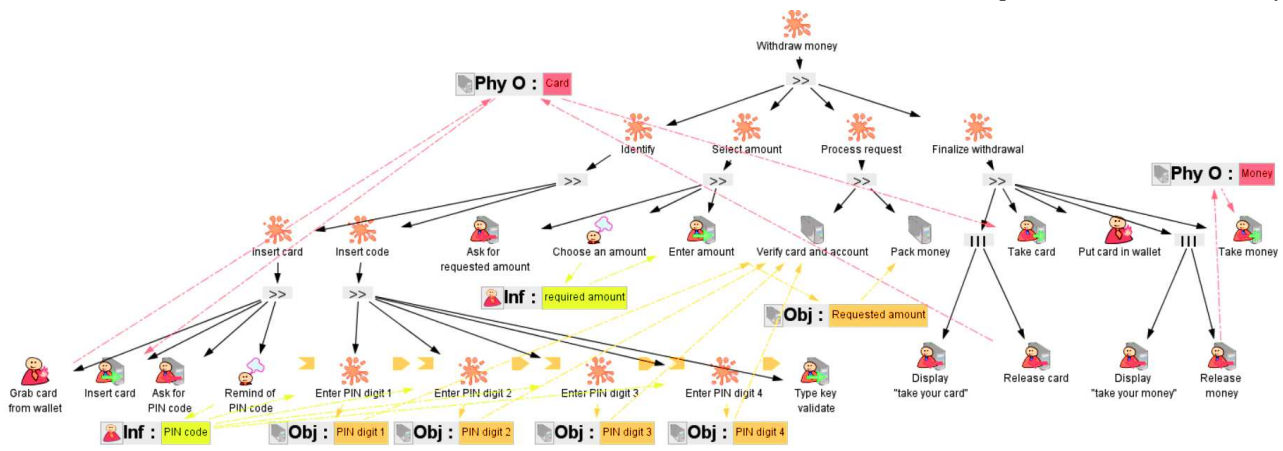


Figure 2. Task model to achieve the goal “Withdraw money”

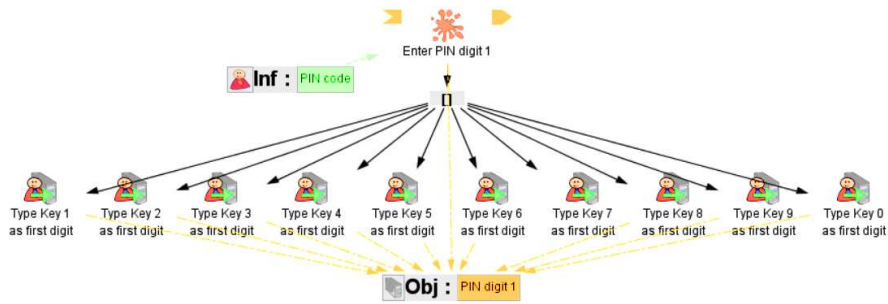


Figure 3. Subroutine “Enter PIN digit 1”

“Select amount”, “Process request”, and “Finalize withdrawal”. At the third level, for sub-goal “Identify”, the following sub-sub-goals have to be performed in a sequential way: “Insert card” and “Insert code”. At the lowest level, of each part of the hierarchical description, user concrete tasks or actions are described. For example, to achieve the sub-sub-goal “Insert card”, the following sequence of user actions has to be performed: “Grab card from wallet”, “Insert card”, “Ask for PIN code”. Tasks may be of several types (user, interactive input, interactive output) and the temporal ordering may be sequential but also concurrent, order independent... For example, in Figure 2, in sub-goal “Finalize withdrawal”, the operator “||” on top of the interactive output tasks “Display take your card” and “Release card” indicates that these two tasks are performed concurrently. Another example is in Figure 3, which depicts the description of the refinement of the task “Enter PIN digit 1” in a subroutine. The operator “[]” indicates that the user has to choose to perform only one of the above described interactive tasks. Data manipulated by the user (information, objects...) are also described in the task model.

3.3 Program the Java application

The TOUCAN IDE is based on the Netbeans Platform [6] and thus provides the same functionalities as the Netbeans IDE from a software programming point of view. User interface Frames may be constructed with the embedded GUI builder. The behavior of the application as well as its functional core may be edited with the Java software editor. Figure 4 shows a screenshot of the UI of the running Java Swing version of the ATM (programmed within the TOUCAN IDE).

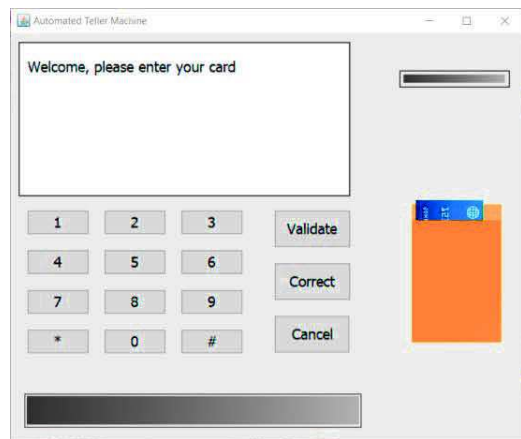


Figure 4. Screenshot of the ATM application

3.4 Annotate Java software

Annotation in Java (since JSE 1.8) is a form of metadata (prefixed with a '@'), providing data about a program that is not part of the program itself. Annotations have no direct effect on the code instructions that they annotate but they are used by the compiler to detect errors. At compilation time, they provide information to generate the code. At run time they provide support for examining the annotated code. The TOUCAN IDE uses this mechanism to allow the programmer to declare:

- event handlers that are related to descriptions of user inputs in task models (interactive input tasks). In that case they use the annotation `@SynergyEventSource` on top of the declaration of the widget variable,
- rendering properties that are related to description of user outputs in task models (interactive output tasks). In that case, they have to use the annotation `@SynergyRenderer` on top of the declaration of the widget variable.

Java Swing	Java FX
<pre>@SynergyEventSource(name = "Key1", event = "actionPerformed") private javax.swing.JButton jButton1;(1) @SynergyRenderer(name = "labelWelcome", property = "text") private javax.swing.JLabel jLabelWelcome;(2) @SynergyEventSource(name = "cardPanel", event = "cardRemoved") @SynergyEventSource(name = "cardPanel", event = "cardInserted") @SynergyRenderer(name = "cardPanel", property = "insertion") private fr.ups.atm.JCardInsertionPanel jCardInsertionPanel;(3)</pre>	<pre>@FXML @SynergyEventSource(name = "buttonOne", event = "actionPerformed") private Button buttonOne;(1) @FXML @SynergyRenderer(name = "welcomeLabel", property = "text") private Label welcomeLabel;(2) @FXML @SynergyEventSource(name = "cardPanel", event = "cardRemoved") @SynergyEventSource(name = "cardPanel", event = "cardInserted") @SynergyRenderer(name = "cardPanel", property = "insertion") private FXMLInsertionPanel insertionPanel;(3)</pre>


Figure 5. Excerpt from the declaration of variables with annotations in Java Swing and JavaFX

Figure 5 presents excerpts from the annotated code of both Java Swing and JavaFX version of the ATM. For each widget, the programmer is able to annotate one event handler (Figure 5, label “(1)”), one rendering property (Figure 5, label “(2)”) or a combination of event handlers and renderings (Figure 5, label “(3)”). Each annotation declared before a variable indicates that the variable may be associated to an interactive task in a task model. In Figure 5, the annotation declared before the variable “jButton1” (Java Swing column, label “(1)”) indicates that the event handler “actionPerformed” of the “jButton1” may be mapped to an interactive input task in a task model. The annotation declared before the variable “jLabelWelcome” (Java Swing column, label “(2)”) indicates that the rendering property “text” may be associated to an interactive output task in a task model. Both types of annotation may be enhanced with extra data to provide information for the correspondence editing. In both cases, it provides a readable name for the correspondence (independent from the attribute name).

3.5 Connect the application to the task model

A dedicated control panel aims at connecting interactive tasks with event handlers and widget rendering properties (depicted in Figure 6). The control panel is composed of three areas. The upper table panel (labelled “Input correspondences” in Figure 6) makes it possible to map interactive input tasks with event handlers (one detailed example of this mapping is described in Table 1).

Table 1. Detailed example of input correspondence

Interactive input task	Event handler
 Type key 1 as first digit	actionPerformed on Key1 (jButton1)

The second table panel (labelled “Output correspondences in Figure 6) makes it possible to map interactive output tasks with rendering properties (one detailed example of this mapping is described in Table 2). The lower part of

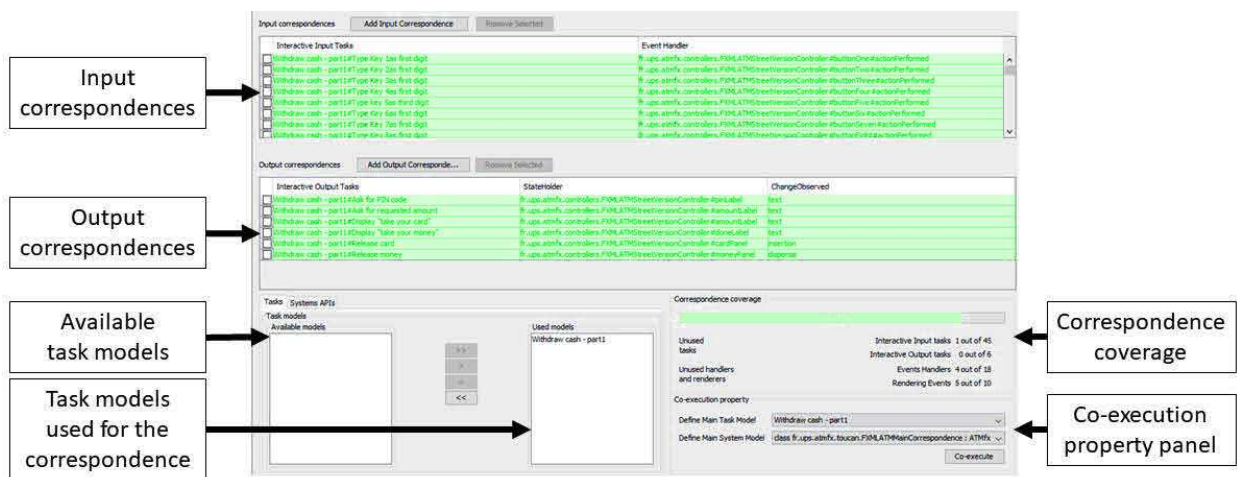



Figure 6. The correspondence edition window in TOUCAN IDE

the control panel is dedicated to the analysis of the correspondence coverage. This correspondence coverage area (labelled “Correspondence coverage” in Figure 6) allows programmers to check the completeness of the correspondences/mapping that have been performed. It highlights the ratio of interactive tasks that are not connected to event handlers and rendering properties. The programmer can then check whether there are missing widgets and/or functionalities in the interactive application or whether the task descriptions has to be modified if there are missing or irrelevant user actions.

Table 2. Detailed example of output correspondence

Interactive output task	Change in the rendering property of a widget
 Ask for PIN code	Property: text of labelEnterAmount (jLabelAmount)

For example, in Figure 6, the correspondence coverage panel displays that the coverage is not full (orange bar filled in up to around 50%). In addition, it displays the ratio of interactive tasks that are not mapped (meaning that a widget is maybe missing) and the ratio of event handlers and property renderers that are not mapped (meaning that some task description are maybe missing or that widgets or functions are maybe useless for the user). This enables to alert the programmer that the mapping is not up-to-date, and that the effectiveness criteria cannot be guarantee to be achieved.

3.6 ° Co-execute the application with task model

Once the correspondence between interactive tasks and event sources and renderers is completed, the button labeled “Co-execute” aims at triggering the co-execution of the interactive application with the task models. Figure 1 depicts the co-execution of the Java application with its associated task model in the TOUCAN IDE. The lower part in Figure 1 depicts the co-execution control panel. The TOUCAN IDE provides three different means for the co-execution of the interactive application and its associated task models: task model driven co-execution, system driven co-execution and scenario driven co-execution. In Figure 1, the co-execution is task model driven. Tasks highlighted in green in the task model as well as in the list are available for execution. Clicking on an interactive input task in the list triggers its execution in the task model and the execution of the corresponding event handler in the interactive application. In the case where an object is produced by the application once the user has performed an interactive task (for example object “requested amount” in the task model presented in Figure 2), the task driven co-execution panel displays an input area for editing the value of this object before triggering

the execution of the task. The interactive application is run through the execution of the task model. The scenario-driven co-execution is quite similar but the programmer triggers the execution of the tasks that are listed in the selected scenario, even if other tasks would have been available. In the case of the system driven co-execution, the programmer interacts with the interactive application and the interactive tasks in the task model are highlighted in light blue if a correspondence is detected. In addition to the blue highlighting, the ordering of the interaction performed while using the application is displayed close to the task.

The task driven co-execution provides support for detecting inconsistencies between the planned tasks and the behavior of the application. For example, if a widget is not enabled in the application and that, according to the execution of the task model, the corresponding interactive input task should be executable, the task is highlighted in orange to show the inconsistency between the specified task and the behavior of the application. In the same way, the scenario driven co-execution provides support for detecting inconsistencies between the specified scenarios and the interactive application. The system driven co-execution can be used at user testing time to track the frequency of occurrence of the interactive input tasks.

4 IMPLEMENTATION OF THE TOUCAN IDE

The TOUCAN IDE is built on top of the Netbeans Platform [6] and encompasses the basic IDE functions such as project management, unit testing, versioning... The TOUCAN IDE integrates several software elements: HAMSTERS for the editing and execution of task models, the Netbeans API for the editing and execution of Java code, and an in home library named Synergistic for the parsing of the annotations as well as for the correspondence edition and execution. The TOUCAN IDE integrates natively a library of classes of adapters for Java Swing and JavaFX widgets. Programmers may develop their own custom widget. In that case, an adapter is required to be programmed for the custom widget. The main attributes of the adapter are: events and properties to be handled, a list of listeners that need to be called once an event occurs. The main methods of the adapter are: a method to process the events, a getter for the instance of the adapter and a getter for the class type of the widget for which the adapter has been programmed. This feature enables programmers to annotate a custom widget and to map its event handlers and property renderers to interactive tasks in task model. In the illustrative example of the ATM, an adapter has been coded for the custom widget “JCardSlotPanel,” which is the panel containing the virtual card and the card slot. The TOUCAN IDE can also be used in the case of large scale interactive applications. A TOUCAN project may contain several java

classes and several task models. The mapping can be done between several tasks in several task models and several handlers and renderers in several Java classes.

5 CONCLUSION

This article highlights the principles and the functioning of TOUCAN IDE blending tasks models and Java code of interactive application. We use the well-known example of a simulated cash machine. TOUCAN provides a precise and concrete way for integrating human factor work (and more precisely tasks analysis and tasks description) within software development. It proposes a tool-support bridge over the gap between design and development. The bridge consists in making explicit the correspondence between common elements in the tasks descriptions and in the user interface. Reaching this goal requires limited additional work on the developer side (adding annotations) while providing significant help for evaluating the usability (and more precisely the effectiveness) of interactive applications. Such IDE could be helpful for increasing the use of task models in interactive system development by providing benefits even for existing and already deployed applications.

REFERENCES

- [1] Barboni E., Ladry J-F., Navarre D., Palanque P. and Winckler M. Beyond modeling: an integrated environment supporting co-execution of tasks and systems models. EICS'10, 165-174.
- [2] Calvary G., Coutaz J., Thevenin D., Limbourg Q., Bouillon L., Vanderdonck J. A Unifying Reference Framework for multi-target user interfaces. *Interacting with Computers* 15(3): 289-308 (2003)
- [3] ISO 9241-11: Ergonomic requirements for office work with visual display terminals (VDTs) - Part 11: Guidance on usability (1998).
- [4] Manca, M., Paternò, F., Santoro, C. A Public Tool Suite for Modelling Interactive Applications. *Handbook of Formal Methods in Human-Computer Interaction 2017*: 505-528
- [5] Martinie, C., Navarre, D., Palanque, P., Fayollas, C. A generic tool-supported framework for coupling task models and interactive applications. In *Proc. of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS'15)*. ACM, New York, NY, USA, 244-253.
- [6] Netbeans Platform, <https://netbeans.org/features/platform/index.html>, last accessed March 2018.
- [7] Palanque P., Martinie C. 2015. Designing and Assessing Interactive Systems Using Task Models. *ACM CHI Extended Abstracts*, 2465-2466.
- [8] Palanque P., Martinie C. Designing and Assessing Interactive Systems Using Task Models. 2016. *ACM CHI Extended Abstracts*, 976-979.
- [9] Wilson S., Johnson P., Kelly C., Cunningham J. and Markopoulos P. Beyond hacking: A model based approach to user interface design. In *Proceedings of HCI'93*, 217–23, University Press, BCS HCI.