Task Model Simulation Using Interaction Templates

David Paquette and Kevin A. Schneider

Department of Computer Science, University of Saskatchewan, Saskatoon SK S7N 5C9, Canada

Abstract. Interaction Templates were previously introduced as a method to help ease the construction of ConcurTaskTrees. In this paper, a language for defining Interaction Templates, the Interaction Template Definition Language, is introduced. This paper also demonstrates how Interaction Templates can be used to enhance task model simulation, allowing users to interact with concrete user interface components while simulating task models. A prototype task model simulator illustrates how Interaction Templates can be used in task model simulation.

1 Introduction

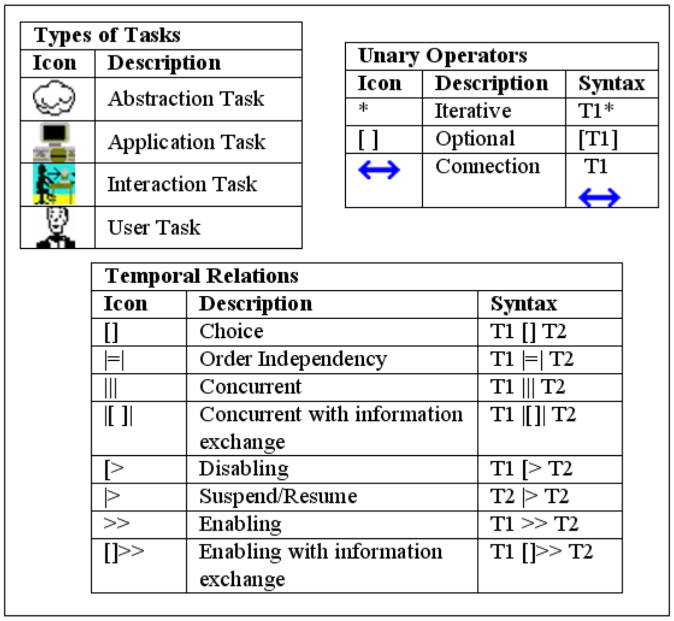
Model based approaches to interactive system design are based on the specification and evaluation of interactive systems using high-level models [9]. Using high-level models to specify interactive systems can help designers to focus on specifying the requirements and the behaviour of the system rather than immediately being hindered by implementation details. High-level models can be evaluated, often with simulation tool support, before implementation has begun, allowing for a refinement of the system specification with fewer resources than if source code were involved in the change. Task models focus on describing interactive systems in terms of user goals and the tasks required to reach those goals. Many model based approaches, such as ADEPT[13], SUIDT[5], and U-TEL/MOBI[12], acknowledge the importance of task models.

Interaction Templates [7] are a template based approach to task modelling. We will show how Interaction Templates can be used to promote re-use and consistency in task models, aiding in the building and understanding of task models. We will also show how concrete user interface components can be used with Interaction Templates to enhance the task model simulation process.

2 Background and Related Work

2.1 ConcurTaskTrees

ConcurTaskTrees (CTT) is a graphical notation used to describe interactive systems [10]. With CTT, tasks are arranged hierarchically, with more complex tasks broken down into simpler sub-tasks. CTT includes a rich set of temporal operators that are used to describe the relationship between tasks, as well as unary operators to identify optional and iterative tasks. A summary of the CTT notation can be seen in Figure 1.



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

Fig. 1. Summary of the ConcurTaskTrees notation

2.2 ConcurTaskTree Simulation

One of the powerful features of CTT is the ability to simulate task models at an early stage in the development process, allowing for a simulation of the system before implementation has started. 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, we will discuss the process involved in simulating ConcurTaskTrees. Next, an overview of two specific task model simulators will be given.

The Simulation Process ConcurTaskTree simulation involves, in some way, the simulated performance of specific tasks in order to reach a pre-defined goal. In a ConcurTaskTree, tasks are related to each other according to their temporal relations and hierarchical breakdown. Depending on what tasks have been performed, some tasks are enabled and others are disabled. The first step in ConcurTaskTree simulation 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) [9]. Enabled tasks sets are identified according to the rules laid out in [9]. The set of all enabled task sets for a specific task model is referred to as an enabled task collection (ETC).

Having identified the enabled task collection 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). In this state transition network, each ETS is a state, and transitions between enabled task sets occur when tasks are performed. The final preparation step for simulation is to calculate the initial state. A simple example illustrating a ConcurTaskTree and it's STN is shown in Figure 2. A command-line tool called TaskLib [3] can be used to extract the ETC, STN, and initial state from a CTT. The details of TaskLib's implementation can be found in [4].

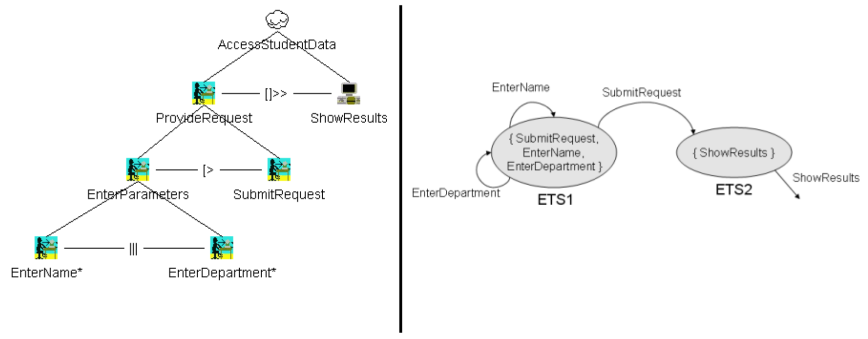


Fig. 2. A CTT (left) and it's State Transition Network (right)

Once the ETC, STN, and initial state have all been 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. As will be discussed shortly, how tasks are performed differs between simulation tools.

Simulation Tools

Basic Simulators The most basic simulators, such as the one shown in Figure 3, simply display the currently enabled tasks in a list. In these simple simulators, double-clicking on a task simulates 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) [8], a tool for both building and simulating task models.

Dialogue Graph Editor The Dialogue Graph Editor[2], a tool developed at the University of Rostock, provides a more complex simulation than the basic simulator found in CTTE. The Dialogue Graph Editor makes use of an extension to CTT that allows for the definition of finite sets of concurrent instances of actions. The Dialogue Graph Editor allows designers to create views and assign tasks from a task model to those views. The views can later be used to simulate the task model as shown in Figure 4. When simulating the task model, views are represented as windows, elements (as well

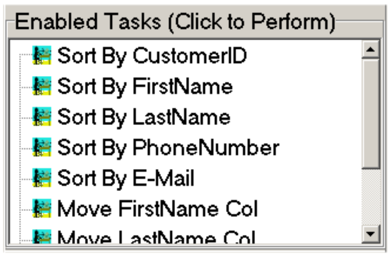


Fig. 3. A simple ConcurTaskTrees task model simulator

as tasks) inside the windows are represented by buttons, and transitions between states are represented by navigation between windows. 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. For example, clicking a button to simulate an interaction task that does not normally involve a button widget may seem strange to end users that may be involved in the simulation. The key advantage in Dialogue Graph Editor is the ability to organize tasks into a dialog. This requires an additional dialog model as well as a mapping between the dialog model and task model.

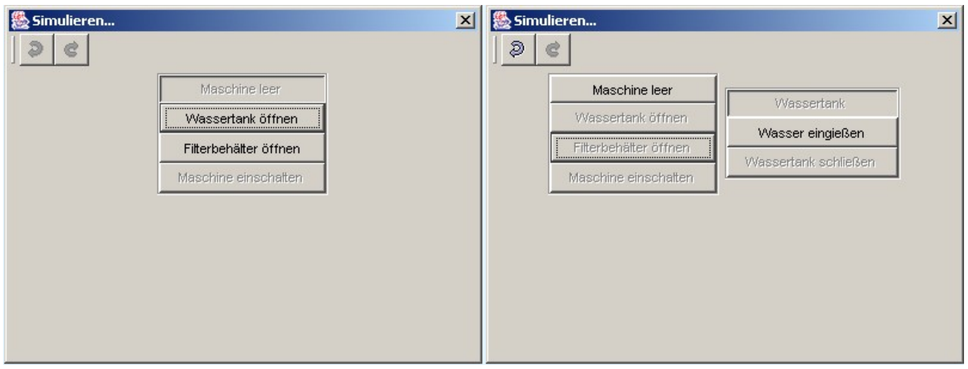


Fig. 4. Simulator included in Dialogue Graph Editor

3 Interaction Templates

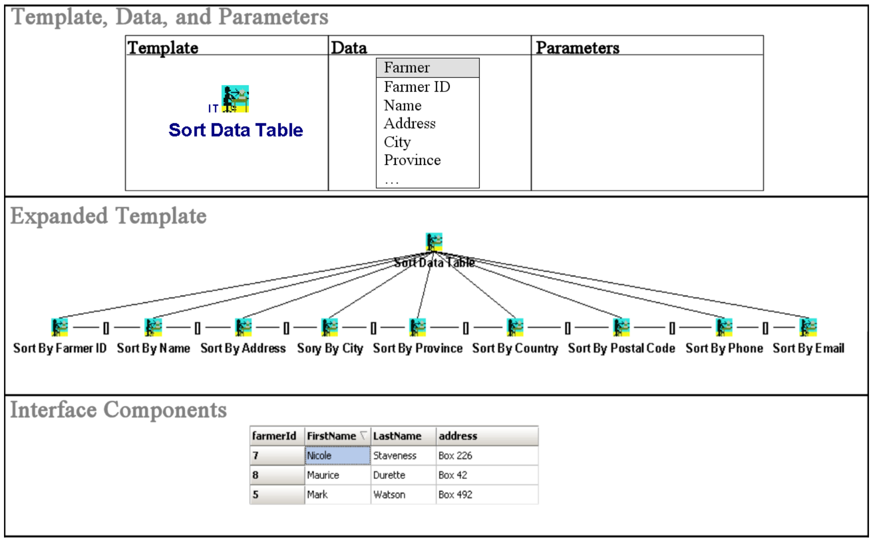
Task modelling has been shown to be useful when designing interactive systems [10]. Unfortunately, the task modelling process can become tedious and models can become very large when modelling non-trivial systems. Previous research has shown that while building task models to specify information systems, there are often subtrees that repeat throughout the model with only slight variations [7]. These subtrees are often associated with common interface interactions found in information systems. Interaction Templates [7] 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 any point in the model. Inserting and customizing Interaction Templates reduces the need to repeatedly model similar interactions in a system, and thus, can greatly reduce the time spent modelling information systems. Interaction Templates are intended to help developers build task models quickly, and allow for detailed simulation while maintaining a useful system overview. As well, Interaction Templates can be designed and tested to ensure their usability in reaching user's goals. Interaction Templates are intended to provide abstract user interface plasticity as discussed in [11].

3.1 Defining Interaction Templates

Figures 5 and 6 outline how Interaction Templates are structured. An Interaction Template includes a set of parameters, some data, and a definition of how the template behaves depending on the data and parameters it is provided. Figure 5 shows a small portion of an Interaction Template that models a data table interaction. The data provided to this template is a schema defining the data that will be shown in the table. No parameters are needed for this portion of the data table Interaction Template. The Sort By 'Column' task is repeated for each column in the supplied data. Given the Interaction Template definition and the data, the template can be transformed into the expanded template shown in the figure.

Figure 6 displays the behaviour of an Interaction Template that models a selection task. This template includes some data in the form of a list specifying the options that are available to the user. This template also includes a parameter specifying whether or not the user is permitted to select multiple options. The template is expanded differently depending on the value of the MultiSelect parameter.

The above two examples have shown the types of transformations that are needed to define Interaction Templates. The remainder of this subsection will show an example of how we can define Interaction Templates that are capable of adapting their behaviour as shown in the above examples. Interaction Templates are described using a custom markup language, called the Interaction Template Definition Language (ITDL). The ITDL is embedded inside an XML description of a ConcurTaskTree. The *it:* namespace is used to denote elements that describe the options and behaviour of Interaction Templates. The XML language used here to describe ConcurTaskTrees is a modification of the language used by the ConcurTaskTreesEnvironment (CTTE) [8].



farmerId	FirstName	LastName	address
7	Nicole	Staveness	Box 226
8	Maurice	Durette	Box 42
5	Mark	Watson	Box 492

Fig. 5. A portion of the Data Table Interaction Template. This template shows how subtasks can be repeated for each field in a data element.

Figure 7 shows an Interaction Template defined using the ITDL. The root task of an Interaction Template is surrounded by an identifying *it:template* tag. The *it:template* tag 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 different types of 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 an Interaction Template's task tree changes according to the options specified for the template. An Interaction Template's adaptive behaviour is defined using two commands: *it:case* and *it:foreach*.

The *it:case* command is used to select a specific task or subtask based on the option values for the template. An *it:case* command contains one or more *it:condition* statements. When evaluated, the *it:case* command will select the first *it:condition* whose expression attribute evaluates to true. The *it:case* command 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.

The *it:foreach* command 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. Inside the *it:foreach* statement, the current element

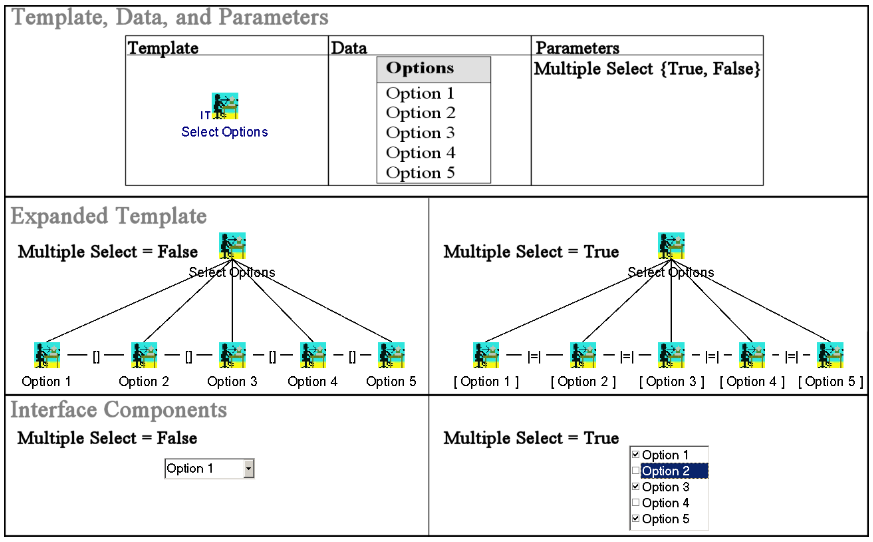


Fig. 6. An Interaction Template for selection from a list of options. This template gives the option to select a single option, or to select multiple options.

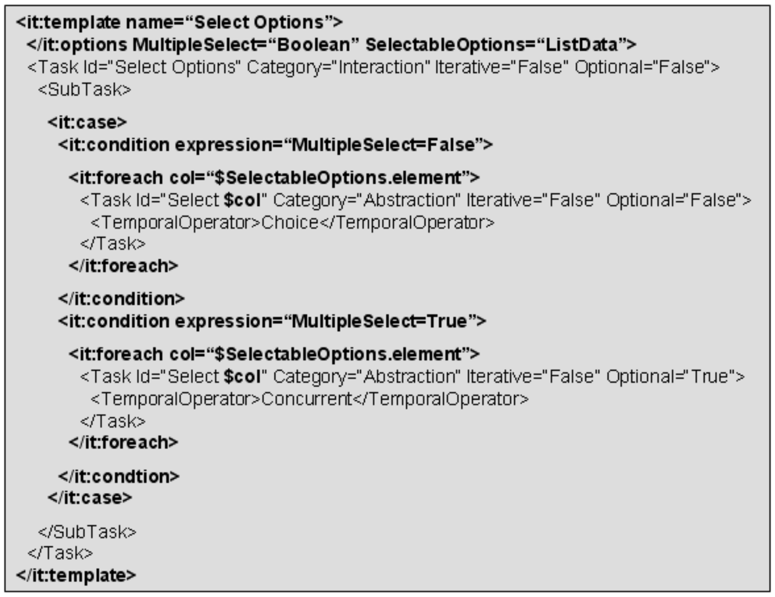
is referenced by the name of the single attribute of the *it:foreach* statement. The current element's name attribute is referenced by adding '.name' to the element reference.

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* commands. 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* commands has been implemented using TXL, a rule-based tree transformation language [1]. The prototype is fairly simple, consisting of only 5 rules implemented in just over 100 lines of TXL code.

3.2 Using Interaction Templates

After an Interaction Template has been defined using the ITDL as described above, using an Interaction Template is simply a matter of inserting the template into a CTT and setting values for the template's options. Once the options have been set, the Interaction Template is expanded using an ITDL interpreter. It is always possible to edit the expanded Interaction Template to customize the template to a specific use. It is also possible to change the options for a template and have the Interaction Template re-interpreted to reflect those changes. Edits to an expanded Interaction Template are recorded and re-applied to the re-interpreted Interaction Template if possible. Currently, no tool support exists for building task models using Interaction Templates.



```
<it:template name="Select Options">
  </it:options MultipleSelect="Boolean" SelectableOptions="ListData">
  <Task Id="Select Options" Category="Interaction" Iterative="False" Optional="False">
    <SubTask>

      <it:case>
        <it:condition expression="MultipleSelect=False">

          <it:foreach col="$SelectableOptions.element">
            <Task Id="Select $col" Category="Abstraction" Iterative="False" Optional="False">
              <TemporalOperator>Choice</TemporalOperator>
            </Task>
          </it:foreach>

        </it:condition>
        <it:condition expression="MultipleSelect=True">

          <it:foreach col="$SelectableOptions.element">
            <Task Id="Select $col" Category="Abstraction" Iterative="False" Optional="True">
              <TemporalOperator>Concurrent</TemporalOperator>
            </Task>
          </it:foreach>

        </it:condtion>
      </it:case>

    </SubTask>
  </Task>
</it:template>
```

Fig. 7. An ITDL definition of an Interaction Template for selecting options from a list

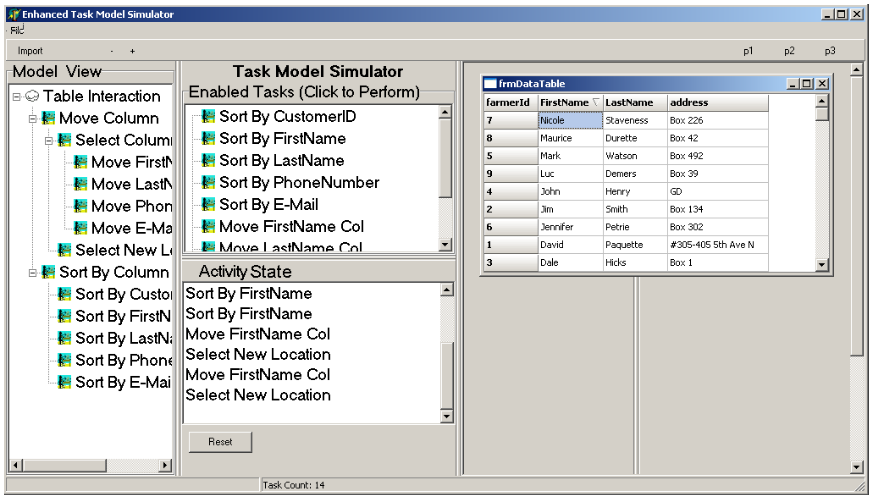
4 Simulation with Interaction Templates

This section will show how partial user interface prototypes can be created from task models that are built using Interaction Templates. These partial prototypes can be used to enhance the task model simulation process, allowing users to interact with concrete user interface components to simulate portions of task models. PetShop[6] provides a different approach for simulating an interactive system. ConcurTaskTrees are mapped to a Petrie net based notation. They argue that detailed design is more appropriately specified with a model notation other than ConcurTaskTrees. We intend task notation to provide a more seamless transition when simulating the software for the end-user.

4.1 Enhanced Task Model Simulator

While Interaction Templates model common interface interactions found in information systems, there also exist concrete user interface components that implement many of those interactions. In interface builders such as Borland's Delphi, interfaces are composed using sets of 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 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 [3]. 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 activity state. 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 current activity state. The activity state 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 the 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.



Model View:

- Table Interaction
  - Move Column
    - Select Colum
      - Move FirstN
      - Move LastN
      - Move Phon
      - Move E-Ma
    - Select New L
  - Sort By Column
    - Sort By Custo
    - Sort By FirstN
    - Sort By LastN
    - Sort By Phone
    - Sort By E-Mai

Enabled Tasks (Click to Perform):

- Sort By CustomerID
- Sort By FirstName
- Sort By LastName
- Sort By PhoneNumber
- Sort By E-Mail
- Move FirstName Col
- Move LastName Col

Activity State:

```
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

Fig. 8. A prototype of the Enhanced Task Model Simulator. Users can interact with the data table, shown on the right of the screen, to control the task model simulator.

Interaction Template prototypes are manually built once, then instantiated 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. Common functionality between prototypes includes the ability to communicate with the simulator, as well as the ability to show and hide itself as controlled by the simulator.

Each specific prototype component implements its own adaptation logic. When a prototype object is created, it reads the *it:options* tag that contains the options for the current use of the Interaction Template. The prototype object adapts itself to the options specified in the *it:options* tag. With the Data Table Interaction Template for example, the data table prototype reads in the schema file to set the column headers and reads 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 attribute values of the interface component. Finally, each specific prototype implements 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 specifying the name of the task that is performed when an event is triggered.

While other task model simulators use abstract interface objects to simulate tasks, concrete user interface components can be used to simulate tasks when Interaction Templates 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 potentially easier for users to understand.

5 Conclusions and Future Work

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 introduced. The Enhanced Task Model Simulator has shown how concrete user interface components can be used to simulate task models using Interaction Templates. This paper concludes with a discussion of the current state of our research and plans for future work.

Tool Support: Tool support is needed both for 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 in order to interpret an Interaction Template and expand it based on the values of the options that are set for the current use of the template. An Interaction Template based task modelling tool called Model-IT is currently being developed. A preliminary screenshot of Model-IT can be seen in Figure 9. There are plans to integrate the Enhanced Task Model Simulator into Model-IT in the near future.

Creating User Interface Prototypes: Currently, prototypes are manually built to be self adaptive to the options set for an Interaction Template. Linking events to specific

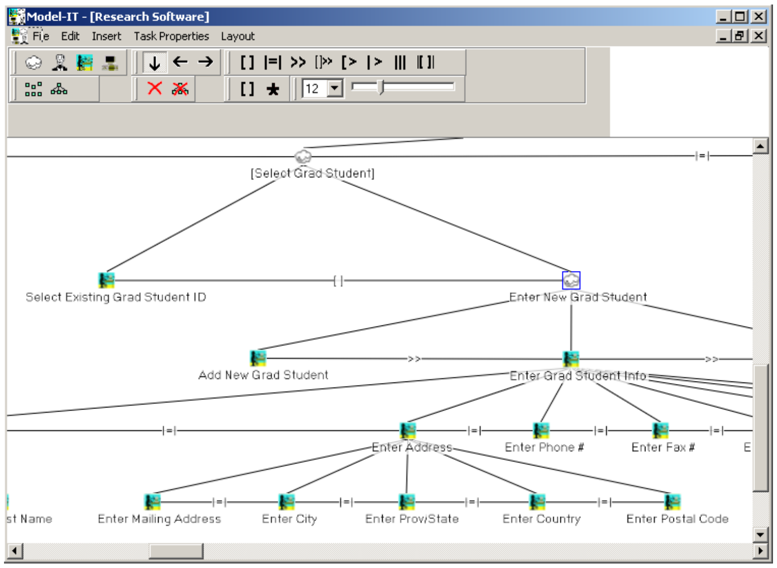


Fig. 9. Model-IT: An Interaction Template based task modelling tool

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, some 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.

Linking Task Models to Final Implementations: Potentially, the technique used to map interface component events to tasks in a task model could be used in the final implementation of a system. The advantages of allowing this mapping to remain in a system's final implementation include the ability to: keep track of a user's current state in the system's task model, verify the system's implementation correctly matches the task model, and to suggest help to users based on their current state.

6 Acknowledgements

The authors wish to thank the National Sciences and Engineering Research Council of Canada (NSERC) and Western Ag Innovations for their support.

References

1. CORDY, J., HALPERN-HAMU, C., AND PROMISLOW, E. Txl: A rapid prototyping system for programming language dialects. *Computer Languages 16*, 1 (1991), 97–107.
2. 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.
3. LUYTEN, K., AND CLERCKX, T. Tasklib: a command line processor and library for concurtasktrees specifications. http://www.edm.luc.ac.be/software/TaskLib/.
4. 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.
5. MICKAEL, B., AND GIRARD, P. Suidt: A task model based gui-builder. In *Proceedings of the 1st International Workshop on Task Models and Diagrams for User Interface Design TAMODIA 2002* (2002), C. Pribeanu and J. Vanderdonckt, Eds., INFOREC Printing House.
6. NAVARRE, D., PALANQUE, P. A., PATERNÓ, F., SANTORO, C., AND BASTIDE, R. A tool suite for integrating task and system models through scenarios. In *DSV-IS '01: Proceedings of the 8th International Workshop on Interactive Systems: Design, Specification, and Verification-Revised Papers* (London, UK, 2001), Springer-Verlag, pp. 88–113.
7. 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.
8. PATERNÓ, F. Concurtasktreesenvironment (ctte). http://giove.cnuce.cnr.it/ctte.html.
9. PATERNÓ, F. *Model-Based Design and Evaluation of Interactive Applications*. Springer, 2000.
10. 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.
11. SCHNEIDER, K. A., AND CORDY, J. R. Abstract user interfaces: A model and notation to support plasticity in interactive systems. In *DSV-IS* (2001), C. Johnson, Ed., vol. 2220 of *Lecture Notes In Computer Science*, Springer, pp. 28–48.
12. TAM, R., MAULSBY, D., AND PUERTA, A. U-tel: A tool for eliciting user task models from domain experts. In *Proceedings IUI 98* (1998), ACM Press.
13. WILSON, A., JOHNSON, P., KELLY, C., CUNNINGHAM, J., AND MARKOPOULOS, P. Beyond hacking: A model-based approach to user interface design. In *Proceedings HCI'93* (1993), Cambridge University Press.