SUIDT: Safe User Interface Design Tool

Mickaël Baron, Patrick Girard
LISI/ENSMA
1 rue Clément Ader
86961 Chasseneuil Futuroscope Cédex, France
+33 5 49 49 80 70
{baron,girard}@ensma.fr

ABSTRACT
SUIDT (Safe User Interface Design Tool) is a model-based system that allows building interactive systems with respect to the formal semantics of functional cores. It implements a complete cooperation between task models (abstract and concrete) with both the domain model and the presentation model, while ensuring the properties of the models. Last, it maintains all during the design cycle the links between every part of the system, even the functions of the functional core (the actual code).

Categories & Subject Descriptors: D.2.2 [Software Engineering] : Design Tools and Techniques – *computer-aided software engineering, top-down programming, user interfaces.*

General Terms: Design, Human Factors, Verification.

Keywords: Model-Based Systems, Task Models.

INTRODUCTION

HCI tools for building interactive software are numerous. On the one hand, GUI builders and tools from suites such as Visual Basic® or JBuilder® do not provide any way to handle any kind of external model.

On the other hand, Model-Based tools [3, 4] deal with models, but are currently not usable for actual software development. Often, they cover the design phase of interactive systems, and they allow the generation of the system itself. But the semantic link between model and code is generally cut during this generation.

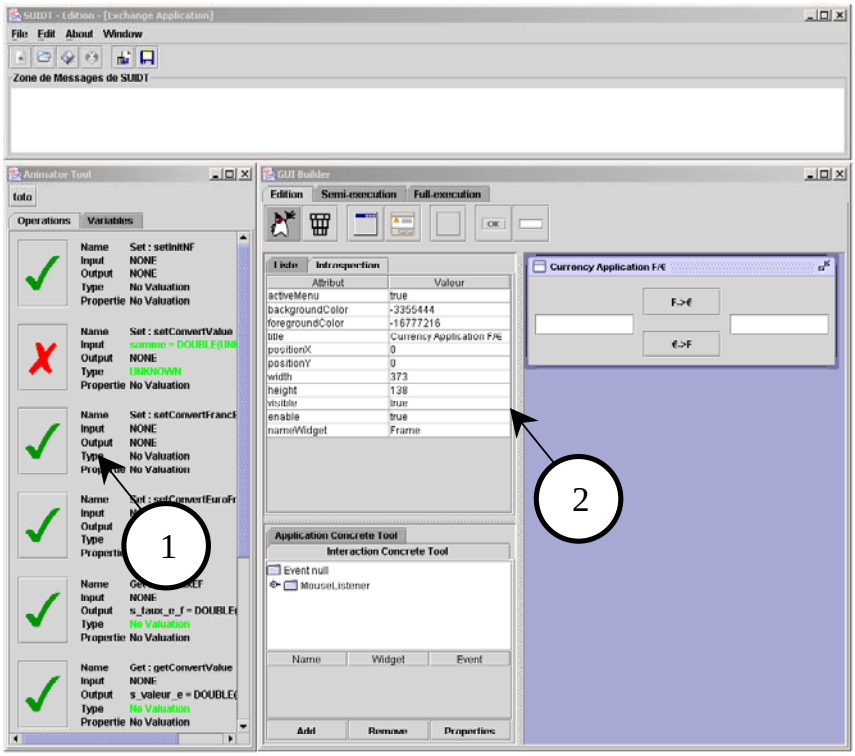
Our goal in this paper is to present a system that preserve this link between models and code, and that permits to work together with any model while testing the application. We believe that this approach may be a solution to improve the usability of model-based systems.

IUI'04, Jan. 13-16, 2004, Madeira, Funchal, Portugal.
ACM 1-58113-815-6/04/0001.

FIRST STEP: THE SYSTEM SIDE

Our first step was to demonstrate how it is possible to build a tool that ensures a semantic formal link. We start from a formally developed functional core. We assume that this functional core, which has been specified with the B method [1], delivers services through an API. It is possible to automatically link such a functional core to a tool that exploits function signatures and formal specifications to help building interactive software.

In figure 1, we can see a screen copy of some SUIDT elements. On the left (tag 1), the animator consists in a fully generated interface that allows to interactively run the functional core. Every function of the functional core is usable through button activation. When parameters are



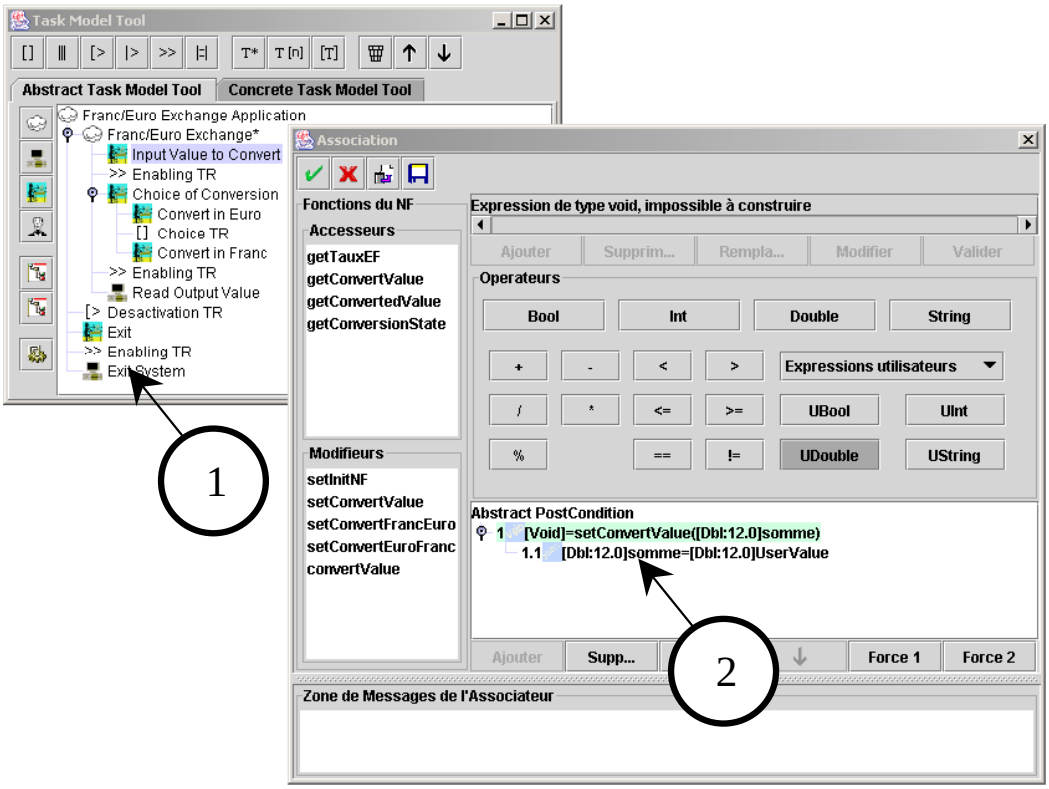
required, a dialog box appears to allow the user to enter them. Functions are textually described, and current state of the functional core can be estimated through the result of all functions. It is important to notice that all that part is fully automatically generated. It allows the user to "play"

with his/her functional core, and to be aware of functional core state.

SECOND STEP: THE USER SIDE

The second step of our study consists in focusing on task-based analysis. We incorporated task-based analysis into our system by the way of two task models (abstract and concrete task models) using the Concurrent Task Tree (CTT) formalism [2].

In figure 2, we see on the upper left (tag 1) a view of the abstract task model. While CTTE provides a purely graphical view of CTT trees, we chose to draw them in a tree widget. This avoids many problems like sizing or beautifying. The original part of the study consists in the link that exists between the abstract task model and the functional core. In tools such as CTTE, we can animate the task model, in order to ensure that the specifications of the system are consistent. In SUIDT, we can go one step further. We can animate the system itself; we exploit the possibility to interactively run the functions of the functional core, with respect to the formal specifications of this one. More, we can also link the pre- and post-conditions of CTT to functions, in order to dynamically control the execution. This is shown in the front window on figure 2 (tag 2).



The domain model plays a central role in the SUIDT approach. Every model is in fact related to the it, which allows to run an actual program each time we execute a model. This property gives to SUIDT a very high degree of interactivity. More, while links are always active, the user never loose the benefits of the modeling and can modify either part of the application with complete respect to model properties.

In order to reach the concrete level of interaction, it is necessary to add a level. We named this level the Concrete Task Model (CTM). It consists in a refinement stage of the Abstract Task Model (ATM), where every interactive or application task of the ATM is described in terms of concrete interaction or application objects. Because we are in the same whole environment, it is possible now to rely these concrete abstract objects to the concrete GUI level.

CONCLUSION

We can summarize the above presentation of SUIDT relating to the links between models. Every model in SUIDT is an executable model. Models can be executed for inspection, for simulation, or for verification.

REFERENCES

1. Abrial, J.-R. (1996). The B Book: Assigning Programs to Meanings. Cambridge University Press.
2. Paternò, F. (2001). Model-Based Design and Evaluation of Interactive Applications. Springer.
3. Puerta, A. (1996). The MECANO project: comprehensive and integrated support for Model-Based Interface development, Proceedings of the Computer-Aided Design of User interface (CADUI'96), 19-35. Presse Universitaire de Namur.
4. Szekely, P. (1996). Retrospective and challenge for Model Based Interface Development. In F. Bodart & J. Vanderdonckt (Eds.), Eurographics Workshop on Design, Specification, and Verification of Interactive Systems (DSV-IS'96) (pp. 1-27). Namur, Belgium: Springer-Verla