Creating and Analysing Models in IBM Task Modeler

Mark Farmer
IBM United Kingdom Limited
Birmingham Road
Warwick, CV34 5JL
+44-1926-465232
farmerm@uk.ibm.com

Colin Bird
IBM United Kingdom Limited
Hursley Park
Winchester, SO21 2JN
+44-1962-816025
colinl_bird@uk.ibm.com

ABSTRACT

We illustrate the basic constituents of a model and demonstrate how the facilities of Task Modeler, such as the visualization options, enable the rapid creation, analysis, and communication of the model

Categories and Subject Descriptors

I.6.5 [Model development]: Modeling methodologies

General Terms

Documentation, Design, Human Factors.

Keywords

Modelling, HTA, Task Analysis, DITA

1. INTRODUCTION

IBM Task Modeler is a tool for creating rich models of human behaviour.

Originally developed to support pure task analysis techniques, as described by John Annet [3], Task Modeler allows an overall goal to be decomposed into tasks, and further into subtasks. The result is a Hierarchical Task Analysis (HTA). Later versions also support the development of hierarchical models other than the classic HTA.

In this paper we demonstrate how Task Modeler enables analysts to develop their task model representations visually and "rapidly create, explore, analyse, and share these models. [1]"

2. CREATING A MODEL

A model is developed to decompose and organize a complex area of interest until it is more fully understood. For example, an HTA can be created to understand the user's tasks during software development; or a DITA map [2] can be created to model the structure and content of an online help system [4]. Task Modeler supports the creation of models of different types through *dialects*.

2.1 Dialects

When creating a model in Task Modeler, you are asked which style of model to create. Task Modeler comes with a set of pre-defined modelling styles, each of which is defined by a Task Modeler dialect. Version 5 provides the following dialects: DITA maps, Roles and Goal models, Question Option Criteria (QOC) models, a use case analysis, mind maps and site maps.

A dialect includes templates and sample models, allowing models to be created and further developed with concrete examples. Dialect-specific method and modelling help is also included, and presented to the user throughout development. These dialect contents help the creation of new models, and aid the model's continued development.

2.2 Nodes and Properties

Each model dialect defines the model elements and metadata that construct the model, where Task Modeler represents each model element as a *node*, and each item of metadata as a *property*.

Once a model style is selected, a model can be created by hierarchically creating and arranging instances of the available node types, and setting property values.

As an example, to create a use case analysis model, a *Use Case* node is added to represent each use case. Use cases are decomposed into steps, by adding child *Step* nodes, which are further described by child *Actor Action* and *System Response* nodes. See Figure 1.

Figure 1: A use case decomposed into steps, actor actions and machine responses.

Each node type also provides a distinct set of properties to more fully describe the model. For example, a Use Case can define the *actors* that it applies to, and the *trigger* that is the source for the use case; a Step can be linked to by a source *Requirement* node, and document step-specific *issues*. The list of properties

can also be customized and extended, making Task Modeler a flexible modelling tool.

Each property is of a certain type, examples of which are: free-form text, integer, real number, constrained keyword, date, and an external file reference.

The property type serves two purposes, firstly to constrain the possible values, where appropriate, and secondly to allow analysis of the model's property values, in a type-specific way. This property analysis is described below.

Task Modeler has a highly visual modelling interface, allowing nodes to be created and added to the model efficiently and clearly. This is enhanced with intuitive keyboard shortcuts, based around the standard arrow keys, to rapidly add nodes relative to the selected node.

Nodes can be dragged to other parts of the model, allowing for rapid moving and copying of nodes. This promotes a hands-on and low-commitment modelling experience, very similar to using a stack of sticky notes.

The model editor automatically lays out the model hierarchically, ensuring the model structure is always clear during editing, but supports alternative model views. This allows analysts to concentrate on modelling, rather than ensuring the model is correctly and sensibly laid out.

3. ANALYSING A MODEL

A model is only as good as the information it contains. An incomplete or incorrectly structured model, or a model containing erroneous property values, can cause more serious and expensive problems later in the development process. Therefore it is important to get the model right. Task Modeler provides mechanisms for ensuring the accuracy of a model, as early in the development process as possible.

3.1 Visualization

To ensure property values are accurate and complete, Task Modeler can visually show them on the model. Property values can be viewed either as symbols, colours, label contents, emphasis and network lines. See figure 2.

Figure 2: Visualizing properties as symbols, colours, and network lines

Recent versions have improved these visualization facilities, allowing the model to talk-back to the user. This provides a powerful way to ensure the model is fully and accurately documented. It also highlights hotspots and trends within the model, and areas that might need further work and analysis.

3.2 Managing a Complex Model

Typically the models can quickly become large and unmanageable, making them difficult to use effectively. Task Modeler provides facilities for selectively hiding and showing parts of the hierarchy, allowing the user to focus on just the area of interest.

Task Modeler allows particularly large models to be split across multiple files, which are linked in a variety of ways. Task Modeler can then show the entire model, by visually *embedding* the linked models within their linking model, allowing the model to still be viewed, analysed and used as a whole.

3.3 Validation

Version 5 of Task Modeler provides embedded model-specific validation, by presenting a list of model problems on the analyst's request. This further ensures that the structure and content of the model are complete and accurate.

4. REUSING MODELS

Along with ensuring that a model is accurate, it is important to understand where it fits in the overall development process. The resulting models are not the end of the work, and are used as inputs to the next phase of development, or as a working document that is constantly referenced and updated as development continues.

As the model continues to be central to development, Task Modeler provides facilities for using and reusing the model content. Models can be exported in a variety of ways, for example use case models can be exported as HTML Use Case reports, which document the relationships between use cases, requirements and development line items, or as a simpler development plan to use as development progresses.

5. REFERENCES

- [1] IBM Task Modeler Overview, <http://www.alphaworks.ibm.com/tech/taskmodeler>
- [2] DITA (Darwin Information Typing Architecture), <http://dita.xml.org/>
- [3] Annett, J. Hierarchical Task Analysis. In *The Handbook of Task Analysis for Human-Computer Interaction* Lawrence Erlbaum, New Jersey, 2004.
- [4] Bird, C. and Farmer, M, *Information Architecture with IBM Task Modeler*, HCI 2007