CS405 Introduction to Empirical Modelling
Lab 5: Graphical representation of dependencies

This lab session will introduce you to ways of representing, in a graphical form, the dependencies found in an EM model. Find the material for this lab session in `/dcs/emp/empublic/teaching/cs405-2006/lab5/`.

1 The Dependency Modelling Tool (DMT)

The DMT is a Java program that uses acyclic graphs to visualise the dependency present in a model. The DMT can be started by running:

```
java -jar /dcs/emp/empublic/projects/dmtWong2003/dmt075.jar
```

Exploring a simple model with the DMT

As an introduction to the DMT we will examine a simple jugs model.

Task 1

Input the definitions from the file `dmtJugs.eden`. Paste the contents of this file into the Script Input Window (choose *Input Window...* from the *Script* menu) and choose 'Accept'. This will create a dependency graph in the main window. Reorganise the nodes to explore the dependencies present in the model.

There are several points of interest with the graph that is displayed:

- The DMT is designed to be used for both the exploration and the creation of models. Nodes can be moved by clicking with the left mouse button. Through appropriate organisation of this graph the model's dependency structure can be exposed.
- Look at the dependency structure of the model. Note how `contentA` is linked to two separate parts of the model.
- How are functions in the model represented? (Look at the `max` function).
- Investigate some of the observables that are not connected to the main dependency graph. What do they represent in the model?
- Notice that some observables in the DMT are presented with green nodes. These correspond to observables with the value @ – the Eden representation of undefined.

  Why might the DMT be representing these observables as @ when they are defined in tkeden? What's special about these nodes?

  Note: This highlights a limitation in the current version of the DMT.
- `menu` is an isolated definition (and so no edges linked to it). In the model SCOUT definitions are dependent on this observable – note that only eden definitions can be loaded into the DMT.

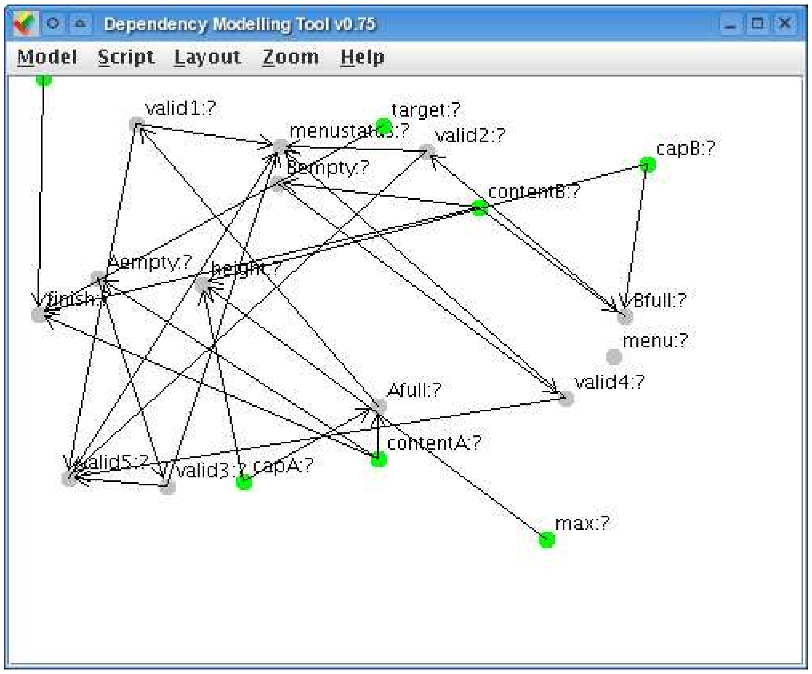

Figure 1: The Dependency Modelling Tool

2 The graph model

We will now switch to the tkeden tool. The graph model can be loaded by including the file `run.eden`. If you look at this script you will see that it is including the `graph.eden` script, creating a new graph and adding it to a SCOUT screen. It also adds two example nodes to the graph and creates a link between them.

The model is based around two key observables `graph_nodes` and `graph_links`. The first is a list of lists (ordered 4-tuples) representing the nodes on the graph, the second a list of lists (ordered 3-tuples) representing connections or links between the nodes.

For example:

```
%eden

nodes = [["x",[240,330],"node x","red"],["y",[100,350],"node y","red"]];
## There are two nodes "x" and "y" which have positions [240,330] and
## [100,350]. They are represented on the graph as red nodes with the
## respective labels "node x" and "node y".

links = [["x","y","black"]];
## There is a link from x to y, represented by a black line with an arrowhead.
```

Nodes can be moved around the graph display using the left mouse button.

Task 2

Experiment with the addition and removal of both nodes and links to the graph.

3 Using the EDEN symbol table

Various functions exist in EDEN notation that enable you to query the symbol table and find out about the definition and value of observables. You can find out about an observable with the `symboldetail` function:

```
symboldetail(string): returns a list of information about the named symbol:
  [name, type, text, targets, sources, master]
  where type is one of "var", "formula", "proc", "procmacro", "func",
  "builtin", "Real-func", "C-func".

symboldetail(pointer): returns a list of information about the named symbol.
```

4 Build your own DMT

The tasks in this laboratory have two purposes: to develop your modelling skills and to understand the benefits of representing and constructing models in a graphical form. By using the DMT you should have realised the benefits of displaying dependencies in a graphical form, but also encountered the difficulties of using a tool that is separate from the actual modelling environment. The following tasks are open-ended exercises that enable you to build a general dependency modelling tool, similar to the DMT, which is inside the `tkeden` environment.

Task 3

Create an observable called `watches` that is a list of strings containing observable names. Write a trigger on this observable such that when it changes it updates the `graph_nodes` and `graph_links` observables so that a graph of the dependencies is displayed on the screen (similar to the DMT).

Once you have completed exercise 3 you should be able to add observables to the `watches` list and the graph model will be updated. Ensure that positions are retained wherever possible and that deleted nodes are removed.

Task 4

Using SCOUT, create a textbox and a button which, when clicked, adds the observable in the textbox to the `watches` list.

To complete exercise 4 you will need to refamiliarise yourself with creating buttons in SCOUT, and also learn about textboxes in SCOUT (from either the website or the quick guide in tkeden).

Now you should have a simple DMT that allows you to explore any model. Load in a model from a previous lab and explore the dependency structure. The next section suggests some possible extensions to your DMT model to improve the visualisation and exploration of models.

5 Extending the DMT

The graph model contains observables which are updated on particular mouse actions. When the mouse hovers over a node, the observable `graph_nodeMouseOver` is set to the name of the node. When a mouse button is clicked (on a particular node), the observables `graph_nodeClicked` and `graph_buttonClicked` are set to the name of the node and the mouse button respectively.

Task 5

Add a trigger such that when a node is clicked with the right button, nodes and links are added to the graph for every observable dependent on the clicked observable.

Task 6

Add a trigger such that when a node is clicked with the middle button then it is removed from the `watches` list (and hence no longer displayed).

Hint: Adding to and removing from the `watches` list could be made simpler by adding two procedures that add or remove an observable name to the list, and also deal with difficult circumstances like an unknown observable name or a duplicate observable name.

Task 7

If you look back at the DMT, you'll notice that when you hover over a node some information about that observables relationships within the model is displayed. Replicate this functionality in your Eden model. The colour of the node should alter as the mouse pointer hovers over it.

6 Summary

You should by working through this lab sheet **and by additional study**:

- be able to analyse and build models using a graphical representation of dependency;
- have learnt how the Dependency Modelling Tool works and be able to highlight its deficiencies;
- know how to access the EDEN symbol table;
- have further developed your knowledge of Scout; and,
- have learnt how to adapt existing models for other purposes.

7 Revision history

This lab sheet was originally written for CS405 in November 2005 by Antony Harfield, Russell Boyatt and Charlie Care. Last updated for CS405 in November 2006.