Lecture W1: Agent-oriented Modelling and Simulation for Discrete Event Systems

This lecture develops the idea of agent-oriented modelling as complementary to definitive representation of state. It also introduces the LSD notation and the ADM (Abstract Definitive Machine) to be discussed at greater length in later lectures. The focus is on discrete-event modelling, but the principles generalise to systems where continuous processes are involved, as the VCCS itself illustrates.

W1.0 Outline of lecture

1) Introduction

- Applications of discrete-event modelling:
  - convergence of programming and modelling perspectives
- Issues for discrete-event modelling
- Problems with current approaches

2) Background to our Approach

- Orientation: state-based, incremental model development
- Basic principles:
  - agent-oriented specification over definitive representations of state
- Programming as modelling
  - relationship to OOP: Simula and Actor models
  - advantages of our approach

3) Overview of our Techniques

- System state representation: the spreadsheet principle
- Agent-oriented specification using the LSD notation
- Operational limitations of LSD specification
- Animation of LSD specifications in the Abstract Definitive Machine

4) An Illustrative Example

- How our methods apply to simulation of activity at a railway station

5) Conclusions

W1.1. Background ideas

W1.1.1. Applications of discrete-event modelling

Applications of discrete-event modelling historically of two kinds:

Science and Engineering

- simulation and visualisation of complex systems

Orientation: Complex Systems Modelling

Software Development

- specifying requirements for reactive systems [defn]

Orientation: Advanced Programming

Convergence between these two fields

- non-determinism and concurrency, interaction with designer

engineering:	engineering systems incorporate electronics
computer science:	computer programs have rich interaction with environment
physics:	computational physicist experiments with computer model

- encapsulated behaviours no longer enough

Science &Engineering:	not just numerical solution of differential equations
Software Engineering:	iterative nature of the software development process

W1.1.2. Issues for discrete event modelling:

• *size*:
techniques for small subsystems unsuitable for complex state-transition models

• *comprehensibility*:
a model must be intelligible to the designer and should correspond so closely to the system as conceived at a high level of abstraction that it has explanatory power

• *modifiability*:
during development, incomplete system models must be represented in such a way that they are easy to enhance or revise.

W1.1.3. Current approaches to discrete-event systems modelling

Two approaches to specifying system behaviour

1. System behaviour encapsulation

Conceive behaviour in its totality
Represented in an encapsulated form
e.g. by a comprehensive state-space model
a set of differential equations
CSP specification of traces

Stateless mathematical description

In programming terms cf declarative programming

2. Explicit state-based models

Model the system state (in simplified form)

Two varieties

- unambiguous operational interpretation
  e.g. Petri nets, CCS models
  cf semantic models of programs in CS

  generally difficult to relate to the application

- closely related to the application
  e.g. concurrent object-oriented models

  generally have ambiguous operational interpretations

Tension between mathematical precision, freedom from ambiguity and intelligibility
cf Harel's *Biting the Silver Bullet*

W1.2. An Overview of our Approach

W1.2.1. Orientation of our approach

Model development = iterative process:
preliminary design
simulation
analysis
revision.

Construction of abstract model ↔ identifying agents and specifying their roles.

Need state-based model that can be:

- developed incrementally to correlate model behaviour with expt and observation

- interpreted by the designer in terms of agents acting in the application

- used for simulating and analysing system behaviour.

W1.2.2. Basic Principles

Our modelling techniques exploit *agent-oriented* and definition-based or *definitive* programming.

- *Agent-oriented* programming involves identifying the agents in a system and describing the interactions between them in terms that are easy to relate to the requirements of the application.

- *Definitive programming* describes a state-transition model of the system in terms of scripts of definitions and redefinitions.

An agent-oriented specification is given an operational interpretation by transformation to a definitive program, using additional application-oriented information about the scenario for agent action.

W1.2.3. Programming as Modelling: Relation to object-oriented programming (OOP)

- programming = modelling

  cf Simula

  Actor model of computation:

  concurrent system state
  ↑↓
  variables bound to agent instances
  dynamically created / destroyed

Differs from OOP:

- explicitly model what agents can observe and change
  – don't presume information hiding necessarily

- message passing is only one of the ways in which agents interact.

Modes of interaction outside the scope of OOP:

- direct action of one agent upon another

- single actions that effect state of several objects
  in one indivisible transition.

Advantages of our approach:

- simplifying abstraction in model design

- can model components (e.g. mechanical linkages, legal processes)
  involving synchronous propagation of change

- more powerful abstractions for communication and synchronised interaction
  ⇒ weaker hypotheses on process synchronisation
  ⇒ easier to interpret in application-oriented terms.

W1.3. Overview of our Techniques

W1.3.1. System state representation: the spreadsheet principle

How to simplify system state representation?

OOP: identify generic pieces of local state

Problem in concurrent interpretation:

Represent synchronised changes occuring
in *independent* pieces of local state
in a *conceptually indivisible* action?

Agent-oriented programming:

- faithful modelling of indivisible transitions
  irrespective of what pieces of local state they affect.
  Model conceptually indivisible propagation of state changes
  as a *single* transition.

Computational mechanism as in spreadsheet:

changing *one* value
simultaneously
changes all dependent values
as if in one indivisible transition.

Definitive state-transition models

- the entire system state is represented by a set of definitions of variables (or *definitive script*)
- an agent action is represented by the redefinition of a variable
- transition from one system state to another is modelled by parallel redefinition associated with concurrent action by one or more agents.

Virtues:

- great expressive power
- reflect context-dependence of a particular action.

W1.3.2. Agent-oriented specification: LSD and the ADM

Behaviour of discrete event system
↑↓
protocols adopted by the agents
+
context for action execution.

Two separable concerns:

- the abstract characteristics of the agents
  how agents act in isolation
- assumptions about context for interaction.

W1.3.2.1. Specifying Agents and Protocols: LSD

Abstract characteristics of an agent specified in LSD

LSD specification describes

- the aspects of the system state to which it can respond – its *oracle* variables
- those aspects it can conditionally change – its *handle* variables
- the circumstances under which state-changing actions can be performed
  – the privileges underlying its *protocol*
- definitive context for state-transition interpretation of actions
  – defined by *derivate* variables

privileges / protocol = set of guarded possible sequences of redefinitions

Role of LSD specification

- correlates possible actions of an agent with perceived states of the system
- basis for simulations of the behaviour via a definitive state-transition model

Specification expresses how

- state-transitions in the system ↔ agent actions
- agent actions consistent with perceived knowledge of system state.

W1.3.2.2. Simulation from an LSD specification

LSD specification NOT concurrent program
– ambiguous operational semantics
(cf concurrent object-oriented programs)

Operational interpretation of agent:

privileges ---> protocol

Extra considerations

- speed of responses relative to other agents
- nature of communication between agents
- scenario for selection of agent actions.

Part of model development process

Conflicts between agent actions may or may not be eliminated.

W1.3.2.3. The Abstract Definitive Machine: ADM

LSD specification interpreted operationally by simulation in Abstract Definitive Machine (ADM)

In the ADM

- computational state ↔ definitive script
- transition ↔ redefinition

ADM program ↔ a set of entities

Entity = set of definitions + set of actions

- typical action =guarded sequence of redefinitions

- definitions / actions for currently instantiated entitie

  ↑↓

  contents of definition / action stores D/ A.

In each machine cycle:

actions in A
enabled
in definitive state D
performed in parallel

From LSD specification to an ADM program:

- agent --> entity
- derivates --> definitions
- protocol --> actions

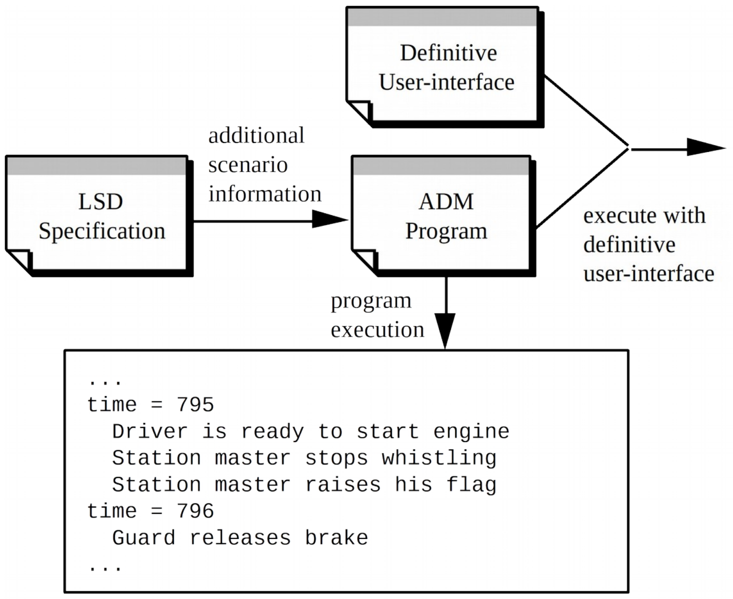
timing of redefinition in simulation
↑↓
semantics of agent actions and perceptions

W1.4. An illustrative example: modelling interaction at a railway station

Our methods have been applied to animate the activity associated with the arrival and departure of a train at a railway station. This includes simulation and visual animation of

- the protocols followed by the station-master, guard and driver
- the actions of passengers boarding and leaving the train.

Full details of the simulation model appear in [1]; LSD and our visualisation methods are further developed in [2,3]. Both textual and graphical output from our simulation are illustrated in Figure 1:



```
...
time = 795
  Driver is ready to start engine
  Station master stops whistling
  Station master raises his flag
time = 796
  Guard releases brake
...
```

Many of the issues abstractly identified in §2 are illustrated in our simulation model. These include:

- the merits of direct action rather than message passing (cf 2.2):
  Our model presumes that the station-master instantaneously registers the fact that a door is opened – thus synchronising door-opening with the station master's perception of this event. We can also model in a realistic way the fact that a passenger can directly open or close a door.
- definitive representations of state (2.3):
  The station-master's knowledge of whether all doors are closed is expressed by a *definition*:
  $$\mathbf{all\_doors\_closed} = \neg\mathbf{open\_door}_1 \wedge \neg\mathbf{open\_door}_2 \wedge ... \wedge \neg\mathbf{open\_door}_N$$
  that is to be interpreted as asserting that the value of the variable **all_doors_closed** is defined by the formula on the RHS. Definitions are also used to express the way in which the location of passengers depends upon the position of the train when they are on board.
- agent action is modelled as redefinition (2.3):
  The action of opening the first door is modelled by the redefinition $\mathbf{open\_door}_1$ = **true**. The effect of this redefinition is context dependent (2.3): it may or may not change the value of the variable **all_doors_closed** in one and the same indivisible transition, according to whether another door is already open.
- concurrent action can be conveniently modelled:
  Two doors being opened simultaneously is modelled by parallel redefinition:
  $$\mathbf{open\_door}_1 = \mathbf{true} \;||\; \mathbf{open\_door}_2 = \mathbf{true}.$$

The active agents in the LSD specification in Figure 1 include the railway personnel and passengers. They also include passive agents, resembling objects, such as the train and its doors. The LSD specification for the station-master reflects his perception of the current state of all other agents. For example, the station-master can detect whether any door is open and close an open door. When the station-master sees that all the doors are closed and the train is due to depart, he will blow a whistle. The role of the station-master can be described in such terms with reference to conditions he can perceive and actions he can perform – independently of the context for action. From this analysis, $\mathbf{open\_door}_i$ is both an oracle and a handle variable for the station-master, **all_doors_closed** is a derivate variable and a conditional privilege to whistle is one of the actions in his protocol (cf 2.4.1).

The animation of the LSD specification in Figure 1 invokes additional information about the scenario for action (cf 2.4.2). The reliability of the station-master's perceptions and the appropriateness of his actions depend on the context in which they are performed. A typical departure protocol will prove unsatisfactory if a passenger decides to disembark and board the train repeatedly, for instance. The station-master need not register every time a door is opened whilst a train is at the station, but it is essential that he correctly perceives that all doors are closed as the train departs. Guaranteeing correct perception presumes facts about the context for interaction beyond the scope of the LSD specification. For instance, the fact that the station-master has continuous visual access to the state of the doors has to be reflected in the manner in which their perceived state is maintained in the model.

W1.3.5. Conclusions

Our experience shows that our approach assists the processes of model design and simulation:

- it leads to the construction of state-based models that can be validated against experiment. Variables in a definitive script correspond to observations of the system. The "definitive user interface" in Figure 1 uses definitive scripts for visualisation of these variables, ensuring that there is a precise correspondence between system observations and transitions in the animation.
- it creates a model that relates the global system behaviour to the characteristics of participating agents. Such a model has explanatory power; e.g. we can explain how activity at the station depends upon the station-master knowing whether there are passengers in the door vicinity.
- it supports incremental development of a complex model as well as easy recovery during design iteration. Definitive scripts can also be developed independently, then linked together. The signalling protocol and passenger interactions were separately developed in this way.
- it enables rich modes of interaction in simulation e.g. allowing greater scope for automatic detection of conflict and for user intervention to modify the model design retrospectively. If the station-master attempts to close a door as it is being opened by a passenger, or two passengers attempt to pass through a single door simultaneously, this is detected in the ADM. The designer can intervene to arbitrate such conflicts by making suitable redefinitions at any ADM iteration.

References

1. W M Beynon, M D Slade, Y P Yung *Protocol Specification in Concurrent Systems Software Development*, Computer Science RR#163, Warwick University 1990
2. W M Beynon, M T Norris, R A Orr, M D Slade *Definitive specification of concurrent systems* Proc UKIT'90, IEE Conf Series **30**, OUP 1991
3. W M Beynon, I Bridge, Y P Yung *Agent-oriented Modelling for a Vehicle Cruise Control System* Proc ASME Conf ESDA'92 Istanbul, Turkey 1992 (to appear)