

Describing, verifying and validating a system definition

A survey of engineers based on web site contacts carried out by the author's company found that 72% of them found initial system definition one of the most difficult parts of their design project. There are a number of well-documented techniques and notations for system architecture and behaviour descriptions. This article describes and discusses the use of a set, based on the popular UML notation, which can be used for describing embedded control systems.

By Alan Moore, Artisan (☎: 01242 229306)

As system engineers explore the requirements for a system and begin to define a high-level solution, they face a number of challenges:

- Defining the system context and top-level structure;
- Understanding system-level behaviour;
- Prototyping physical appearance and user interface;

This article suggests an approach to building prototype physical UI models for such systems.

Automation tools provide significant help in applying these techniques, not just in building and maintaining the descriptions, but more importantly in simulating them at an early stage. This both helps the system engineer formulate and verify his ideas, and engages project sponsors and users in validation during the early stage of system definition.

The survey found that 56% of engineers thought that detailed simulation of a system state-chart before building the system was a valuable and a good use of their time.

A combination of ARTiSAN Real-time Studio (RtS) Professional and Altia FacePlate is presented. When used together, these two products provide an integrated, easy to use set of features that help system engineers meet the challenges they face both effectively and productively.

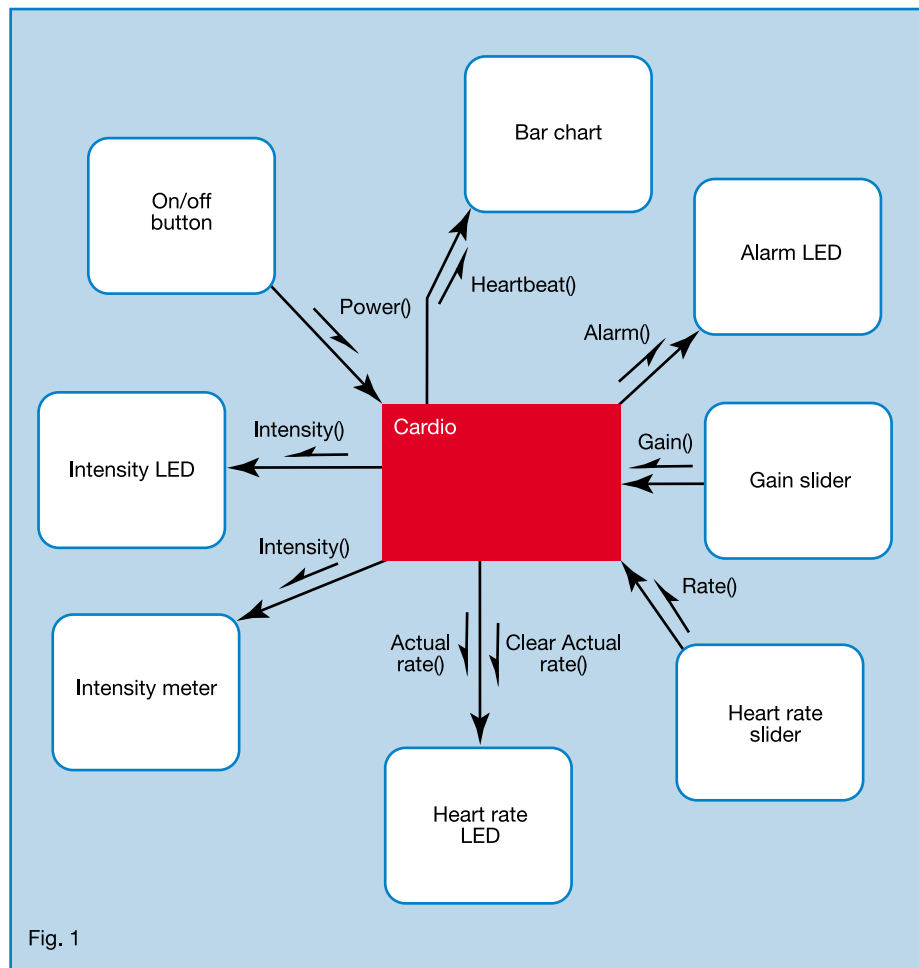


Fig. 1

Figure 1: The high-level system architecture for the cardiac monitor

Embedded System Design

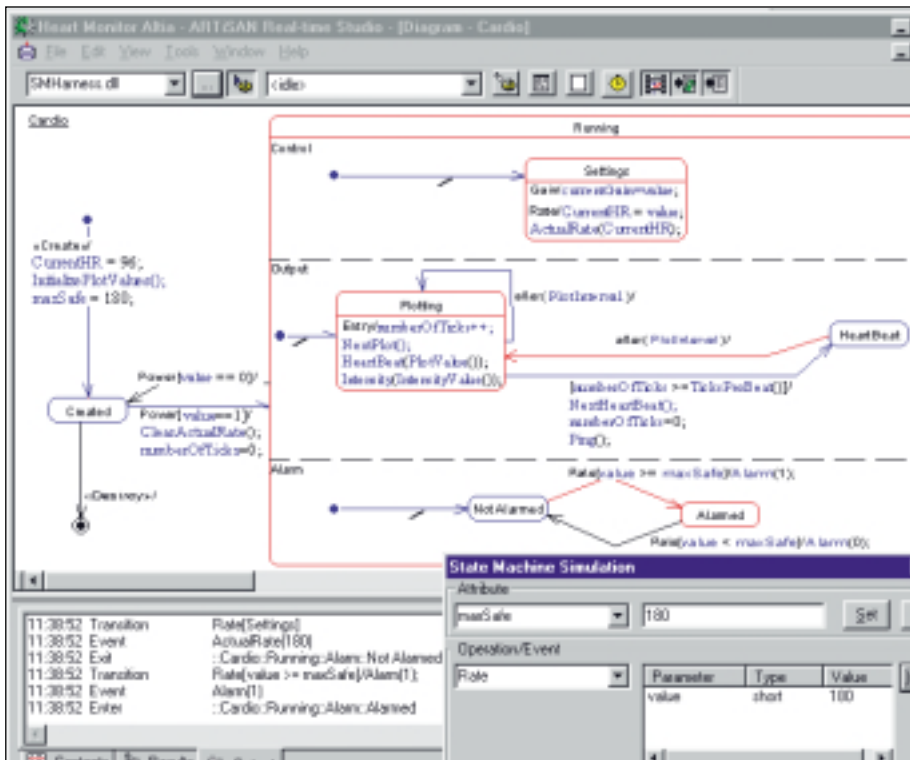


Figure 2: Simulating the system state-chart for the cardiac monitor

Learning by example

The example that will be used in this article is a simple cardiac monitor. The patient's pulse is fed to the monitor, which provides a time-based plot of heart rate plus a visible alarm when the heart rate exceeds a safe value.

System definition: System definition is the process of describing the high-level structure and behaviour of the system. Three different views are often taken of this:

- The system architecture, defining structure and scope;
- System usage, featuring typical scenarios of use;
- System behaviour, how the system reacts to input and produces outputs;

Documenting the system architecture: the place where many system engineers start is with a high-level context diagram, an example of which is shown in figure 1. This high-level architecture (sometimes called context) diagram shows the subsystem (Cardio) that we're trying to build, the devices to which it is connected and the signals that are sent to-and-fro. This is a natural starting place as it scopes the system's responsibilities and provides a "black-box" view of the system interface.

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 exchanged between the system and its attached devices. The context diagram provides vocabulary for describing such scenarios. Specific usage scenarios are easy for users to describe and form good acceptance tests.

Documenting system state

As the set of possible options is built up one can start to produce a complete definition of behaviour. Many control systems benefit from the definition of a behavioural model for the subsystem, which describes how it reacts at the highest level to input and generates outputs. When describing the behaviour of a control system, a reactive model, such as a state-chart, is a natural and often used tool. At this early stage the state-chart is described directly in terms of the signals exchanged between the control system and the system's physical devices. The system engineer doesn't need to drill down into object or class design in order to produce a useful and complete state-chart. The system architecture diagram in figure 1 shows all the information that is needed about the system interface.

The main window in figure 2 shows the state-chart for the Cardio subsystem. The state-chart is mostly triggered by signals directly from the context diagram, for example Rate and Power, but where appropriate a timed event can be used (eg, "after (PlotInterval)").

The model responds to these signals mostly by sending signals back to the devices, for example "Alarm(1)" and "ClearActualRate()" but if required, data can be remembered between signals ("numberOfTicks" for example).

How a modelling tool can help?

Clearly the diagrams described above can be produced using pencil and paper, or a simple drawing tool, so why use a modelling tool? The answer obviously depends on the features of the tool, but typically, there are four benefits:

- Automatic document generation, so that review and contractual documents can be produced simply and quickly;
- Enforcing a standard notation, such as UML, which will make the details of the system definition easier to convey to others;
- Building a sophisticated model of the system that can provide leverage further down the lifecycle (for example all of the signals on the state-chart in figure 2 were reused from the context diagram in figure 1).

■ Maintenance of the models as the project progresses is far easier than simple drawings, because the diagrams understand the semantics and reflect the content of the single, underlying model.

State-chart simulation

Another major benefit is provided by model simulation. Models can get complex when describing more sophisticated control systems, and even though we believe that UML state-charts are the clearest and most powerful of the notations available today, it is often to predict beforehand exactly how a state-chart is going to execute. A test harness that allows attributes to be set and events injected into an executing state-chart can be an essential aid in verifying its behaviour.

Figure 2 shows the simulation of the state-chart for the Cardio subsystem being tested in the test harness. RtS Professional provides automatic generation of both the simulation and a test harness interface to a state-chart.

During simulation, the red outlines identify current states and the last transitions, blue outlines identify states visited and transitions taken; black outlines surround elements that have not been visited.

Signals (called Events here) can be injected and data values inspected and changed. In this case, we can see that maxSafe is set to 180, so that has been set as the "Value" of the input signal "Rate", to test that the transition "Rate[value >= maxSafe]" is taken (the

Embedded System Design

red outline indicates that it has).

The simulation log at the bottom shows that the output signal "Alarm" with value "1" has been sent.

Validation with project sponsors

Long gone are the days when two pages of A4 were all that a project sponsor required as proof of concept. Increasingly a system prototype, the more realistic the better, is required.

Although this is more work, it results in a much clearer understanding of the proposed solution and secures the buy-in of end-users, resulting in a final system that is far more likely to be acceptable when delivered and suffers less requirement creep during development.

This applies to graphical user interfaces on CRTs or other display devices, but also to the physical interfaces provided with many control systems.

The techniques for presenting a physical system interface vary from pencil and paper to a physical mock-up of the interface, with varying degrees of cost and effectiveness. Figure 3 shows a solution of medium sophistication, designed using a physical User Interface (UI) prototyping tool.

Figure 3 shows a potential front panel for our Cardiac Monitor.

The red alarm LED indicates that the heart rate is too high.

The bottom slider emulates the pulse from a patient. The main readout shows the heart-rate plot with a digital readout of the heart rate above.

Note that the components in the simulation have been given a "real" look-and-feel to

increase the sense of reality.

This UI has been produced using FacePlate from Altia. FacePlate is very good for describing the physical UI's found in many embedded systems, such as medical electronics, process control and consumer electronics.

It contains a wide variety of hi-fidelity pre-packaged components that can be built into a new design very quickly. In addition, Altia has a more sophisticated product called "Design" that allows the construction of components

Integrating system behaviour and UI

Using an animated state-chart with a test harness is a good way for system engineers to verify their ideas about the behaviour of the system.

However, when it comes to conveying those ideas to project sponsors or end-users these might not be the ideal tools. Not everyone wants to look at an executing state-chart.

Staring at a mock-up of a front-panel has its uses too, but often users are interested in how the system reacts when the various buttons and dials are manipulated. Far better, then, is to connect the behaviour of the system as proven using the state-chart simulation to the physical UI prototype. This means that system behaviour can be validated by users in the context of a realistic mock-up of the intended UI.

To this end Altia and ARTISAN have collaborated to provide a simple point-and-click interface to enable interconnection between an RtS simulation and a FacePlate UI. The result is shown in figure 3, where the plot

and meters are driven by RtS in response to the slider values provided by "FacePlate".

Moving forward

The system definition above provides an excellent basis for moving forward into both hardware and software design.

Hardware design: The diagram in Figure 1 sits above the level of hardware and software design, providing the required context for both. Hardware engineers start to work on the board designs, defining the required processors, memories and I/O hardware to communicate with the external devices.

Software design: Often in smaller systems, once the system behaviour has been modelled and agreed, detailed design is mostly concerned with device interaction, error handling and providing an adequate level of service.

The system-level state-chart then forms the basis of the controller class in an OO design.

Using RtS this can be achieved with minimal change. Input signals can be mapped to functions in the interface of the controller class.

Output signals can be directed to driver classes. This often leaves the original design substantially untouched.

FacePlate continues to be useful as a front-end for quite a while, particularly if the system is host-based, or if much integration and testing is performed on the host.

Verification

The initial validation period with project sponsors and users provides very valuable acceptance criteria for the finished system. FacePlate provides a record/playback option that can be used as a historical record of various tests, which can be used as acceptance tests for the final system.

Conclusion

Describing a proposed system, verifying it and then validating it with project sponsors are important challenges for a system engineer at the early stage of a project.

This article has outlined a set of techniques and notations that can be used to define a system, based on the popular UML notation. It further suggests that automation can provide significant benefits not just in documenting the system but also in prototyping its behaviour and UI with users at an early stage.

This article presents one potential tool solution that meets these challenges; a combination of ARTISAN Real-time Studio Professional and Altia FacePlate. [ee](#)

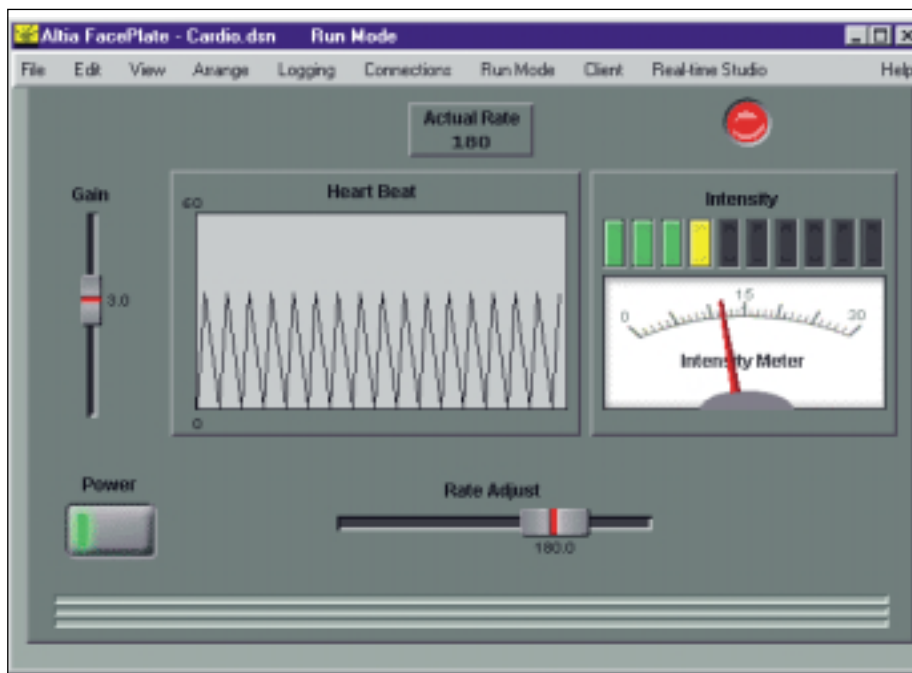


Figure 3: A mock-up of the cardiac monitor front panel