Figure 2

Lecture W2: The LSD Notation for Agent Specification

W4.0 Background and Motivation

The LSD notation was first developed by Beynon in collaboration with Mark Norris of British Telecom in 1986. The original influence over the design of the LSD notation was SDL []. (The terminology of LSD first had "process" rather than "agent" for this reason.) The elaboration of the design and identification of the agent-oriented non-operational nature of the specification was carried out by Mike Slade, as described in his MSc thesis (1989). One of the main objectives of Slade's work, sponsored by British Telecom, was the development of software for animating from LSD specifications; this led to the design and implementation of the ADM. The challenges of graphical animation from the ADM have been met only recently, through the work of Simon Yung in linking the ADM and EDEN.

LSD specification and ADM simulation arguably address issues rather different from SDL. Some points of particular relevance are:

- an LSD specification is oriented towards understanding the interaction between agents in a system in a spirit that is closer to the concerns of the designer of a user-computer interface than to abstract formal specification. The idea is to formalise what an agent observes of a system, and how it can affect the state of the system by performing actions such as an experimentor might. This relates to concerns such as human factors in system design (e.g. what does the user need to know, need to be able to perceive, and what would be the implications of imperfect knowledge or loss of capability), to system models that have explanatory power (e.g. enabling the designer to relate the system behaviour to characteristics of the components that can be verified by experiment) and to identifying the fundamental assumptions upon which satisfactory perforrmance of a complex system depends (e.g. how fast and reliable does the communication and response between agents have to be).

- modelling and simulation associated with LSD specification is in principle intimately connected with physical experiment and observation. (Sometimes only "in principle", because it can be difficult to develop models that relate closely to the external activities they represent in all respects, as will be illustrated in the cricket simulation.)

  Most methods of computer modelling put the emphasis on trying to specify / mimic the global behaviour of a system without explicitly modelling the relationship between its component parts. In formal specification, independence of the physical model is regarded as desirable, on the assumption that understanding the behaviour of a system should precede the construction of its components. Most simulation tools are based on random generation of inputs etc that can give a useful view of the overall behaviour of a discrete-event system, but will generally fail to account for the essential mechanisms that cause particular sequences of events, and be of little help in matters of detail. The trend towards using qualitative models (as in naive physics) also tends to divorce computer-based modelling from physical principles.

  Our approach reflects a different outlook: we consider that complex specifications for systems cannot be constructed without building and experimenting on components, that simulation that is based on a statistical approach is too coarse a tool for many applications, and that a computer model in physics and engineering must be based on a strong relationship to the physical entity it represents. Until these concerns are addressed, the scientific integrity of computer use in many applications is in question.

Figure 2

- the analysis of agent characteristics in an LSD specification does not lead directly to an executable model. An operational interpretation is derived by introducing additional assumptions. This means that the LSD notation has no formal operational semantics, in the computer science sense. LSD is concerned primarily with requirements analysis, and with the identification of system structure that precedes circumscription of its actual / intended behaviour. An important function of LSD is to raise an unusually rich set of questions concerned with requirements that would be difficult to identify without thinking about agents and their interaction.

W4.1. Form of the LSD specification of an Agent

The LSD specification of an agent comprises 4 kinds of variable:

oracle - a variable to which it responds
state - a variable that it owns
handle - a variable conditionally under its control
derivate - an indivisibly coupled stimulus-response relation

As an illustrative example, consider the following specification of the station_master agent, taken from the railway station animation:

```
agent sm() {                                    // The station master:
state       (time) tarrive = |Time|;            // registers time of arrival
            (bool) can_move = false;            // determines whether driver can start engine
            (bool) whistle = false;             // controls the whistle
            (bool) whistled = false;            // remembers whether he has blown the whistle
            (bool) sm_flag = false;             // controls the flag
            (bool) sm_raised_flag = false;      // remembers whether he has raised the flag
oracle      (time) Limit, Time;                 // knows the time to elapse before departure due
            (bool) guard_raised_flag;           // knows whether the guard has raised his flag
            (bool) driver_ready;                // knows the driver is ready
            (bool) around[d]; (d = 1 .. number_of_doors)
                        // knows whether there's anybody around doorway
            (bool) door_open[d]; (d = 1 .. number_of_doors)   // the doors status
handle      (bool) can_move, whistle, whistled, sm_flag, sm_raised_flag;
            (bool) door_open[d]; (d = 1 .. number_of_doors) // partially controls the doors
derivate    (bool) ready = ⋀ (¬door_open[d]) | d = 1 .. number_of_doors);
                        // monitors whether all doors are shut
            (bool) timeout = (Time – tarrive) > Limit; // monitors whether departure due
privilege   door_open[d] ^ ¬around[d] –> door_open[d] = false; (d = 1 .. number_of_doors)
            ready ^ timeout ^ ¬whistled
                        –> whistle = true; whistled = true; guard(); whistle = false;
            ready ^ whistled ^ ¬sm_raised_flag –> sm_flag = true; sm_raised_flag = true;
            sm_flag ^ guard_raised_flag –> sm_flag = false;
            ready ^ guard_raised_flag ^ driver_ready ^ engaged ^ ¬can_move
                                                    –> can_move = true;
}
```

Matters for discussion

*Is variable a good name for these features of the LSD specification? In a sense, they are all observations, but in some instances they are observations associated with **different** agents of what are conceptually the **same** entity. Adopt v-observation to mean observation designated by identifier v.*

This is why the same identifier occurs many times in an LSD specification

Figure 2

e.g.	measSpeed	**state** for speed_transducer
		oracle for throttle_manager
		oracle for the driver
	cruiseStts	**state** for cruise_cutout
		oracle for the driver
		oracle for the throttle_manager

In simulation derived from an LSD specification, identifier instances will in general be represented by different variables.

The system designer determines the observations recorded in the LSD specification. These observations are what the designer would record and subject to experiment in relating the (pre-act-of-faith) behaviour of the system to the (post-act-of-faith) reliable activity of the components.

Classification of variables associated with an agent

states

There are generally certain observations associated with an agent in such a way that were the agent instance to disappear they would also disappear. E.g. accel, windF, actSpeed, ... are meaningful only whilst there is a vehicle agent. Such observations are identified as state variables of the agent. They are the variables bound to the agent.

A state variable v is the primary source of all v-observations: at any time, all v-observations are associated with at most one state variable.

A state variable v is a record of the "authentic value" of v-observations. It's value is not necessarily the value perceived by the agent to which it is bound. That is to say, a variable may be both an oracle and a state to the same agent, as in "The time as recorded by my watch, or my bank balance".

Note that those agents which serve primarily as observers and actors in a system tend to have few state variables. E.g. there are no state variables associated with the driver agent.

consts

Constants are a particular kind of state variable, whose value is not subject to change through actions of agents in the simulation e.g. the physical parameters of the vehicle: mass, windK, rollK.

In the VCCS as implemented, it would be possible to treat the parameters of the vehicle as subject to change. E.g. a simulation might take account of loading the vehicle, or express the relationship between the wind resistance and the vehicle profile.

oracles

In anthropomorphic terms: an **oracle** = value as perceived by an agent.
This admits the possibility that its value is inaccurate.

actSpeed is **state** in vehicle, ...

Is actSpeed an oracle to the speed_transducer? In what sense is the fact that a door is locked an oracle to the door user? Whether the door can be opened or not in no way depends on the user's perception of whether it is or isn't locked, nor is the effect of locking the door subject to delay.

handles

A handle is a variable that an agent can manipulate, subject to certain enabling conditions being met. A handle variable of an agent is not necessarily bound to the agent. E.g. the driver agent can act to change the state variable engineStts of the engine agent.

Handles allow direct action of one agent upon another that is outside the scope of an object-oriented paradigm (cf an object invokes a method to change its state).

Figure 2

When a v-handle and v-oracle co-exist, they together indicate communication between agents. When animating from a specification, appropriate assumptions about communication have to be made. For instance, the VCCS specification does not specify how the speed of the vehicle, as measured by the speed_transducer, is communicated to the throttle_manager. In animation, we might assume idealised communication, or refine the specification to include further details of the communication channel.

protocols

An agent's privileges to redefine handles make up its protocol.

Each privilege represents a possible action, or sequence of actions, that an agent can perform in appropriate circumstances. For example, if the engine is switched off, the driver can switch it on, and vice versa.

There is no fixed set of rules for interpreting agent protocols in operational terms. Relevant considerations include:

- Privileges are not obligations to act. Many possible alternative courses of action are represented in the driver protocol; for example, when the cruise controller is switched on, the driver can switch it off, request it to maintain the vehicle either at the specified cruise speed or at the current measured speed, or reset the specified cruise speed.

- Certain privileges express actions that must be executed promptly as soon as they are enabled. For instance, the cruise controller should not try to maintain the vehicle speed if once the driver touches the brake (see the cruise_cutout protocol).

- Animation from an LSD specification should reflect the external significance of agent actions. For instance, it would be unreasonable to expect the driver to switch the cruise controller on and off rapidly and repeatedly; not only is this is an improbable activity for the driver, but there will be an engineering constraint on how fast it is possible to switch between on and off modes.

W2.2. Principles behind the LSD specification for agents

W2.2.1 What the LSD specification addresses

Issues for the designer (to be addressed by an LSD specification):

- What are the key observations of a system that explain its behaviour?
- What are the agents in the system?
- How are the observations associated with agents?
- How does an agent register changes of state in its environment?
- In what circumstances does an agent have privileges to change state?
- What observations can an agent change?

In experiment, correlate values of observations o1 and o2.
Simplest case, as illustrated by Hookes' Law: "change o1 and observe o2".
Can still recognise FD where another agent is responsible for changing o1.
There is a semantic distinction between o1 and o2:
in changing o1, refer to o1 not as an observation but as state variable
in observe o2, regard o2 as an oracle.

[A spreadsheet user may continue to update though the spreadsheet has yet to be made consistent. In effect, if o1 is displayed value of cell1 and o2 is displayed value of cell2, then the relation o2 = f(o1) is guaranteed to hold only when the spreadsheet is completely up-to-date. This can be understood in terms of the above model as: o2 is an oracle to the spreadsheet agent. In observing o2, the spreadsheet agent does not simply consult the display, but (as when following a protocol for an experimental observation) waits until the spreadsheet is up-to-date.]

The use of derivates in animation from an LSD specification reflects presumed FDs and synchronisations in the view of the system designer. This interpretation may conflict with the concept of a derivate as an indivisible relationship as perceived by an agent.

[In a model of spreadsheet and spreadsheet user system, have to appeal to the protocol for observation used to

Figure 2

determine the value of a cell described above to account for the fact that the spreadsheet user perceives definitive relationships that in the view of the system designer are not universally valid. This is good illustration of how a v-oracle is to be interpreted with reference to subtle conventions of communication and observation. Notice also that the indivisibility of relationships between spreadsheet cells defined by formulae in the view of the spreadsheet user is here guaranteed because no other agent can interrupt the updating process - if this were not the case, the premise on which the user continues to enter data whilst the spreadsheet is still updating would be invalid.]

The usefulness of the concept of a derivate as an indivisible relationship as perceived by an agent is illustrated by the derivates used in the station_master agent in the train specification.

Important distinction between timeless experiments (cf Hookes' Law) and those in which analogue behaviour plays a role. In the latter, events and analogue variables are used, as in the VCCS.

LSD isn't intended to define an unambiguous behaviour for a concurrent system in itself. The interpretation of oracles, the ways in which privileges to act are exercised by agents, the nature of the relationship between observed stimulus and response, and the speed with which sequential steps in the protocol are executed are all subject to further examination before an animation can be constructed.

W2.4. Illustrative Examples

<Here refer to specifications given in detail in an Appendix:
to include (extracts from) LSD specifications
for the VCCS, the RSA, the Digital Watch
The Electronic Catflap and the Telephone>