Simulating Task Models Using Concrete User Interface Components

David Paquette
Department of Computer Science
University of Saskatchewan
dnp972@cs.usask.ca

April 29, 2004

Abstract

Interaction Templates were previously introduced as a method to help ease the construction of ConcurTaskTrees. This paper begins with a brief introduction to task modelling, ConcurTaskTrees, and Interaction Templates. Interaction Templates are then discussed in more detail in terms of a language for defining Interaction Templates. An overview of ConcurTaskTrees simulators is given, and the details of how Interaction Templates can be simulated using concrete interface components are also discussed. A task model simulator that shows how a concrete interface component can simulate a task model was built. Finally, the paper concludes with an outline of possible future work with Interaction Templates.

1 Introduction

Task models are formal models that describe interactive systems in terms of goals and user activities [9]. Activities can be either logical or physical. Take for example a multimedia software package. Retrieving a CD's play list would be a logical activity, whereas Selecting the play button would be a physical activity. A goal can either be a desired change in the state of a system or an attempt to retrieve some information from a system. Retrieving a CD's play list would be the retrieval of information from an application, whereas Changing the track would be a modification of the state of an application.

Task models can be used for many different purposes such as:

- helping to better understand an application domain;
- serving as a common language for all stakeholders, including system developers, interface designers, managers, users, and domain experts;
- helping to ensure that systems built match the user's conceptual model;
- analyzing and evaluating the usability of interactive systems, often before the system is ever built;
- supporting the user through a task-oriented help system;
- documentation of interactive systems. [9]

ConcurTaskTrees (CTT) [9, 8], introduced by Fabio Paternó, is the latest approach to task modeling that has been popular in several research communities. "[CTT] is a notation aiming at supporting engineering approaches to task modeling" [9]. CTT is a graphical notation with a rich set of operators for describing the relationships between tasks. An overview of the CTT notation can be seen in Figure 2, while Figure 1 shows a simple graph editor modeled using ConcurTaskTrees.

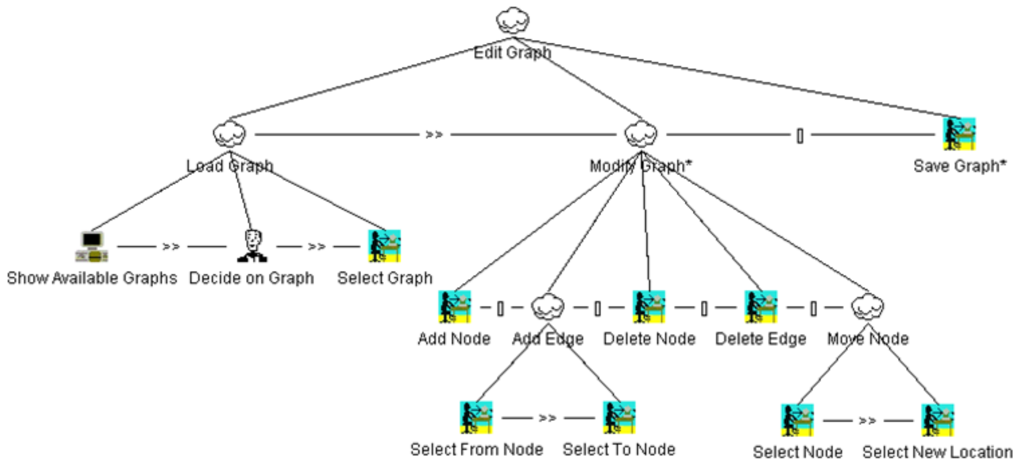
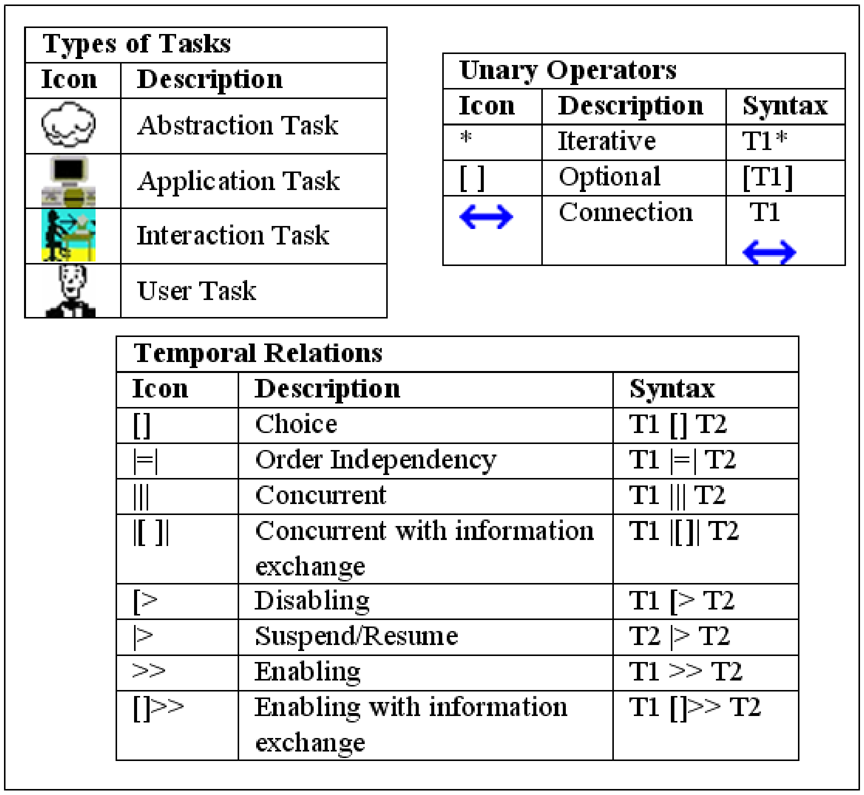


Figure 1: A ConcurTaskTree for a simple graph editing program.

The main features of ConcurTaskTrees are:

- a focus on activities that helps designers avoid low level details that would obscure the designer at the design state;
- an intuitive hierarchical structure;
- a graphical syntax, in tree-like form, which is often easier to understand;

- a concurrent notation through a rich set of temporal relationships among tasks;
- task allocation with four different types of tasks;
- and objects, both user interface objects and application domain objects, that have to be manipulated can be indicated in the task model [9].



Types of Tasks

Icon	Description
	Abstraction Task
	Application Task
	Interaction Task
	User Task

Unary Operators

Icon	Description	Syntax
*	Iterative	T1*
[ ]	Optional	[T1]
↔	Connection	T1 ↔

Temporal Relations

Icon	Description	Syntax
[]	Choice	T1 [] T2
|=|	Order Independency	T1 |=| T2
|||	Concurrent	T1 ||| T2
|[ ]|	Concurrent with information exchange	T1 |[]| T2
[>	Disabling	T1 [> T2
|>	Suspend/Resume	T2 |> T2
>>	Enabling	T1 >> T2
[]>>	Enabling with information exchange	T1 []>> T2

Figure 2: Overview of the CTT notation.

Despite the advantages of CTT, large interactive systems described using the CTT notation can become too large to be easily understood and can be very tedious to build [6]. The remainder of this paper will discuss how Interaction Templates can be used to ease the task modeling process, as well as enhance task model simulation.

2 Interaction Templates

Interaction Templates are intended to help developers build task models quickly, and allow for detailed simulation while maintaining a useful system overview [6].

While building task models for information systems, there are subtrees that repeat throughout the model with only slight variations. These subtrees are often associated with common interface interactions found in information systems. Interaction Templates model these common interface interactions. They include a detailed task model, an execution path (i.e. dialog), and a presentation component. An Interaction Template is a parameterized subtree that can be inserted into a ConcurTaskTree at anytime. Inserting and customizing Interaction Templates reduces the need to model the same interaction repeatedly in a system, and thus, can greatly reduce the time spent modeling information systems. As well, Interaction Templates can be designed and tested to ensure their usability in accomplishing a task. Interaction Templates can also be designed to be 'plastic' and thus adapt to different contexts.

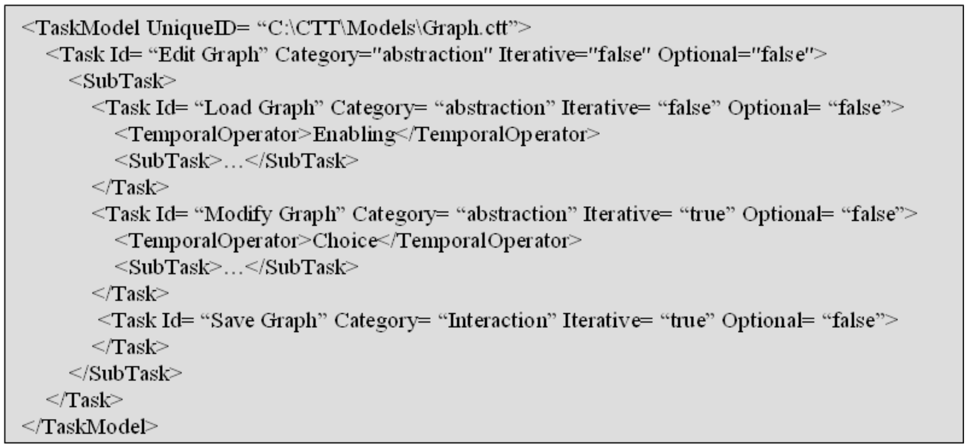
2.1 Defining Interaction Templates

Interaction Templates are described using a custom markup language embedded inside an XML description of a ConcurTaskTree. An example XML description of a ConcurTaskTree is shown in Figure 3.

The root task of an Interaction Template is surrounded by an identifying *it:template* tag that contains a single name attribute specifying the name of the template. The first element found inside the *it:template* element is the empty *it:options* element. The *it:options* element contains attributes specifying all the options for the current template. The name of the attribute identifies the name of the option, while the value of the attribute identifies the option's type. Option types include boolean values, numbers, strings, or file paths to XML documents such as schemas or sample data.

The options specified in the *it:options* element are referenced inside the template using Interaction Template commands. Interaction Template commands are used to specify how the template's task tree is built based on the options specified for the template. Interaction Templates are defined using two basic commands: *it:case* and *it:foreach*.

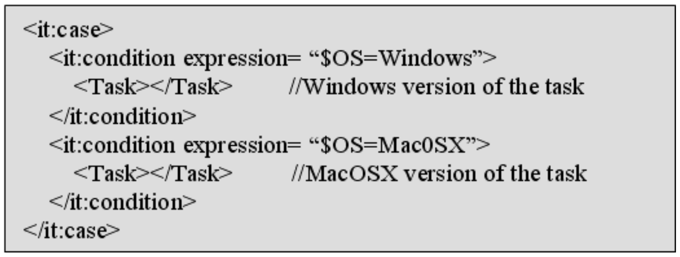
The *it:case* statement is used to select a specific task or subtask based on the options set for the template. An *it:case* statement contains one or more *it:condition*



```
<TaskModel UniqueID= “C:\CTT\Models\Graph.ctt”>
  <Task Id= “Edit Graph” Category="abstraction" Iterative="false" Optional="false">
    <SubTask>
      <Task Id= “Load Graph” Category= “abstraction” Iterative= “false” Optional= “false”>
        <TemporalOperator>Enabling</TemporalOperator>
        <SubTask>…</SubTask>
      </Task>
      <Task Id= “Modify Graph” Category= “abstraction” Iterative= “true” Optional= “false”>
        <TemporalOperator>Choice</TemporalOperator>
        <SubTask>…</SubTask>
      </Task>
       <Task Id= “Save Graph” Category= “Interaction” Iterative= “true” Optional= “false”>
      </Task>
    </SubTask>
  </Task>
</TaskModel>
```

Figure 3: Partial XML description of the task model shown in Figure 1

statements, and will select the first *it:condition* whose expression attribute evaluates to true. The expression attribute found in the *it:condition* element can contain any boolean expression made up of =, >, <, >=, <=, &&, and ||. The *it:case* statement is the statement that allows templates to be designed to be plastic based on the values given to a template’s options. The *it:case* statement can appear anywhere inside a template definition and can also be nested, allowing a template to be plastic at any level in the task tree. An example of an *it:case* statement with conditions to select between two operating systems is shown in Figure 4.



```
<it:case>
  <it:condition expression= “$OS=Windows”>
    <Task></Task>        //Windows version of the task
  </it:condition>
  <it:condition expression= “$OS=Mac0SX”>
    <Task></Task>        //MacOSX version of the task
  </it:condition>
</it:case>
```

Figure 4: it:case and it:condition statements

The *it:foreach* statement is used to repeat a task or subtask for each element in a specified list of elements. An example of a list of elements is all of the elements contained in a complexType of an XML schema. In Figure 5, the *it:foreach* statement is used to repeat a task for each of the elements contained in the complexType found in a schema file specified as one of the template’s options. Inside the *it:foreach* statement, the current element is referenced by the name of the single attribute of the *it:foreach* statement. The current element's name attribute is referenced by adding '.name' to the element reference.

```
<it:foreach col= "$schemaFile.complexType.element">
   <Task Identifier= "Sort By $col.name"></Task>
</it:foreach>
```

Figure 5: it:foreach statement

When an Interaction Template is inserted into a ConcurTaskTree, and the required options have been set, the tree is expanded according to the *it:case* and *it:foreach* statements. References to options and the *it:foreach* attribute, identified by '$optionName or $attributeName', are replaced by the option or attribute's value respectively.

A prototype of an Interaction Template Definition Language interpreter that recognizes and expands *it:foreach* statements has been implemented using TXL, a rule-based tree transformation language [2]. The prototype is fairly simple, consisting of only 5 rules implemented in just over 100 lines of TXL code.

2.2 Using Interaction Templates

After an Interaction Template has been defined using the Interaction Template Definition Language described above, using an Interaction Template is simply a matter of inserting the template into a task model and setting values for the options of that Interaction Template. Once the options have been set, the Interaction Template is expanded using an Interaction Template Definition Language interpreter. After the template has been expanded, it is always possible to edit the expanded task model to customize the template to a specific use. It is also possible to change the options for a template and have the Interaction Template expand again to reflect those changes.

Currently, no tool support exists for building task models using Interaction Templates. Interaction Templates must be inserted by hand into XML descriptions of ConcurTaskTrees that are saved from CTTE.

3 Simulating Task Models

One of the advantages to using a formal modelling language such as ConcurTaskTrees is the ability to simulate the system before it is built. Simulation can help to ensure the system that is built will match the user's conceptual model as well as help to evaluate the usability of a system at a very early stage. Several task model simulators have been built for ConcurTaskTrees. First, this section will discuss how ConcurTaskTrees can be simulated. Next, an overview of some of the task model simulators that are available will be given. The section will conclude with a description of the Enhanced Task Model Simulator that was built to show how Interaction Templates can be simulated using concrete user interface components.

3.1 The Simulation Process

Simulating a ConcurTaskTree involves simulating, in some way, the performance of specific tasks in order to reach a pre-defined goal. In a ConcurTaskTree, tasks are related to each other according to the temporal relations and hierarchical breakdown of the tasks. That is, depending on what tasks have been performed, some tasks are enabled and others are disabled. The first step in simulating ConcurTaskTrees is to identify the sets of tasks that are logically enabled at the same time. A set of tasks that are logically enabled at the same point in time is called an enabled task set (ETS) [8]. Enabled tasks sets are identified according to the rules laid out in [8]. The set of all enabled task sets for a specific task model is referred to as an enabled task collection (ETC).

Having identified the ETC for a task model, the next step is to identify the effects of performing each task in each ETS. The result of this analysis is a state transition network (STN), where each ETS is a state and transitions occur when tasks are performed. The final preparation step for simulation is to calculate the initial state. A command-line tool called TaskLib [4] can be used to extract the ETC, STN, and initial state from a CTT. The details of TaskLib's implementation can be found in [5].

Once the ETC, STN and initial state have all be identified, simulation can begin. This initial process is common to all ConcurTaskTree simulators. The actual simulation involves the user navigating through the STN by simulating the performance of tasks in some way. As will be discussed shortly, the simulation of performing a task is done differently in the simulation tools that exist.

3.2 Simulation Tools

Basic Simulators: The most basic simulators, such as the one shown in Figure 6, simply display the currently enabled tasks in a list. In these simple simulators, double-clicking on a task will simulate the performance of that task. When a task is performed, the enabled tasks are updated accordingly. A basic task model simulator can be found in ConcurTaskTreesEnvironment (CTTE) [7], a tool for both building and simulating task models, as well as the Enhanced Task Model Simulator described below.

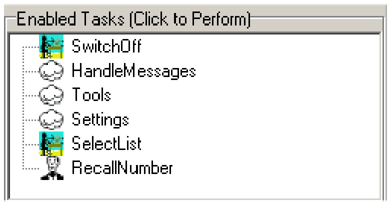


Figure 6: A simple ConcurTaskTree task model simulator.

Dialogue Graph Editor: The Dialogue Graph Editor, a tool developed at the University of Rostock, provides a slightly more complex simulation than the basic simulator found in CTTE. The Dialogue Graph Editor allows designers to create views and assign tasks from a task model to those views. These views can later be used to simulate the task model as shown in Figure 7. When simulating the task model, views are represented as windows, elements (as well as tasks) inside the windows are represented by buttons, and transitions are represented by navigation between windows [3]. Views become visible when they are enabled, and invisible when they are disabled. Likewise, buttons become enabled and disabled when their associated tasks are enabled or disabled. Users can simulate the task model by clicking buttons to perform tasks and navigate through windows to select between available tasks.

The windows and buttons generated by Dialogue Graph Editor for simulation purposes are considered to be abstract interface prototypes. However, clicking buttons to perform tasks does not seem to provide much of an advantage over the basic simulators, and at times might be more confusing. The key advantage in Dialogue Graph Editor is the ability to organize tasks into a dialog. Unfortunately, this requires an additional dialog model as well as a mapping between the dialog model and task model.

Enhanced Task Model Simulator: While Interaction Templates model common interface interactions found in information systems, there are often concrete user interface components that implement those interactions. In interface builders such as Borland's Delphi, interfaces are constructed using pre-built components. If an Interaction Template models a common interface interaction and there exists an interface component that implements that common interface interaction, then that

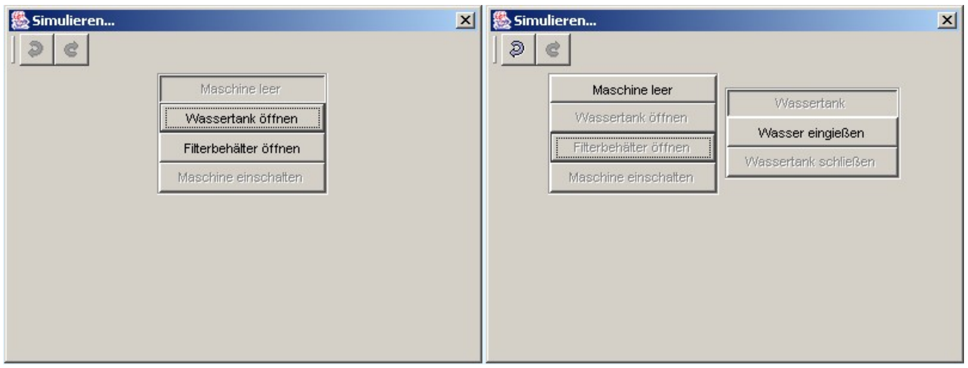


Figure 7: Simulator included in Dialogue Graph Editor.

interface component can be used to simulate the Interaction Template. For example, the Data Table Interaction Template can be simulated using a data table interface component that is included with Delphi.

The Enhanced Task Model Simulator (ETMS), shown in Figure 8, was built to show how concrete user interface components can be used to simulate the sections of a task model where Interaction Templates are used. The ETMS was built using Borland Delphi 6, and contains a traditional task model simulator based on the Enabled Task Sets and State Transition Networks derived using TaskLib [4]. The ETMS contains three views: the model view, the simulator view, and the prototype view. The model view shows a simple tree view of the entire task model. The simulator view, titled 'Task Model Simulator', contains a basic task model simulator as well as a list displaying the current activity chain. In the simulator view, tasks can be performed by double-clicking on them. When a task is performed, it is added to the bottom of the activity chain [5]. The activity chain shows a history of the interactions that have occurred in a simulation session. The prototype view shows the currently enabled Interaction Template prototypes. The Interaction Template prototypes allow the user to interact with a concrete user interface component to simulate a portion of a task model. When the tasks from an Interaction Template become enabled in a simulation session, a prototype consisting of a concrete interface component corresponding to that Interaction Template is shown in the prototype view. When those tasks are disabled, the prototype is hidden. In the current implementation of the ETMS, creation, enabling, and disabling of prototype instances are done manually.

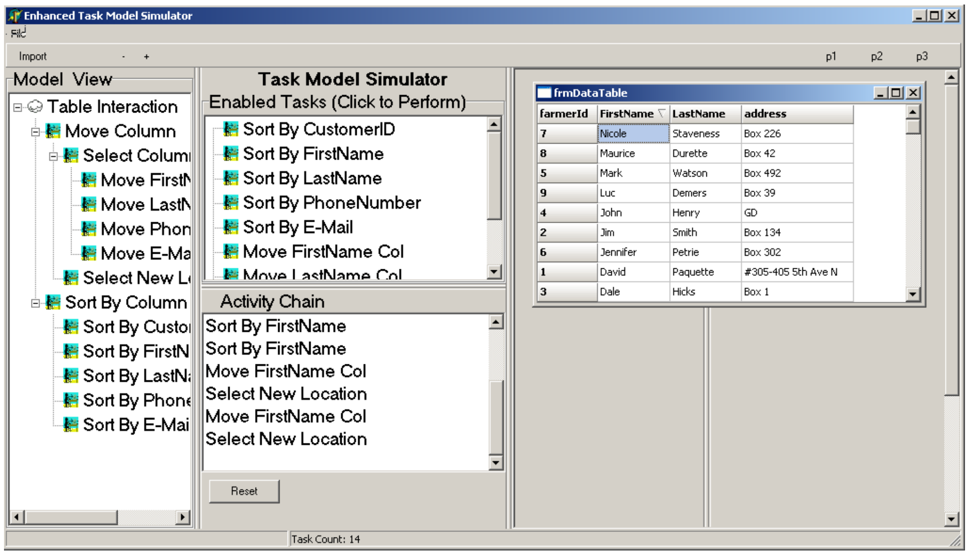
Interaction Template prototypes are manually built once, then created and customized dynamically during simulation sessions. A new Delphi form containing the appropriate interface component is created for each type of Interaction Template. Each new prototype inherits from the generic *TfrmPrototype* object, which contains the functionality that is common with all Interaction Template prototypes. Functionality that is in common with all prototypes includes the ability to communicate with the simulator, was will as the ability to show and hide itself when told by the simulator.

Each specific prototype will implement its own adaptation logic. When a prototype object is created, it will read-in the *it:options* tag that contains the options for the current use of the Interaction Template. The prototype object will adapt itself to the options specified in the *it:options* tag. With the Data Table Interaction Template for example, the data table prototype will read in the schema file to set the column headers and read in the sample data to fill in the rows. Most other Interaction Template options have a one-to-one mapping to the attributes for the interface component that is used to simulate the Interaction Template. For example, the Data Table Interaction Template contains a boolean option called ‘allowsort’, which has a direct mapping to the boolean ‘showsort’ attribute of the data table component used in its prototype. Adaptation logic for those options is simply a matter setting the attributes of the interface component. Finally, each specific prototype will implemented a mapping between events and task occurrences in the task model. Since communication between the prototype and the simulator is already implemented, this is simply a matter of defining the name of the task that is performed when an event is triggered.

While all other task model simulators use abstract interface objects to simulate tasks, concrete user interface components can be used to simulate Interaction Templates that have been inserted into ConcurTaskTrees. Using the Enhanced Task Model Simulator, users can interact with concrete interfaces to simulate portions of a larger task model. The Interaction Template prototypes can also be populated with sample data, making the simulation less abstract and easier for users to understand.

4 Open Problems

This paper has further explored Interaction Templates as a tool to help in building and simulating Task Models using ConcurTaskTrees. A language for defining Interaction Templates has been proposed. The Enhanced Task Model Simulator has shown how concrete user interface components can be used to create Interaction Template prototypes, and ultimately enhance the task model simulation process.



Sort By FirstName
Sort By FirstName
Move FirstName Col
Select New Location
Move FirstName Col
Select New Location

farmerId	FirstName	LastName	address
7	Nicole	Staveness	Box 226
8	Maurice	Durette	Box 42
5	Mark	Watson	Box 492
9	Luc	Demers	Box 39
4	John	Henry	GD
2	Jim	Smith	Box 134
6	Jennifer	Petrie	Box 302
1	David	Paquette	#305-405 5th Ave N
3	Dale	Hicks	Box 1

Figure 8: A prototype of the Enhanced Task Model Simulator.

While several concepts have been introduced here, there are still many questions that need answering. The goal of Interaction Templates is to aid in building task models while providing enhanced simulation using concrete interface components, and remaining plastic with respect to operating systems and environments. This paper will concluded with an outline of some areas that need to be further investigated in order to reach this goal.

Tool Support: Tool support is needed both for building and defining Interaction Templates and for building task models using Interaction Templates. A tool for building task models using Interaction Templates must include an interpreter for the Interaction Template Definition Language described earlier. Such a tool would interpret an Interaction Template and expand it based on the values of the options that are set for the current use of the template.

Plasticity: Do the *it:case* and *it:condition* statements provide enough flexibility achieve plasticity? Some concrete examples of common interface interactions that differ when performed in different contexts are needed. Using concrete examples would allow the Interaction Template Definition Language to be tested to see if Interaction Templates can be designed to implement the examples.

Generating Prototypes: Currently, prototypes are manually built to be self adaptive to the options set for an Interaction Template. Linking events to specific tasks is also manually coded when the prototype is initially created. The manual coding is only done once, and since the adaptation logic is built in, a prototype can be reused a number of times to simulate a template. Ideally, prototypes would be automatically generated from Interaction Templates. Unfortunately, there is no obvious solution to how data can be automatically loaded into interface components, nor is there an obvious way to automatically decide on a mapping between event occurrences and tasks in the task model. It is likely that the mapping between events and tasks will always need to be manually defined once. Also, unless all interface components begin to comply to a common interface for loading data, manual code will need to be written to load data into components as well as to set component attributes based on options set for an Interaction Template. In the current implementation, event-to-task mapping and adaptation logic must be manually coded once for each interface component. The amount of code needed to implement these two requirements is minimal, making the current solution a viable option.

Enhanced Simulation: The current ETMS prototype allows an Interaction Template prototype to simulate the task model. Is it possible to have the traditional task model simulator control the prototype? The reverse mapping might be more difficult to implement. Is it possible to have template prototypes communicate with each other. For example, if a task in one template has an enabling-with-info-exchange relationship to a task in another template, is it possible to forward the information from one prototype to the other? Exchanging information between prototypes would require the prototypes to share a common information exchange mechanism.

References

[1] BARON, M., AND GIRARD, P. Suidt: A task model based gui-builder. In *First International Workshop on Task Models and Diagrams for User Interface Design - TAMODIA 2002* (2002).

[2] CORDY, J., HALPERN-HAMU, C., AND PROMISLOW, E. Txl: A rapid prototyping system for programming language dialects. *Computer Languages 16*, 1 (1991), 97–107.

[3] DITTMAR, A., AND FORBRIG, P. The influence of improved task models on dialogues. In *Fourth International Conference on Computer-Aided Design of User Interfaces* (2004), pp. 1–14.

[4] LUYTEN, K., AND CLERCKX, T. Tasklib: a command line processor and library for concurtasktrees specifications. http://www.edm.luc.ac.be/software/TaskLib/.

[5] LUYTEN, K., CLERCKX, T., CHONINX, K., AND VANDERDOCKT, J. Derivation of a dialog model from a task model by activity chain extraciton. In *Design, Specification and Verification of Interactive Systems 2003 (DSV-IS 2003)* (2003), Springer-Verlag, pp. 191–205.

[6] PAQUETTE, D., AND SCHNEIDER, K. A. Interaction templates for constructing user interfaces from task models. In *Fourth International Conference on Computer-Aided Design of User Interfaces* (2004), pp. 223–235.

[7] PATERNÓ, F. Concurtasktreesenvironment (ctte). http://giove.cnuce.cnr.it/ctte.html.

[8] PATERNÓ, F. *Model-Based Design and Evaluation of Interactive Applications*. Springer, 2000.

[9] PATERNÓ, F. Task models in interactive software systems. In *Handbook of Software Engineering and Knowledge Engineering*, S. K. Chang, Ed. World Scientific Publishing Co., 2001.

[10] PATERNÓ, F. Tools for task modelling: Where we are, where we are headed. In *First International Workshop on Task Models and Diagrams for User Interface Design - TAMODIA 2002* (2002), C. Pribeanu and J. Vanderdonckt, Eds., pp. 10–17.

[11] UHR, H. Tombola: Simulation and user-specific presentation of executable task models. In *Proceedings of HCI International* (2003).