

System Validation through Model Simulation

Tony Beckwith

The importance of specifying system requirements before significant development takes place has been recognised for many years as an important ingredient for a successful software development project. However, identifying a complete and unambiguous set of requirements, both functional and structural, is often fraught with difficulties. A survey of people hitting the ARTISAN web site found that 72 per cent of system engineers considered this as one of the most difficult parts of the project.

A variety of techniques, with varying degrees of formalism, have been applied to the task of requirements definition with the aim of improving the effectiveness of this activity. Requirements modelling and prototype simulation are two approaches that have been applied with success in a number of projects. This article illustrates how these two approaches, in combination, and with the aid of appropriate tools, can significantly help engineers define system requirements.

An Example Application

The example we will use in this article is a simple car park access control system. Access to the car park is controlled by a motorised barrier and a set of traffic lights (red and green only). An IR beam detector is used to signal the arrival and entry of a car. A separate pressure sensor is used to detect cars exiting the car park. Access is only permitted if the pressure sensor is used to detect cars exiting the car park.

Access is only permitted if the number of cars currently in the car park is less than an operator-defined capacity. Setting or revising this capacity is achieved via an operator I/O unit consisting of a numeric keypad and display, and two buttons, one to display the current capacity, the other to reset the capacity to the current display value. The I/O unit also provides a power on/off key.

System Definition

System definition is the process of describing system requirements in terms of the high level structure and functioning of the system. Three different views are often taken of the system requirements:

- System architecture, defining the physical structure and connectivity.
- System usage, featuring typical scenarios of how the system will be used.
- System behaviour, describing how the system reacts to events such as input signals.

System Architecture

A number of physical architecture diagrams may be produced, at different levels of detail, to identify the physical structure of the system. This information can be summarised to show the specific devices that the control system must interface to, and what signals pass between the control system and the devices. Information relating to the physical architecture of the system can be summarised in the form of a Context Diagram.

System Usage

An increasingly successful and often-used system definition technique is to describe specific uses of the system (called use cases), and document individual scenarios as sequences of messages exchange between the system and its attached devices. The Context Diagram provides the vocabulary for describing such scenarios. Specific usage scenarios are easy for users to describe and form good acceptance tests. Modelling of system usage can be achieved using standard UML use case diagrams to indicate the structuring of, and relationships between, use cases.

System Behaviour

As the set of use cases evolves it becomes possible to build up a complete picture of system behaviour. This will enable use to specify, at a high level, how the control system will react to events, in the form of incoming signals for example, in all circumstances.

State diagrams are commonly used to describe this type of reactive behaviour. An example state diagram for the car park system is shown in Figure 2.

You will notice that the level at which the behaviour is described in this diagram is at high level, corresponding

to the level of interaction depicted in the Context Diagram. For example, the events that invoke state transitions are either the receipt of signals from interface devices or perhaps 'timer' events.

At this stage, it is not necessary or desirable to attempt to determine the structure of the software and specify system activity in terms of interaction between software units. However, you will also notice that the actions and conditions in Figure 2 are program language oriented. This will enable us to animate the state diagram as a means of verifying system behaviour.

State diagram animation is achieved in Real-time Studio via the creation of a software test harness. The test harness is generated from the state diagram and, when run, provides operational features similar to those found in sophisticated program development environments. These include facilities to inject events, view and amend data values, log activity, reset, etc. The state diagram animates in real-time using colour to show current states, previous states or unvisited states.

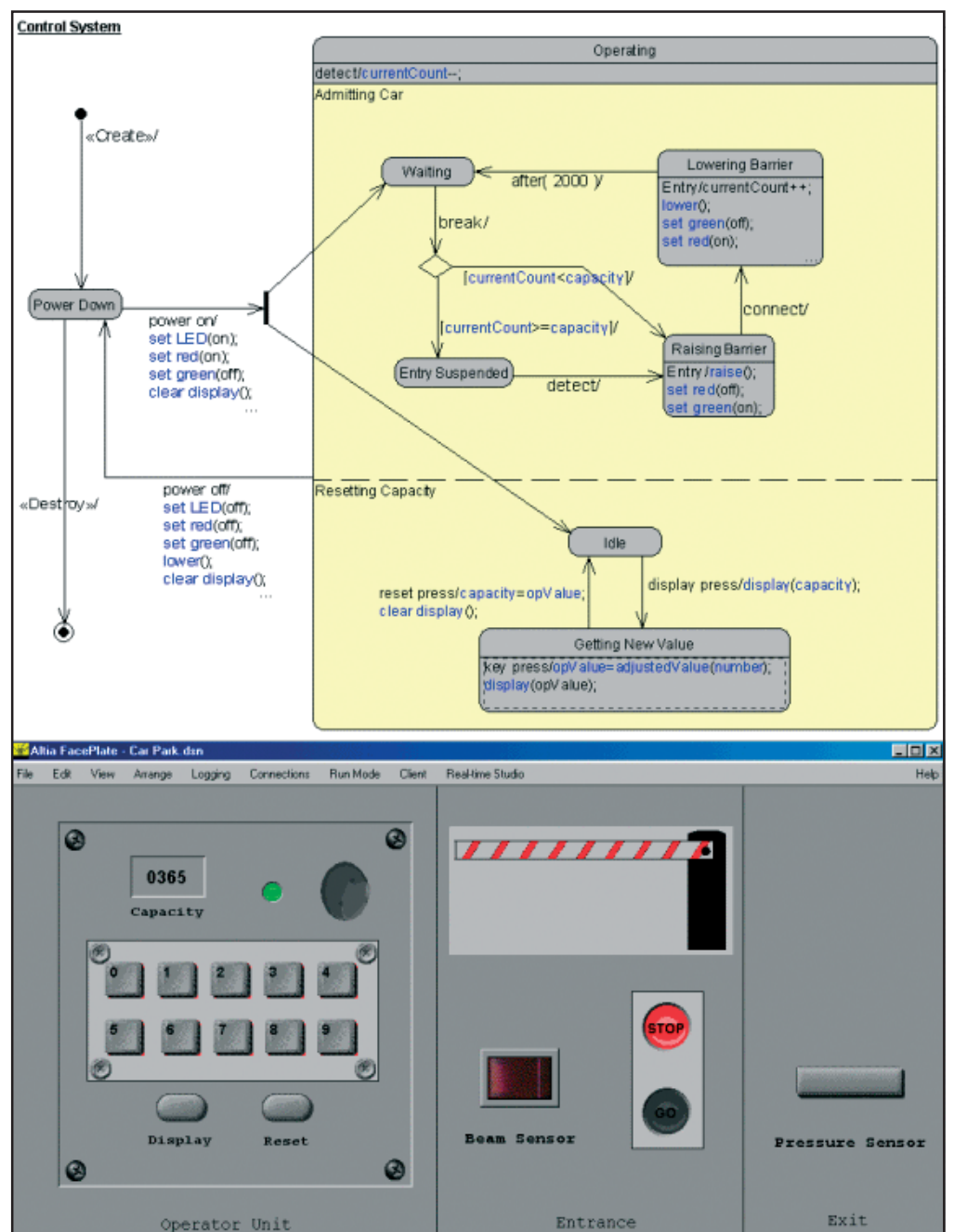
System Simulation

Using animation to verify state machine behaviour can be of great help in enabling system engineers to refine models of system activity. Increasingly though, a more demanding user community requires more rigorous methods of validating system requirements. The use of system prototypes has long been recognised as an effective mechanism for system validation.

User interface (UI) mock-ups are fairly easy to produce and can be highly effective as a means of identifying requirements for systems where the UI is well correlated to system functionality. In many real-



ARTISAN's CEO Caine O'Brian: "UML is a higher order language than 'C'".



Top: figure 2 Bottom: figure 3

time and embedded systems, this correlation is not always present, the system's functionality often being manifested in terms of device interaction. Figure 3 illustrates a mock-up of the car park system produced using Altia's FacePlate software.

This mock-up easily created using standard Altia component libraries. Each component has a set of properties that can be used to configure it for a specific purpose - for example, red or green lights. In addition, each component has a set of connector 'hooks' that can be used to define what events can be sent or received by that component. These connectors can be linked to events specified in Real-time Studio (RtS), so that, for example, clicking the Pressure Sensor on the mock-up injects a 'detect' event into the animating state machine in RtS.

If the RtS state machine was in the 'Entry Suspended' state prior to this, the result would be a transition to the 'Raising Barrier' state and the consequent sending of the 'raise', 'set red(off)' and 'set green(on)' signals to the Altia mock-up. The receipt of the signals by the mock-up would trigger the connected components, in this case to raise the barrier, turn off the red light and turn on the green light.

Conclusion

This combination of state machine animated through an Altia interface is a powerful mechanism for system requirements definition and validation. It provides a cost-effective approach to one of software development's biggest problems.

Tony Beckwith is UK training manager with ARTISAN Software Tools.
Tel: (01242 229300
www.artisansw.com