The LSD Notation for Domain Analysis and Description

(formerly MSc lecture T2 by Meurig Beynon; revised November 2006 by Steve Russ)

Preamble

Beware: In many documents from the 1990s LSD is spoken of a 'system design'notation for the 'specification' of agents. It may, indeed, become that. But initially it will probably have a much more provisional status and is better regarded (we now think) as giving some analysis of an environment or domain which results in the identification of agents and a classification of the observables that are considered relevant to the kind of understanding that the modeller is seeking
.

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 [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 the description (or 'specification' as it was then known) 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 descriptions; this led to the design and implementation of the ADM. The challenges of graphical animation from the ADM have been met subsequently through the work of Simon Yung in linking the ADM and EDEN.

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

- an LSD description 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 animation associated with LSD description 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 description does not lead directly to an executable model. An operational interpretation is derived by introducing additional assumptions. (See [007] and [017] 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 domain analysis, and with the identification of system structure that precedes circumscription of its actual or intended behaviour. An important function of LSD is to raise an unusually rich set of questions concerned with understanding a domain that would be difficult to identify without thinking about agents and their interaction.

1. Form of the LSD account of an Agent

The LSD account of an agent comprises four kinds of variable:

state - an observable that the agent owns

oracle - an observable to which the agent responds

handle - an observable conditionally under the agent's control

derivate - an indivisibly coupled stimulus-response relation

It also includes a **protocol** that specifies the possible actions that the agent can perform to change the system state, subject to certain enabling conditions being met. In setting up an animation from an LSD description, instances of agents are introduced. The effect of agent action may be to invoke or delete other agents.

As an illustrative example, consider the following description 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*

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**
}

In the past the term 'variable' was used for the various components of an LSD account; they are all observables, but in some instances they are observables associated with different agents for what is conceptually the 'same thing'. This is why the same identifier may occur many times in an LSD specification:

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, identifier instances will in general be represented by different names.

The modeller determines the observables recorded in the LSD account.
These observables are what the modeller 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.

More now about 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 variables bound to the agent. A state observable v is the primary source of all observables using the name v at any time, all such observables are associated with at most one state observable.
A state observable v is a record of the "authentic value" of observable v.
Its 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 observables. 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, ...
Note that actSpeed is not deemed to be an oracle to the speed_transducer,
though it is referred to by a derivate. We might ask in a similar spirit
whether, in the privilege

!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 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).

When an observable occurs as handle for one agent and oracle for another, they together indicate communication between the 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 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.

2. Principles behind the LSD specification for agents

Issues for the modeller (to be addressed in an LSD account):

- 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, refer to o1 not as any observable but as state observable
> 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 specification reflects presumed functional dependencies and synchronisations in the view of the modeller. 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 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.
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.

Illustrative Examples

For illustrative examples, refer to the LSD accounts attached: the VCCS, the Railway Station animation. There are also LSD accounts for the Digital watch, the electronic cat flap and a telephone.

References

[004] W.M. Beynon and M.T. Norris Comparison of SDL and LSD
[017] W.M.Beynon et al. Definitive Specification of Concurrent Systems
[007] W.M.Beynon et al Definitions for modelling and simulating concurrent systems

(Numbers in square brackets refer to directories in /dcs/emp/empublic/papers.)