Lecture 4: The LSD Notation for Domain Analysis and Description

Empirical Modelling activity creates artefacts and patterns of interaction with them involving different human and automated agents. Conceptually, this activity need never terminate - there are always new possibilities for interaction and extension to be explored. Anyone who has tried to understand an existing EM model faces the problem of identifying what characteristic interactions and interpretations the modeller had in mind, especially as some of the interactions that are explicit in the early stages of the model-building become hidden as the model reaches a more mature state. The LSD notation is conceived as a way of documenting interactions with and within an EM artefact developed in this fashion.

The original motivations for investigating EM were quite closely aligned to the core objectives of programming - the prescription of reliable processes with precise formal and operational interpretations. As a result, the first papers referring to LSD tend to be expressed in terms of concepts such as 'process', 'specification', 'variable', 'protocol' etc that suggest an environment for interaction that resembles the traditional setting for a program or a well-defined information processing system. Given the perspective on EM that has since emerged, this could be seen as a fundamental conceptual error. In some sense, it is *inappropriate* to adopt that position. Though EM activity begins in a context where the nature of the agency, the interactions and the environment are typically obscure, it can migrate towards a context in which the interpretations of these concepts become quite precisely determined (subject to discretion on the part of the modeller in respecting these interpretations, maintaining the integrity of the referent, and not making absurd redefinitions etc). In effect, what begins as an LSD account with agents and observables can dissolve into an LSD specification that relate processes and variables. For more discussion, see the lecture: *A Perspective on Concurrent Systems*. Viewed in this way, what might be seen as a "fundamental conceptual error" is more appropriately regarded as a critique of the idea that there is a satisfactory formal linguistic framework to capture what is going on in EM.

The variations of style within LSD accounts can also be seen as reflecting the different contexts in which they are to be interpreted rather than as evidence of confusion about the essential character of the notation itself. For instance, it is quite appropriate that the LSD account of the vehicle cruise control contains observables that are classified as "constants" and are given specific types, and that, in the original form drawn up by Ian Bridge in 1991, the handles and oracles of agents were grouped together to be classified as outputs and inputs for an agent interface. Such treatment suits the engineering context within which `cruisecontrolBridge1991` was conceived, and reflects the status of his account as a form of LSD "specification" of a "simulation".

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 the telecommunications Specification and Description Language SDL [004]. (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 what we now call *an LSD account* (at that time referred to as 'an LSD 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 accounts; this led to the design and implementation of the ADM. The challenges of graphical animation from the ADM were subsequently met through the work of Simon Yung in linking the ADM and EDEN. (Though something is lost in the translation process and these translation tools have not been much exercised in recent EM projects.)

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

- An LSD account 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 experimenter 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 performance 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 an LSD account 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 is illustrated e.g. by the simulation of the driver's protocols for interaction in the VCCS.)
- 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.
- The analysis of agent characteristics in an LSD account does not lead directly to an executable model. An operational interpretation is derived by introducing additional assumptions. (See [BNS88] and [BNOS90] for more discussion and illustration of these issues.) 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.

1. Framing an LSD account of an Agent

An LSD account of an agent comprises four kinds of observable:

state - a observable that it owns

oracle - a observable to which it responds

handle - a observable conditionally under its control

derivate - an indivisibly coupled stimulus-response relation

It also includes a **protocol** that identifies possible actions that the agent can perform to change the system state, subject to certain cues being given and certain enabling conditions being met. In setting

up a simulation from an account, instances of agents are introduced. The effect of agent action may be to invoke or delete other agent instances. The special variable `LIVE` can be used to specify when an agent instance is deleted.

As an illustrative example, consider the following account of the `station_master` agent, part of a railway station train arrival and departure simulation:

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*

protocol

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**

}

Identifying oracles, states, handles and derivates for the LSD agent should be viewed as classifying observables according to how they are being perceived by that agent. Where the same identifier appears in several different agents in an LSD account it refers to what is conceptually one and the same observable. This is illustrated in the LSD account of the vehicle cruise controller (as animated in the EDEN model `cruisecontrolBridge1991`):

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.

The **oracle**, **state**, **handle**, **derivate** characterisation reflects the status of the observable with respect to the agent. For instance, a handle resembles what we have previously termed an experimental parameter. Note that in an animation derived from an LSD account, it is in general necessary to introduce independent names for what the multiple references to a single identifier in the LSD account. This is in order to reflect the different ways in which the same observable is perceived by different agents.

2. More details of the classification of observables associated with an agent

States

There are generally certain observables 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 observables are identified as state observables of the agent. They are the observables bound to the agent.

A state observable v is a record of the "authentic value" of the observable v. The observable v can only be classified as a state for at most one agent, but may otherwise be represented in many different ways in LSD accounts. This reflects the diverse ways in which the same observable may be perceived by agents. Note in particular that the current value of the observable v is not necessarily the value perceived by the agent to which it is bound. That is to say, an observable may need to be identified - and distinguished - as 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 observables. E.g. there are no state observables associated with the driver agent.

Constants

Constants are a particular kind of state observable, 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 reflects the value of an observable as perceived by an agent.This admits the possibility that its value is inaccurate. By way of illustration, `actSpeed` is a **state** in vehicle, and is deemed to be an oracle to the Mspeed_transducer, though it is conceptually difficult to conceive how it could be anything other than authentic in its value. The driver's knowledge of the `cruiseSpeed`, as authentically recorded in the `control_panel`, may on the other hand be unreliable, and depend on depend on the attentiveness of the driver. The connection between observed and authentic values can be both complex and subtle. We might ask for instance whether, in a room user's privilege to open a door:

```
!door_locked ^ !door_open  ->  door_open = true
```

the fact that a door is locked is 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 an observable that an agent can manipulate, subject to certain enabling conditions being met. A handle observable of an agent is not necessarily bound to the agent. E.g. the `driver` agent can act to change the state observable `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).

An observable that occurs as a handle for one agent and an oracle for another, indicates a form of communication between agents. When animating an LSD account, 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 account so as 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 `control_panel` protocol).
- Animation from an LSD account 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.

3. Principles behind giving LSD accounts of agents

In giving an LSD account of agents, there is potentially a transition from a descriptive to a prescriptive stance. The human agent involved can accordingly be regarded both as a modeller and as a system designer. The modeller/designer determines the observables recorded in the LSD account. These observables are what the modeller would record and subject to experiment in relating the ("pre-theory") behaviour of the system to the ("post-theory") reliable activity of the components.

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

- What are the key observables of a system that explain its behaviour?
- What are the agents in the system?
- How are the observables 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 observables can an agent change?

In experiment, correlate values of observables o1 and o2.

Simplest case, as illustrated by Hooke's Law: "change o1 and observe o2".

Can still recognise a functional dependency where another agent is responsible for changing o1. There is a semantic distinction between o1 and o2:

in changing o1, regard o1 as a handle

in observing 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 account reflects presumed functional dependencies 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, we have to appeal to the protocol for observation used to 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 a good illustration of how an 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.

There is an important distinction between timeless experiments (cf Hookes' Law) and those in which analogue behaviour plays a role. In the latter, timers, events and analogue variables feature, as in the

VCCS (cf. *The ADM in computer-based Empirical Modelling*).

Note the importance of interpreting LSD with reference to an external context for interaction (cf. the semantics of a definitive script). LSD isn't intended to define an unambiguous behaviour for a concurrent system in the way that a formal specification does. 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 to be interpreted with reference to external counterparts that can be independently conceived and experienced (at least in the imagination of the modeller!). Observation and experiment directed at these issues has an essential role to play before a satisfactory animation can be constructed.

4. Illustrative Examples

For illustrative examples, refer to the LSD accounts given in the Appendix: the VCCS, the Railway Station Animation, the Digital Watch, the Electronic Catflap (in versions by Simon Yung and Ian Bridge) and the Telephone.

References

[BN87] WMB + MTN: Comparison of SDL and LSD

[BNOS90] WMB et al: Definitive Specification of Concurrent Systems

[BNS88] WMB et al: Definitions for modelling and simulating concurrent systems