Abstract. This chapter claims that task models per se do not contain sufficient and necessary information to permit automatic generation of interactive systems. Beyond this, we claim that they must not contain sufficient and necessary information otherwise they could no longer be considered as task models. On the contrary we propose a way of exploiting in a synergistic way task models with other models to be built during the development process. This chapter presents a set of tools supporting the development of interactive systems using two different notations. One of these notations called ConcurTaskTree (CTT) is used for task modeling. The other notation called Interactive Cooperative Objects (ICO) is used for system modeling. Even though these two kinds of models represent two different views of the same world (a user interacting with an interactive system), they are built by different people (human factors specialist for the task models and software engineer for the system models) and are used independently. The aim of this chapter is to propose the use of scenarios as a bridge between these two views. On the task modeling side, scenarios are seen as a possible trace of user’s activity. On the system side, scenarios are seen as a trace of
user’s actions. This generic approach is presented on a case study in the domain of Air Traffic Control. As both CTT and ICO notations are tool supported (environments are respectively CTTE and PetShop) an integration tool based on this notion of scenarios is presented. Its use on the selected case study is also presented in detail.

17.1 INTRODUCTION

In the Human-Computer Interaction domain (HCI), when dealing with interactive systems it is widely agreed upon that user information has to be taken into account and that this must be done through task analysis and task modeling. During the design process of interactive systems, user goals have to be analyzed as part of the specification phase, while task analysis is conducted during the design phase.

Classically formal notations are meant to guarantee certain quality of the models. For instance, they are used to ensure completeness or non-ambiguity of the descriptions. We try to demonstrate in the following that they can also be used to ensure consistency among the various models that are build in the various phases of the development process of an interactive system. As models built using formal notations are non-ambiguous, they can be analyzed automatically by inspection tools.

Even though formal notations are used for task modeling, it is not possible to generate, from the task models, the interactive application supporting those tasks. The justification of this claim comes from the following argument we develop hereafter: “it is impossible because there is not enough information in the task models to generate code.” It is possible to extend the task models with additional information, but in that case we claim that the resulting model is not anymore a task model but a merging of both a task model and a model of the behavior of the system. Another alternative is to use generic information about interactive application and to use them in order to generate the code of the final interactive system. In that case we claim that the application generated is very stereotyped and that this is only possible for a very small number of applications.

We take as an example the Trident project (Bodart et al., 1994) that was dedicated to the generation of form-based interactive applications from task models and data models. Even for this kind of “simple” interactive application a data model was mandatory and was actually the core of the generation process (the task model was only used for the dialogue part of the application and the structure of the various windows of the application). Instead of generating the code from a task model, we propose to use the task model as a means for checking that the system model is compliant with it (Palanque and Bastide, 1997).

In the chapter, after a short discussion of related works, we recall the basic concepts of the approaches and tools that we aim to integrate. Then, we discuss the architecture of the solution identified. An example of application of the integrated set of tools is discussed before drawing some concluding remarks. The case study presented is extracted from the European Project MEFISTO which is a long-term research project dedicated to the definition and use of formal method for the design of safety-critical interactive systems. In particular, the project has focused on the air traffic control application domain from which this case study has been drawn.