Notes on dtkeden

Introduction

The dtkeden tool is an extension of the tkeden tool with the addition of support for distributed modelling. Built on network communication over TCP/IP, dtkeden is based on a client/server configuration and provides a multi-agent, experiential modelling environment. Within this environment, each modeller has their own personal modelling environment similar to that supplied by tkeden, together with network communication via a server. Each dtkeden client is identified by a login name, and communication from this client to the server is attributed to a *virtual agent* with the appropriate login name. Virtual agency brings some of the advantages of object-oriented classes to Empirical Modelling.

The qualities of dtkeden as a modelling environment can only be realised by following good practice in its deployment. In interpreting interaction in the dtkeden environment, it is vital to conceive the representation of states and transitions within tkeden in an idealised form, viz. as based on the use of definitive scripts specified using Eden definitions and functions, together with actions that can be regarded as redefinitions within a script. This presumes that the modelling activity at each network node is in spirit close to that discussed in connection with the Abstract Definitive Machine (ADM), and that communication is essentially based upon transmitting definitions from one modeller to another. Note that, in practice, just as tkeden can be abused as a purely procedural or hybrid procedural/definitive programming language, so also can dtkeden be abused so that issues of concurrency and synchronisation are at least as complex and difficult to address as they are in traditional concurrent programming environments. In particular, communication between nodes can involve the transmission of any fragment of valid tkeden input, and may not be confined to a block of redefinitions. The discussion of different modelling modes below makes sense only with respect to disciplined use of dtkeden in which the principles of modelling with the ADM are as far as possible respected.

Conceptually, the modeller who uses the server's modelling environment as *superagent* is modelling/observing the interactions amongst the internal agents in the entire system from an external observer's viewpoint. The modeller who uses a client as a standard *agent* is modelling/observing part of the system from an internal agent's perspective. There are many ways in which the external observer and the internal agents can construe their interaction. To accommodate this, there are four different modes in which the dtkeden client-server environment can be configured. In typical use, it is to be expected that the mode is chosen to suit the modelling context at the outset, and is subsequently unchanged. The four possible modes of interaction are **Normal**, **Broadcast**, **Interference** and **Private**. The nature of the construal of multi-agent interaction to which each informally corresponds, and the principal mechanisms used to support this construal, are as follows:

1. **Normal Mode** (the default mode)

   Normal mode is associated with 'normal' interaction in the everyday objective world. All the agents are deemed to have a common perception of what the shared observables are, but each may have a different perspective when interacting with them. For instance, all attendees at the CS405 module can be viewed as having 'the same' perception of the key features of the lecture room CS101, such as the location of the door, the chairs and the lights, but only some of them are currently in a position to see whether or not the door is open or the lights are on, and only some will be in a position to open the door or move a chair. The important characteristic of this construal of a situation is that all observables are deemed objective, in the sense that, subject to having access to their values, all agents will ascribe the same values to them. For instance (in a 'normal' construal of reality), whether we can see the door to CS102 or not, we all believe it to be currently either locked or unlocked, and, were we all in a position to determine whether the door was or was not locked, we would all come to the same conclusion about its status. It is then appropriate to consider that each observable has an authentic value, even though - when the specific context of the modelling scenario is taken into account - we are not all able to see the same observables, and may have different privileges to change their values.

   In Normal mode, observables can be classified using the LSD notation at the dtkeden server. For this purpose, the LSD concepts (*oracle* and *handle*) are invoked. An *oracle* refers to the privilege of an agent to observe the authentic value of a variable (considered as an *observable*), and a *handle* refers to an agent's privilege to change the authentic value of a variable (observable). A special automated agent, the *LSD Agent*, is located at the server in dtkeden. This agent is responsible for the management of access privileges for variables. When a request to

access a variable (for reference or for alteration) comes from a client to the server, the LSD Agent will check the privilege of the sender for their access permission to that variable. At the same time, if a change to an observable is made, the LSD Agent will propagate this change to all other agents who are authorised to receive it.

In modelling in Normal mode in dtkeden, the authentic values of observables are recorded at the server, and it is these values that are propagated to all agents for which they are oracles, and that are affected by communications from client agents for whom they are handles. In all manually invoked communication from a client, the modeller has discretion over the precise set of agents to whom redefinitions of observables are directed. Sending a value of an observable directly to another client resembles communicating your private notion of its current value; sending it to the server resembles a request to change its current authentic value. The LSD Agent at the server is responsible for determining how requests of this nature affect the authentic values of observables and their perception by other agents. For instance, a redefinition of x followed by a 'Send[Server]' at a client that has x as an handle will update the authentic value of x at the server, whilst a redefinition of the authentic value of x at the server will update the value of x at all clients for which x is an oracle. Note that the authentic value of x can be redefined in several ways: it may be the result of a client updating x as a handle and communicating this change, or of a redefinition of the observable x followed by a 'Send' or 'Accept' at the server. Note also that there is scope within Normal mode for each client agent to have a private notion of the current value of x, such as might result from redefinition of x by the client followed by an 'Accept'. If x is an oracle for the client, this private value is subject to be overwritten by redefinition of the authentic value of x at the server. Private values for observables of this nature might correspond to values of observables 'as remembered'.

The mechanisms by which the modeller manually communicate redefinitions are complemented by communication procedures ('sendServer' and 'sendClient') that can be invoked automatically at a client node. Redefinitions sent directly to the server (by sendServer()) update the authentic value and are then propagated to all clients that have the appropriate oracles. Redefinitions sent from one client to another (by sendClient()) are conceptually different; they do not affect the authentic values of variables but represent the private communication of values perceived by one client to another client.

2. **Broadcast Mode**

   Broadcast mode can be seen as a special case of the scenario for 'normal' concurrent interaction. In this mode, any communication from a client is automatically propagated to all other clients (contrast the way in which a client selects particular targets for their communication in Normal mode). Each client has access to those observables that are broadcast by other clients, together with those private observables that the client has defined. Broadcast mode supplies an open environment for multi-agent modelling such as is needed for a multi-user game. A typical application for such an environment might be modelling a card game such as whist, in which at any stage there is a pool of public cards, and each player sees these cards together with the cards in their private hand.

3. **Interference Mode**

   Interference mode is associated with an understanding of agent interaction that can be seen in one sense as more primitive than that associated with normal everyday interaction (in that it does not assume a shared objective world), and in another sense as more sophisticated (in that it admits explicit reference to personal viewpoints and perceptions). In Normal mode, the external observer can be seen as maintaining the authentic values of objectively defined observables about which the internal agents have a shared understanding. While it is possible in Normal mode for clients to have private values (such as might represent the values of observables as remembered or last recorded by them), these values cannot be referenced by the server: from the perspective of the external observer, the symbol 'x' refers to the authentic value of the observable x as recorded at the server, and any private value associated with x at a client is inaccessible. In Interference mode, in contrast, the external observer can be seen as acknowledging the essentially private nature of the observation of each client and trying to bring coherence to all their independent observations. In this mode, the external observer plays the archetypal role of 'creator of the concurrent system', and has complete discretion over whether and how the individual perceptions of the internal agents can be integrated so as to shape a reality.

   Virtual agency is the syntactic device by which private values of observables are associated with clients in the server environment. The observable a as viewed by client X is referred to as X_a. A redefinition of an observable that is communicated to the server from a client is presented to the modeller at the server in the input window in a form which makes it convenient for the external observer to shape concurrent interaction to suit their construal. For instance, the redefinition

```
a is b+c;
```

communicated from the client X will appear in the input window at the server as:

```
>>X
a is b+c;
>>
```

If this is processed without any intervention by the modeller (using Accept or Send), it will be associated with the redefinition:

```
X_a is X_b + X_c;
```

so that in effect the server is recording/transmitting information about X's private perception of the relationship between the observables a, b and c. The modeller can alternatively conveniently edit the input

```
>>X
a is b+c;
>>
```

to the form

```
>>
a is b+c;
>>
```

and thereby interpret the action of the agent X as redefining authentic value of the observable a.

Since Interference mode allows the external observer to act as a superagent to directly interfere with the interactions between agents, or between each client and the server, it is best suited to applications in which negotiation of concepts or of the values of observables is involved. These are characteristic of concurrent design activity, where there are many autonomous agents, each of whom has a different perspective on the objective system or product under development.

4. **Private Mode**

In the private mode, each client has their own private values of observables and a "private" communication channel to the server. The conventions for referring to a client's private perception of an observable are similar to those in Interference mode, and make use of virtual agency. In contrast to the Interference mode, a communication to the server takes direct effect on the appropriate observables as recorded by the server - the modeller acting as an external observer does not exercise discretion over its interpretation. Moreover, in contrast to the broadcast mode, no request will be propagated to the other clients. This mode is suited to activity in which the modelling has been modularised into independent components, or where other forms of independent interaction - such as the private interactions of a class of pupils with the same model - are being monitored or drawn together.

The choice of mode in which to conduct distributed modelling with `dtkeden` is determined by the nature of the external observer's construal. It is clear that this choice is pragmatic in character and that the justification for making one assumption about the nature of agent interaction rather than another is informal and empirically based. The constructs that support one mode of interaction are not necessarily available in another mode, and - if they available - may have a different significance. For example, the use of the LSD notation is confined to Normal Mode, whilst virtual agency is a concept that can be used in any mode even where it refers to agents other than the modelling agents sited at clients. More technical discussion of the practical use of `dtkeden` follows.

Considerations governing modes and behaviours in `dtkeden`

In distinguishing the four modes of interaction in `dtkeden`, it is helpful to consider various issues that differentiate interaction as it is supported by the interpreter. Relevant questions include:

- Can the external observer refer to the private observables of internal agents?
- Does the server automatically accept the input sent from clients?
- Does input from clients get transmitted automatically to other clients via the server?
- How are the default conventions EveryOneAllowed set when observables are introduced?
- Are communications from clients automatically accepted by the server?
- When extra clients are introduced, what observables are in existence?
- How are client instantiation and observable introduction synchronised?
- Is there visualisation associated with observables?
- Does communication from clients get registered as potential input at the server, rather than automatically accepted?

In broad terms, we can consider the role of the external observer in Normal mode as relatively passive, and as relating to singular conditions that are encountered in normal shared reality. The external observer may be called upon to frame unusual changes in the environment for interaction (e.g. such as might simulate some engineering works in the lecture room CS102) or to arbitrate where there are singular interactions (e.g. as when two people are in conflict about opening / shutting the door), but this does not involve a true negotiation of reality. The role of the server is to mediate a shared reality, and this may involve synthesising both manual and automatic communication. Broadcast mode can be seen as a specialisation of interaction in Normal mode, where the possibility of agents not observing changes to certain aspects of shared state, or of making partial communications of state, is not admitted.

In contrast, Interference mode is concerned with more radical and active intervention on the part of the external observer in shaping what is deemed to be inter-subjective. It requires activity that is dominated by human intervention and the exercising of intelligence on the part of the modellers: the potential for automation is then more limited, and communication is predominantly hand-crafted. Private mode is a specialisation of communication within this framework, in which some of the external observer's discretion over what is communicated from agents is removed, but the privacy of these communications is preserved. In this role, the external observer can play a powerful role in organising and interpreting the private interactions of client modellers, but is more limited in the extent to which they can interfere with them.

Matters arising

- The current implementation of `dtkeden` makes provision in the symbol table for an observable to be classified as an LSD state observable: completing this implementation would involve ensuring that observables were introduced into and deleted from the modelling environment as and when agents themselves entered and withdrew from the concurrent interaction. Implementing this feature would involve the use of the Eden forget() procedure to manage the symbol table appropriately. This would enable `dtkeden` to emulate more of the functionality of the ADM.
- Virtual agency has an essential role in `dtkeden` as a way of distinguishing the different agent views of what is conceptually the same observable. In this application of virtual agents, nesting of agents is not appropriate. As the generation of an array of buttons above illustrates, virtual agency can also be used to create multiple instances of data, and been incorporated into `tkeden` for application in this role. In this context, nesting of agents might be a welcome feature. Because of the clumsy procedural nature of virtual agency as it is currently implemented in `tkeden`, it has proved difficult to use it effectively and intelligibly (see e.g. the way that grids of Scout buttons are generated in timetableKeen2000), and other methods of generating multiple instances of data are generally to be preferred.
- Because Empirical Modelling is rooted in personal experience, and objectivity is derived through classifying experience, it might seem that distributed EM should be based on a peer-to-peer rather than a client-server architecture. A practical motivation for adopting such an architecture is that the computational load on the server in `dtkeden` can be exceptionally heavy. It may well be that there is a place for a peer-to-peer tool to support distributed EM, but the client-server framework is itself quite well-suited to the concept of concurrency as created in the mind of the external observer, and the server has a key role in supporting the external observer's construal.