Considering Task Pre-Conditions in Model-based User Interface Design and Generation

Marco Manca, Fabio Paternò, Carmen Santoro, Lucio Davide Spano
CNR-ISTI, HIIS Laboratory
Via Moruzzi 1, 56124 Pisa, Italy
{marco.manca, fabio.paterno, carmen.santoro, lucio.davide.spano@isti.cnr.it}

ABSTRACT
Deriving meaningful and consistent user interface implementations from task models is not trivial because of the large gap in terms of abstraction. This paper focuses on how to handle task preconditions in the design and generation process, an issue which has not adequately been addressed in previous work. We present a solution that is able to manage the information related to task pre-conditions at the various possible abstraction levels. The paper also reports on some example applications that show the generality of the solution and how it can be exploited in various cases.

Author Keywords
Task models, Model-based User Interface Design, User Interface Generation

ACM Classification Keywords
H.5.m. Information interfaces and presentation (e.g., HCI): Miscellaneous.

INTRODUCTION
Task models have been widely investigated in the literature, since they provide a structured representation of how different activities should be carried out for reaching users' goals. They are popular for their high-level description that can be understood by people without programming background and therefore can be used for communicating between the different actors involved in the design and development process: designers, developers and users.

According to the CAMELEON Reference Framework [2] there are four abstraction levels in model based-design and generation: task models, abstract user interfaces, concrete user interfaces and implementations. Thus, when generating from task models it is possible to go through all of them for progressively refining the interactive application description. Over time the notations for task models have become more expressive in order to be able to describe more complex sets of activities. However, this increase in expressiveness raises new challenges for model-based user interface design and generation. In this paper, we show how some of such challenges can be addressed. In particular, we show how to handle pre-conditions expressed in task models both in the task model analysis phase and when refining them through the various abstraction levels. We describe our solution by considering ConcurTaskTrees (CTT) [7], a widely known notation for task models, and show how we have designed the solution's implementation in the analysis environment and in the refinement process. The corresponding user interface can be derived through the abstract user interface (AUI) and the concrete user interface (CUI) abstraction levels. We consider the MARIA language [8] for such abstraction levels.

In the paper, after discussing related work, we describe the method that we have designed and implemented for managing the pre-conditions information across the various abstraction levels. We then describe some example applications showing how such new features can be exploited in various cases, and lastly we draw some conclusions with indications for future work.

EICS'14, June 17 - 20 2014, Rome, Italy

ACM 978-1-4503-2725-1/14/06…$15.00.
http://dx.doi.org/10.1145/2607023.2610283

RELATED WORK
Many tools and notations based on hierarchical descriptions and different sets of operators exist for describing task models. A comparison is available in [9]. Most of them include the possibility to specify conditions on the task execution, but often this specification is limited to natural language, in order to reduce the modelling complexity. Examples of such approach can be found in both research (e.g. UsiXML [4] and Hamsters [5]) and commercial tools (e.g. IBM Information Architecture Workbench [3]). They include a *precondition* and/or a *postcondition* for the task execution, which are defined as strings. This approach does not allow manipulating the condition definition while generating code or other models from the task description.

Other modelling approaches select a structured representation of pre- and post-conditions, describing them through Boolean predicates to be checked on the domain objects manipulated by tasks. The cost of having a more complex representation is balanced by the possibility of

enabling tool support for model checking during definition and simulation. For instance, the KMADe [1] tool supports model checking, while AMBOSS [3] simulates different scenarios allowing the designer to indicate interactively whether a condition is satisfied or not. Thus, there is a lack of tools able to manage preconditions in task models, in the user interface design and generation process, and in this paper we present a solution to cover this gap.

HANDLING PRECONDITIONS AT THE VARIOUS ABSTRACTION LEVELS

The design and implementation of interactive applications starting with task models and involving preconditions can go through various steps:

- Specification of task models, including task preconditions;
- Analysis of the dynamic behavior of the task model with preconditions through the interactive simulator, which allows designers to enter values associated with the data defining the preconditions as well.
- In the case of service-based applications, it is possible to bind Web service operations and corresponding tasks. Bindings are then exploited at runtime (by the generated application) to dynamically get values for the user interface objects that are considered when checking the preconditions.
- Identification of Presentation Task Sets (which are sets of tasks associated with the presentations in the user interface logical descriptions), with the support of some heuristics to merge them [8];
- Generation of the corresponding logical UI descriptions at the abstract and/or concrete level;
- Generation of the corresponding Final UI (FUI) implementation.

In this refinement process it is important to identify tasks that are mutually exclusive because of their preconditions. This happens when the preconditions associated to these tasks involve shared variables that assume different values: according to the values that are taken, either one task or the other is enabled. Consequently, the mutually exclusive tasks will be associated with different user interface presentations that correspond to the different cases. For example, if the precondition is associated with the user's role then depending on it (associated with the preconditions) different user interfaces should be enabled.

In the refinement process, we exploit two concepts: *Presentation Task Sets* (PTSs) [8] and *conditional connections*. Tasks enabled over the same period of time according to the temporal constraints indicated in the task model are grouped into PTSs. The latter are automatically calculated through an algorithm that takes as input the formal semantics of the temporal operators of the CTT notation and a task model. In the transformation into an abstract or concrete user interface each PTS corresponds to a presentation. In order to avoid fragmented user interfaces some PTSs can be merged according to heuristics [8]. Navigating from a resulting presentation to another is described through connections, which are defined by the interface element activating the navigation and the target presentation. Conditional connections are a particular type of connection: they model cases where moving from a presentation to another is triggered only if the specified condition is verified. Therefore, the approach proposed to appropriately support task preconditions in the logical description of UIs is first to include mutually exclusive tasks in distinct PTSs, and when deriving the corresponding logical user interface description, set a conditional connection able to support moving to a presentation or another according to the value of the variable associated with the precondition. In the following we analyse more in detail the steps of the approach proposed and how to consider task pre-conditions in each of them.

Task Model Specification

In this step the designer should specify tasks, their temporal relationships, objects manipulated by tasks and possible task preconditions. A task precondition indicates what must be verified before the task is carried out. Mutually exclusive preconditions on two (or more) tasks indicate that the associated tasks cannot be enabled at the same time, for instance the preconditions involve some Boolean expression or a comparison with numbers having a value higher/equal/less than another, or a comparison between strings, etc. Thus, they indicate that the tasks cannot be enabled at the same time simply because they are associated with (pre)conditions that cannot hold at the same time.

Preconditions in Interactive Task Model Simulation

In order to analyse how the task model behaviour varies depending on the values associated with the objects defining the preconditions it is possible to use the interactive simulator. CTTE [6] provides an interactive simulator, useful for checking the task model against usage scenarios. The simulator highlights the leaf task nodes that can be executed at a given time, considering the temporal relationships among them (defined in the model). The user can simulate the performance of a task by double-clicking its icon. After that, the simulator updates the set of enabled tasks.

We have extended the existing simulator for supporting pre- and post-conditions and analyse their impact on the dynamic behaviour of the tasks. We added a panel for controlling the dynamic state of task objects. Different scenarios can be supported by specifying different object values exploited by the preconditions. The designer can modify the values before each simulation step, through the interface shown in Figure 1. The panel shows each object's name, its type (e.g. string, integer etc.), the current value

(which can be interactively modified) and the list of tasks that manipulate the object. In each step, the simulator updates the list of enabled tasks, disabling those that have preconditions that are not satisfied by the current object values. In addition, it prompts the user when one or more post-conditions are not verified after the execution of a task. In this way, designers can verify the correspondence between the model and different usage scenarios.

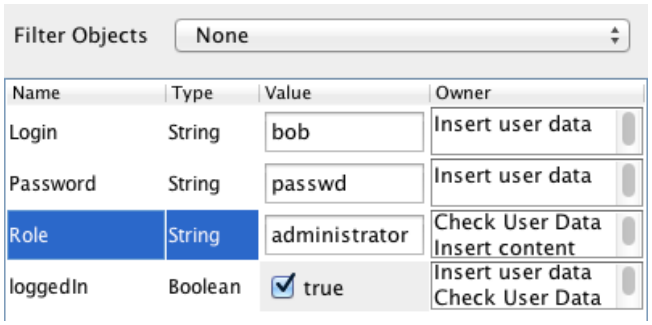

Figure 1: Task model simulator interface for objects

Generation of PTS

PTSs are sets of elementary tasks enabled in the same period of time and associated with a given presentation when transforming the task model into an abstract or concrete user interface specification. PTSs are identified taking into account temporal relationships between tasks [8]. For instance, if two tasks are composed by the choice operator, they are both enabled at the same time, therefore they should belong to the same PTS. However, preconditions have an impact on the PTS definition. For instance, two tasks composed by the choice operator are both enabled and they should belong to the same PTS. However, if both tasks have a precondition involving the same object and these preconditions are mutually exclusive, then the tasks cannot belong to the same PTS because they cannot be enabled at the same time. Thus, we had to extend the rules to identify PTSs with the following rule:

a) if in the task model there are two (or more) sibling tasks composed through a choice relationship, and

b) these tasks share one (or more than one) object, and

c) these tasks have mutually exclusive preconditions involving one (or more than one) shared task object

then such tasks should be included in distinct PTSs.

Application of heuristics for PTSs processing

After having generated PTSs according to the semantics of the temporal operators, the designer can apply some heuristics to merge two or more PTSs.

This is done in order to avoid generation of fragmented user interfaces with many presentations with little content. The heuristics that are currently supported are the following:

- If two (or more) PTSs differ by only one element and their elements are at the same level composed by an enabling operator, they can be joined together.
- If a PTS is composed of just one element, it can be included within another superset that contains such element.
- If two (or more) PTSs share most elements, they can be unified in order not to duplicate elements that are already available in another presentation.
- If there is an exchange of information between two tasks, they can be put in the same PTS in order to highlight such relation.

The automatic support guarantees that tasks having mutually exclusive preconditions involving the same objects are kept in distinct PTSs even *after applying such heuristics*. In the tool, if the generated PTSs contain tasks with mutually exclusive preconditions, they are not merged (i.e. the application of heuristics has no effect on the PTSs), otherwise they are merged according to the above heuristics.

Generation of Abstract and Concrete User Interfaces

In this step for each PTS a presentation is created together with connections to enable navigation to/from other presentations. The type of connection that is created between two presentations depends on whether the target PTS contains a task with preconditions. On the one hand, if a target PTS does not contain tasks with preconditions then an elementary connection is created in the source PTS. On the other hand, if a target PTS contains tasks with preconditions, then a conditional connection is created in the presentation corresponding to the source PTS. The condition in the connection is associated to the expression defined in the precondition.

Generation of the Final User Interface

In this step the automatic support generates the UI final code supporting what was specified in the more abstract UI descriptions. For instance, it generates appropriate code for supporting conditional connections amongst various presentations. In particular, the final code generated enables the UI to move to the target presentation only if the condition contained in the conditional connection is verified. In next sections we detail how the conditional connections are implemented.

Dynamic association of precondition values

At run-time the actual values for the objects used in the precondition can be received in various ways. In some cases the values can be entered by the user. In other cases, in order to associate data values with preconditions it is useful to exploit bindings between the application tasks and the operations specified in the Web services, which should implement the tasks. Out tool allows designers to specify such bindings at design time, and then they will be

exploited at runtime by the generated application to get actual values for the precondition parameters and then check the precondition validity. An illustrative example showing how the binding phase works with the support of the tool is provided in the next section.

EXAMPLE APPLICATIONS

We present three examples that show the broad impact that task models with preconditions can have and the associated issues that we have solved.

Content Management System Access

In this section we show an example application to demonstrate how we manage task model preconditions during the generation process involving first Presentation Task Sets, then AUI/CUI and finally the FUI.

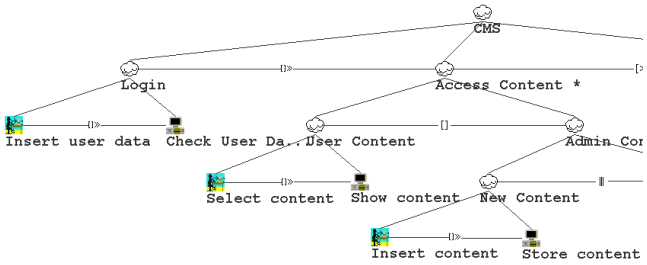

Figure 2: Content Management System example

Figure 2 shows an excerpt of the task model of a Content Management System (CMS) that allows publishing, editing and modifying the content in a Web application.

This model contains different tasks, such as '*Select Content*', '*Insert Content*', that can be performed depending on the user role. These tasks are composed by the choice operator and are enabled after the 'Check User Data' task has been performed.

Two (or more) tasks composed by a choice operator usually are enabled at the same time, but in this case only one task can be enabled at a given time, and the choice depends on the precondition expressed in the task properties.

In particular, the first of the considered tasks describes the possibility for people with the 'user' role to select an article and read it. The other one models the activity to insert content for users with the administrator role. Both tasks contain a pre-condition that describes when they can be enabled. The precondition concerns the same task object, but the values are different and mutually exclusive. For this reason, these two tasks cannot belong to the same Presentation Task Set and during the generation process should be placed in different PTSs. Also during the heuristics application process, even if one of the tasks could be merged with another, they are kept in distinct PTSs.

At run-time the interactive application needs a Web service that given the user data returns the user role. For this reason, a binding between the 'Check User Data' task and the corresponding Web Service operation should be specified at design time. For this purpose, our tool allows designers to interactively select a task and an operation of the analysed Web services, which can be automatically imported. Once this binding has been created, the tool shows input and output parameters of the operation along with the corresponding tasks, which should provide the input parameters and receive the output ones (see Figure 3). Such tasks are automatically identified through an analysis of the task model [8], the association can be modified by the designer if something is wrong. In order to manage the preconditions, we have introduced in the tool the possibility to automatically identify whether an output parameter of the Web service corresponds to a data object used in a precondition. In this case, the tool allows the designer to specify for which values of the output parameter the precondition is true in each of the tasks associated with it.

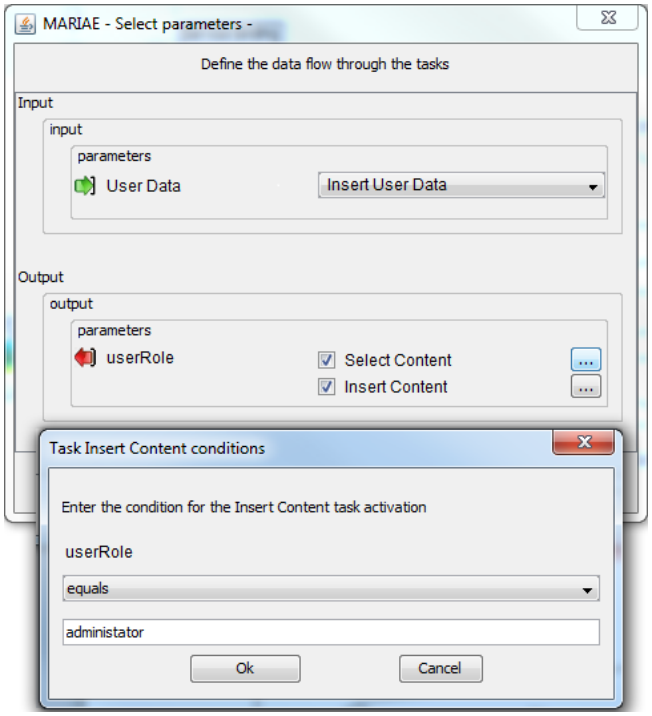

Figure 3: Example of binding between a task and a web service operation

In more detail, we assume to use a Web service implementing the login operation that, according to a username and a password, returns a string indicating the user's role (e.g. "administrator" or "user"), if the login is successful. The relevant excerpt of the related task model is shown in Figure 2. The *Insert Content* interactive task is available only for administrators. Therefore, it contains a precondition checking if the role task object is equal to "administrator". It is possible to bind a task in the model and an operation in the web service through the following steps: 1) importing a description of the web service operation (specifying its URL), 2) connecting a system task with a web service operation (in our example the Check User Data task

and the login operation), 3) specifying which interactive task provides the input values for the operation (Insert User Data) and 4) specifying which tasks receives the operation output, connecting the returned values and the task objects (Figure 3). In our case, the string returned by the login operation is therefore connected with the role task object. The binding allows the generator to exploit the web service result at runtime in order to check the precondition. For this purpose the designer should have indicated which value is associated for each of the two tasks. Figure 3 shows the case in which the designer selects the administrator value for the Insert Content task.

Three PTSs are generated from the tasks considered: the first one contains 'Insert User Data' and 'Check user Data' tasks and the other two contain one task with one precondition each. Consequently, the automatic support generates one presentation for each PTS. Since there are PTSs containing tasks with preconditions, then in the first presentation one conditional connection containing two target presentations is generated in the MARIA description (see Figure 4): these presentations are those generated from the PTSs with preconditions. The values of the 'data reference' attribute are associated with the data model element containing the user role value, and are those entered during the binding operation described before.

```
<connections>
  <conditional_conn interactor_id="P1_Check_User_Data">
    <target presentation_to_load="Show_Content_Presentation">
      <condition
        data_reference="data:[ns1_checkUserData/userRole==user]"/>
    </target>
    <target presentation_to_load="Insert_Content_Presentation">
      <condition
        data_reference="data:[ns1_checkUserData/userRole==administrator]"/>
    </target>
  </conditional_conn>
</connections>
```

Figure 4: Conditional Connection Example

For the implementation, the tool generates a JSP page in which, when an action is triggered, the value of the first operand expressed in the data reference (stored in a session field) is compared to the second operand contained in the data reference attribute (*user or administrator*). If the value is equal to one of the two values, then the JSP page transfers the control to the corresponding presentation (see **Figure 5**).

```
if(session.getAttribute("ns1_checkUserData/userRole")
    .equals("user") {
    pageContext.forward("Show_Content_Presentation.jsp");
} else
  if(session.getAttribute("ns1_checkUserData/userRole")
    .equals("administrator") {
    pageContext.forward("Insert_Content_Presentation.jsp");
}
```

Figure 5: Implementation of the example conditional connection

Educational Application

The second example we present is a task model that describes an Educational Application that, after a login task, permits displaying the student timetable, subscribe/unsubscribe course and other tasks that are omitted for the sake of brevity.

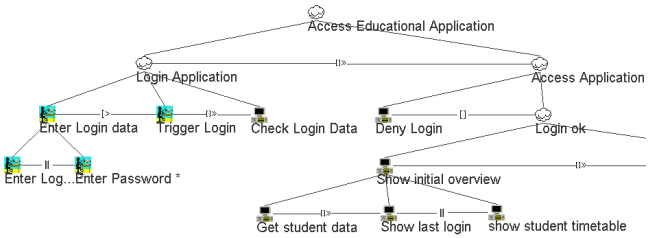

Figure 6: Educational Application Task Model

Figure 6 shows an excerpt of the task model: after performing the 'Check Login Data' there are two tasks enabled, 'Deny Login' and 'Get Student Data', since they are composed by the choice operator. These two tasks model activities that cannot be performed at the same time: the first one should be enabled if the login operation fails and the second one only if the login task has success. For this reason the designers have to specify a precondition that models the possibility to perform tasks only if the login service returns the Boolean value true. 'Deny Login' and 'Get Student Data' tasks share the same task object named *logged_in*, however the preconditions involve the same object but the values are mutually exclusive: the first task is enabled if the object logged_in is false and the second is enabled only if the object is true. For the reasons explained before these two tasks have to be placed in different PTSs and consequently in different presentations.

Figure 7 shows the generated user interface: when the login is successful, user's information and the student's timetable will be shown, otherwise the user will be redirected to an error page.

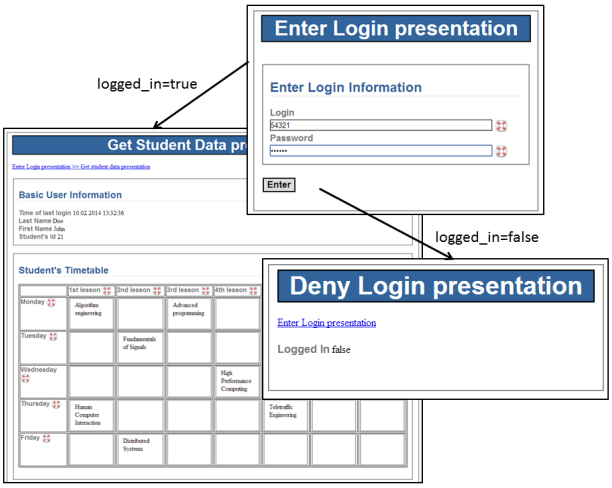

Figure 7: The User Interface generated

ATM Application

Figure 8 shows an example of an excerpt of an ATM task model. The bank's customers can *Enable Access* to the ATM (insert cart and then inset pin) and after they can *Withdraw*

Cash, *Deposit Cash*, *Get Information* and finally *Close the access*. *Withdraw Cash* is an abstract task and its sub-tasks describe the activities necessary to complete the withdrawal: *Select Withdraw, Show Possible Amount, Decide Amount, Select Amount, Check Amount* and *Check Cash*.

The task *Check Amount* is decomposed into two sub-tasks (*Provide Cash* and *Amount not valid*) that cannot be performed at the same time. Indeed, two preconditions are needed: the first one specifies that *Provide Cash* can be executed only if the amount is greater than zero, the other one states that *Amount not valid* is enabled if the amount is equal to zero. The preconditions share the same object (amount) and cannot be satisfied simultaneously, then the respective tasks are placed in different PTSs (and, accordingly in different presentations).

CONCLUSIONS AND FUTURE WORK

We have presented a method to handle task preconditions in the model-based user interface design and generation process, an aspect that has not adequately been addressed in previous work. Our solution is able to manage the information about task pre-conditions at various abstraction levels. A set of example applications has been considered in the paper to show how the approach works and demonstrates the effectiveness of the method in various cases.

Future work will be dedicated to a study involving several designers and developers applying our solution in various case studies in order to analyse its expressiveness and usability.

REFERENCES

1. Caffiau, S., Scapin, D., Girard, P., Baron, M., and Jambon, F. Increasing the expressive power of task analysis: Systematic comparison and empirical assessment of tool-supported task models. *Interacting with Computers 22*, 6 (2010), 569–593.
2. Calvary, G., Coutaz, J., Bouillon, L., Florins, M., Limbourg, Q., Marucci, L., Paternò, F., Santoro, C., Souchon, N., Thevenin, D., Vanderdonckt, J., 2002. The CAMELEON Reference Framework, Deliverable 1.1, CAMELEON Project, http://giove.isti.cnr.it/projects/cameleon/pdf/CAMELEON%20D1.1RefFramework.pdf.
3. Giese, M., Mistrzyk, T., Pfau, A., Szwillus, G., and von Detten, M. AMBOSS: A task modeling approach for safety-critical systems. In *Engineering Interactive Systems*. Springer, 2008, 98–109.IBM. IBM Information Architecture Workbench. http://www14.software.ibm.com/webapp/download/preconfig.jsp?id=2009-09-02+13%3A57%3A13.416731R&S_TACT=&S_CMP=
4. Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., and López-Jaquero, V. Usixml: A language supporting multi-path development of user interfaces. *Engineering Human Computer Interaction and Interactive Systems*, (2005), 200–220.
5. Martinie C., Palanque P., Winckler M.: Structuring and Composition Mechanisms to Address Scalability Issues in Task Models. INTERACT (3) 2011: 589-609
6. Mori G., Paternò F., Santoro C., "CTTE: Support for Developing and Analysing Task Models for Interactive System Design", IEEE Transactions on Software Engineering, pp.797-813, August 2002 (Vol. 28, No. 8), IEEE Press.
7. Paternò, F., 2000. Model-Based Design and Evaluation of Interactive Applications. Springer Verlag.
8. Paternò, F., Santoro, C., Spano, L.D.,: Engineering the Authoring of Usable Service Front Ends. Journal of Systems and Software 84(10): 1806-1822 (2011)
9. Paternò, F., Santoro, C., Spano, L.D., and Ragget, D. (eds). MBUI-Task Models. 2012. http://www.w3.org/TR/2012/WD-task-models-20120802/.

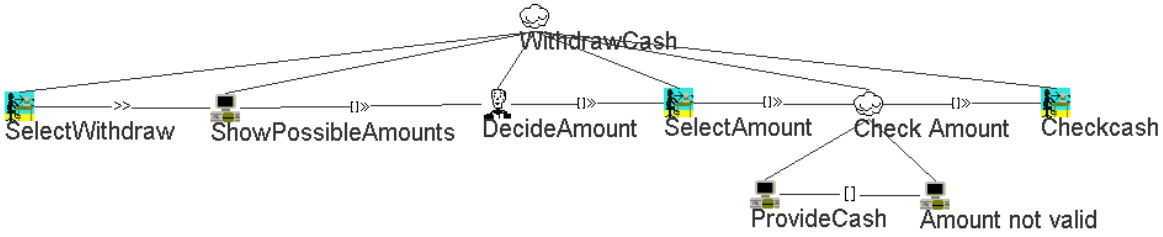

Figure 8: The ATM task model