Agent-oriented Modelling and Simulation for Discrete Event Systems

Meurig Beynon, Yun Pui Yung
Dept of Computer Science, University of Warwick, Coventry CV4 7AL

Abstract

A modelling technique that relates concurrent system behaviour to agent interaction at a high level of abstraction is outlined. This exploits two complementary programming paradigms: "agent-oriented" and "definition-based". The method is illustrated with reference to modelling and simulation of activity at a railway station.

1. Introduction

Modelling complex discrete event systems has important applications in systems engineering, e.g. in:

- animation of the requirement for "reactive systems"
- simulation and visualisation in engineering and science.

Particular issues for large-scale discrete event modelling include:

- *size*: techniques used for small subsystems are 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.

Current approaches to discrete event modelling have limitations. Typically, ***either*** they apply to systems whose behaviour can be conceived in its totality and represented in an encapsulated form e.g. by a comprehensive state-space model or a set of differential equations, *or* they use explicit state-based models. Such models may be intelligibly related to the application (e.g. concurrent object-oriented models), or have an unambiguous operational interpretation (e.g. Petri nets), but do not combine intelligibility with freedom from operational ambiguity in realistic applications.

Our approach treats model development as an iterative process of preliminary design, simulation, analysis and revision. Construction of an abstract model proceeds in parallel with identification of agents and the specification of their computational roles. This needs a state-based model that:

- can be developed incrementally to correlate model behaviour with experiment / observation
- can be interpreted by the designer in terms of agents acting in the application
- can be used for simulating and analysing system behaviour.

2. An Overview of our Approach

2.1. Principle of the method

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.

2.2. Relation to object-oriented programming (OOP)

Our approach conflates programming and modelling in the spirit of Simula, the first object-oriented programming language. It also resembles extensions to OOP for concurrent execution, such as the

actor model of computation – the state of a concurrent system is recorded by variables bound to agent instances that are dynamically created and destroyed. It differs from an OOP paradigm in that:

- no concept of information hiding is presumed
- message passing is only one of the ways in which agents interact.

Modes of interaction outside the scope of the OOP paradigm include:

- direct action of one agent upon another
- single actions that effect state changes in several objects in one indivisible transition.

The merits of modelling direct action of one agent on another include:

- it has a useful role as a simplifying abstraction during design of the model
- it is good for modelling components (such as mechanical linkages) that involve synchronous propagation of change
- more powerful abstractions for communication and synchronised interaction mean weaker hypotheses on process synchronisation, easier to interpret in application-oriented terms.

2.3. Mode of system state representation: definitive state-transition models

In OOP, simplify the representation of system state by identifying generic pieces of local state. The problem in concurrent interpretation is then to represent synchronised changes that occur in independent pieces of local state in a conceptually indivisible action. Agent-oriented programming aims at faithful modelling of indivisible transitions irrespective of what pieces of local state they affect. Where an agent action results in a conceptually indivisible propagation of state changes involving several agents, as typically occurs in a mechanical system, this is to be modelled as a single transition. Make use of a new fundamental abstraction to model of system state, based on the computational mechanism that underlies the spreadsheet, where changing a single value simultaneously changes all dependent values *as if in one indivisible transition.*

Our representation of concurrent system state uses *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.

Definitive state-transition models have great expressive power. They also reflect the fact that context affects the indivisible transition from one system state to another associated with a particular action.

2.4. Principles of agent-oriented specification

The behaviour of a discrete event system is to be related to the protocols adopted by the agents and the context for their execution. In justifying a high-level protocol, there are two separable concerns:

- the abstract characteristics of the agents, and how they act in isolation
- what assumptions must be made about the context for interaction.

2.4.1. Specifying Agents and Protocols: LSD

The abstract characteristics of an agent are specified using a special-purpose notation called LSD. An LSD specification for an agent 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 that underlie its *protocol*, consisting of a set of guarded possible sequences of redefinitions.

It also includes definitions – associated with *derivate* variables – that can express the different ways in which agent actions are to be interpreted in state-transition terms according to context. An LSD specification correlates possible actions of an agent with perceived states of the system and provides the basis for simulations of the behaviour via a definitive state-transition model. It ensures that

- all state-transitions in the system are associated with action on the part of one or more agents
- every agent action is consistent with its perceived knowledge of the system state.

2.4.2. Simulation from an LSD specification: the ADM

LSD specifications (like object-oriented programs) invite concurrent interpretation, but have a quite

ambiguous operational semantics. The consequences of executing agent protocols cannot be predicted without additional assumptions about an agent, such as the speed of its responses relative to other agents, the nature of the communication between agents and the scenario governing the selection of agent actions. Identifying suitable hypotheses about the mode of execution is part of the development process. Conflicts between agent actions may or may not be eliminated in this process.

An LSD specification is interpreted operationally by simulation in the Abstract Definitive Machine (ADM). In the ADM, computational state is represented by a script that changes dynamically as redefinitions are performed. The state information that determines when a redefinition is to be performed is itself encoded within the script. An ADM program is specified by a set of entities. An entity comprises a set of definitions and a set of actions. A typical action is a guarded sequence of redefinitions. In execution, the definitions and actions associated with currently instantiated entities form the contents of definition and action stores D and A. In each machine cycle, actions in A whose enabling conditions are true in the state specified by the definitions in D are performed in parallel. In transforming an LSD specification to an ADM program, each agent is modelled by an entity constructed from the set of oracles, handles and derivates to which it refers:

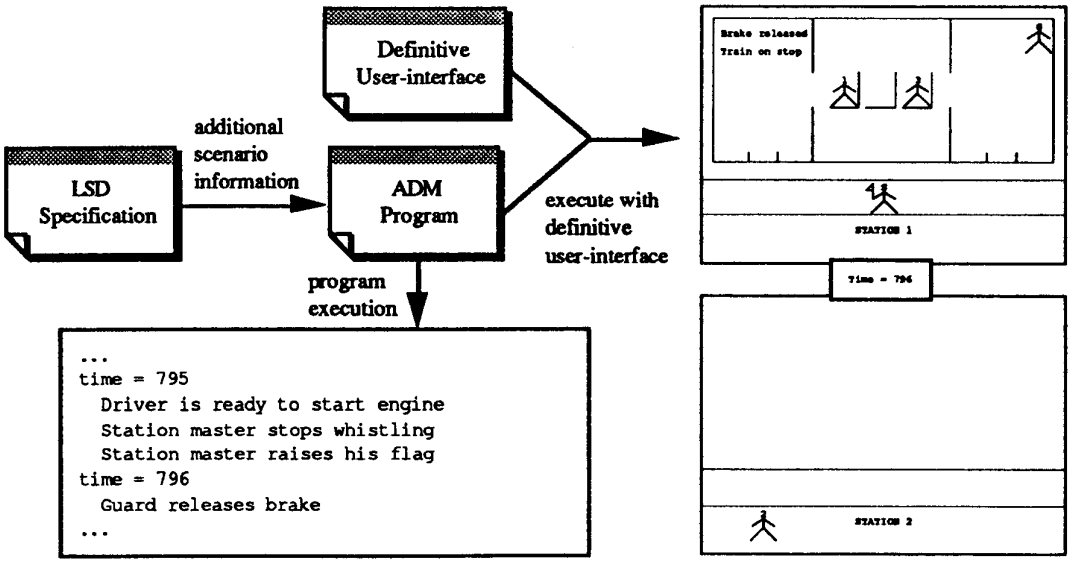
- derivates are transformed into definitions
- the agent protocol is transformed into a set of mutually exclusive guarded actions to be performed with delays between commitment to action and successive redefinitions that reflect the semantics of the associated agent actions and perceptions.

3. 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:



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*:

  $$\mathbf{all\_doors\_closed} = \neg\mathbf{open\_door}_1 \wedge \neg\mathbf{open\_door}_2 \wedge \ldots \wedge \neg\mathbf{open\_door}_N$$

  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 **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:

$$\textbf{open\_door}_1 = \textbf{true} \parallel \textbf{open\_door}_2 = \textbf{true}.$$

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, **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.

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.

Supplementary 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)