CS405 Introduction to Empirical Modelling
Lab 7: Distributed modelling with dtkeden

Empirical Modelling Research Group
http://www.dcs.warwick.ac.uk/modelling/

Introduction

In the previous lab sessions you have been using the `tkeden` tool as an individual person modelling on an individual machine. This lab explores the potential for distributed models and distributed modellers. The `dtkeden` tool, the distributed version of `tkeden`, can be used for this purpose.

The `dtkeden` tool has a number of potential uses. The two that we shall mainly be concerned with are:

- building a distributed model.
- building a model collaboratively.

The aim of the following tasks is to familiarise you with `dtkeden` and the concept of distributed models. You will be pleased to know that we are returning to the [now famous] JUGS model to demonstrate the distributed features of `dtkeden`.

You should recall that JUGS was originally an educational model designed to introduce pupils to the idea of the "highest common factor of m and n" by allowing them to fill, empty and pour between jugs of integral capacities m and n. Only one pupil was involved in the interaction with the model. This lab addresses a distributed variant of the model ("Distributed JUGS") in which each pupil owns one jug, and pouring involves transferring liquid from one pupil's jug to another's.

1 The objective

The objective of the lab is to set up the environment to support the interaction for Distributed JUGS. This environment allows two or more people to cooperate in building the Distributed JUGS model, and then enables them to interact through selectively redefining observables both locally and globally. Specifically: each person has a jug (of some capacity) which they can fill or empty. Each person may also pour their own jug into exactly one other person's jug. For example, if there are three people, then the first person can pour into the second person's jug, the second into the third, and the third into the first.

Task 1

Get into a small group of two or three people.

2 Starting dtkeden

The `dtkeden` tool requires more preparation than `tkeden` and must be supplied with some command line options. When you run `dtkeden` you must specify whether you wish to start the tool in *server* or *client* mode. In a typical setup there should be one server, and one or more clients. The server should always be started first. A channel on which clients can connect must be specified.

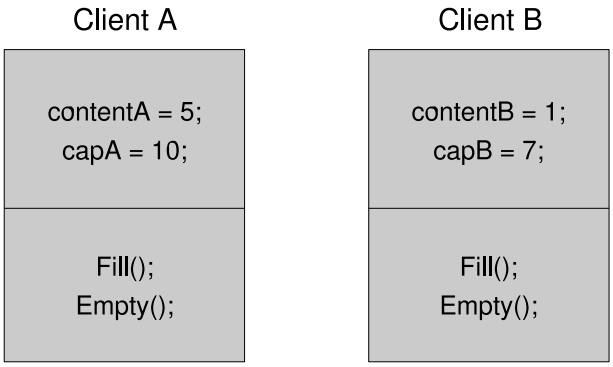

Figure 1: Executing definitions in a local context

Task 2

One person in the group should `ssh` into `joshua` and set up a `dtkeden` server with the command:
`dtkeden -s -c 55 -e 'EveryOneAllowed=0;'`
where `55` is a channel number between 0 and 99. By default, `dtkeden` runs in the 'Normal' mode (see *The Not So Quick Guide to dtkeden* for details).The `-e` option assigns the value zero to the observable `EveryOneAllowed` on startup. This is required when making use of LSD permissions in the 'Normal' mode.

Once the server has been set up then any number of clients may connect to the server by using `dtkeden` in client mode and specifying the hostname of the server (as well as the channel to connect on). Each client must supply an *agent name* which uniquely identifies the client.

Task 3

Each person (including the person who setup the server) should run a `dtkeden` client that connects to the server:
`dtkeden -a -h joshua -c 55 -w jugA`
where `55` is a channel number on the server and `jugA` is a name that uniquely identifies the client.

3 Collaborative model building

The `dtkeden` tool can be used in the same way as `tkeden` to build a model. There is a button called 'Accept' in the input window which is used for executing scripts on the client. A definition of the observable `x` interpreted in this way is said to define the value of `x` in the *local context*. The main difference between `tkeden` and `dtkeden` in client-mode is that there is another button called 'Send' in the latter. This button sends the contents of the input box to the server for execution. A definition of the observable `x` interpreted in this way is said to define the value of `x` in the *global context*. In the following task, you will be creating your own models in parallel and each will be on your own client so you only need to use the 'Accept' button. In this way, all observables will be defined in the *local context* as shown in Figure 1.

Task 4

Each person should build a single jug model in the local context of their `dtkeden` client (i.e. using only the 'Accept' button). Start by creating observables for 'content' and 'capacity', and link them to a simple SCOUT visualisation. Try to avoid using the same names for observables. You can make use of your answers to lab 3 if you wish.

Before continuing, be sure that you can change the content of your jug and that the change is reflected in your SCOUT visualisation.

4 Building a distributed model using the LSD notation

You should already be familiar with LSD accounts from the lectures. The `%lsd` notation in `dtkeden` uses the concepts of *oracles* and *handles* to specify what observables an agent is privileged to observe and change in the global context. This enables us to model communication between agents, as represented by clients.

A `handle` is an observable in the global context that an agent has permission to change. An agent (client) always has permission to change an observable in its local context (using the 'Accept' button). If agent A has the `handle` on contentA then he has permission to change it in a global context (using the 'Send' button). If you try sending any redefinitions before specifying the permissions then you will find that nothing happens (just try sending `contentA=5;` in one client and querying `contentA` in another, and it should be undefined).

An `oracle` is an observable in the global context that an agent can respond to (or is aware of). An agent is always aware of the observables in its local context. If agent A has an `oracle` on contentA then if any changes are made to `contentA` in a global context (using the 'Send' button), then `contentA` will be updated in his local context.

If we wish to give agent A the permission to change `contentA` in a global context, and then make agent B aware of changes to `contentA` then we can use the following syntax:

```
%lsd
agent A
    handle contentA
agent B
    oracle contentA
```

This definition must be executed on the server using the %lsd notation. When these permissions have been set, if client A types in the input box `contentA = 6;` and presses 'Send' then client B can query the observable `contentA` locally to find the current value (as set in the global context). Note that the input is not consumed when the 'Send' button is pressed; in typical use, client A should also press the 'Accept' button to update the value of `contentA` in their local context (otherwise the local and global values of the observable are inconsistent, as is sometimes appropriate!). Figure 2 shows the result of sending and accepting `contentA = 6;` in client A with the above LSD permissions.

To remove any of the LSD permissions:

```
%lsd
agent A
    remove handle contentA
```

You can choose the 'View LSD Description' item from the 'View' menu on the `dtkeden` server to check the current status of your LSD description.

Task 5

Add handles and oracles for each of your agents so that each agent is aware of the content and capacity of each jug in your group. Only one agent should be able to change each jug content and capacity (i.e. agent A should not be able to change the content of agent B's jug). It might be a good idea to add other SCOUT windows to view the contents of other agent's jugs.

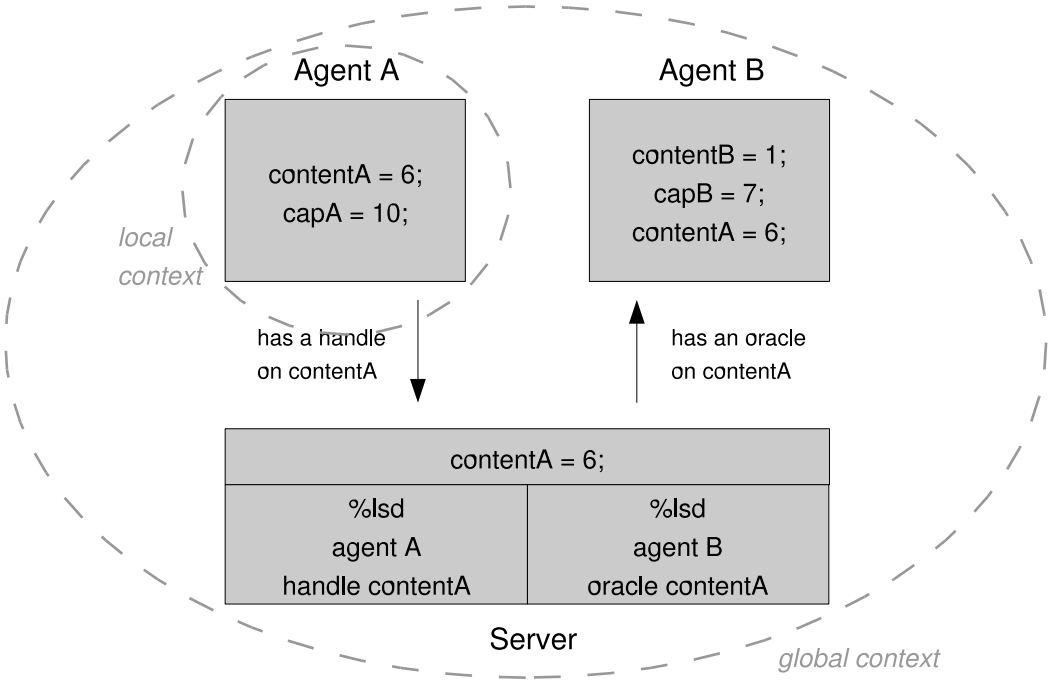

Figure 2: Executing definitions in a global context from client A

5 Communicating within scripts

The last task introduced you to the 'Send' button which allows you to execute scripts in a global context. As your distributed model becomes more advanced, you might want to send definitions to the global context from within your scripts.

The equivalent of using the 'Send' button on the client is using the `sendServer(servername,script)` procedure where `script` is a string containing the script. As we only have one server, the servername can be set to an empty string. For example: `sendServer("", "contentA = 5;");`.

Task 6

Add oracles and handles so that an agent can update another jug other than their own. Each modeller should now be able to write procedures to pour from their jug to another modeller's jug.

If you have time, try to implement the full repertoire of interactions required for the Distributed JUGS model and see if you can solve the problem of achieving a specified target collaboratively.

Summary

You have now been introduced to distributed modelling. In this lab session you should have learnt how to:

- start a dtkeden server and clients
- build a model collaboratively
- create distributed models using the lsd notation to communicate between agents

Revision history

This worksheet was written by Antony Harfield for CS405 Introduction to Empirical Modelling.