Some Structural Relationships among Models in the Development of Software-Intensive Systems

Michael Jackson, The Open University

jacksonma@acm.org

Introduction

This paper discusses relationships among certain models in the development of software-intensive systems. By ‘model’ we mean a formal description of some clearly identified subject matter, intended to support reasoning within and between descriptions. By ‘software-intensive systems’ we mean systems that interact with the human and physical world. These include systems for administrative, embedded, telecommunication and control applications, but not purely computational applications such as factorisation of large integers or finding cutsets in a graph.

We adopt the problem-oriented view of system development, in which the goal is to develop a *machine*—a computer executing the software—that will achieve some desired effect—the *requirement*—in the *problem world*. For example, in a lift control system the machine is the control computer, the requirement is safe and efficient lift service, and the problem world is the lift equipment, the request buttons, the sensors, the doors, the floors, the users, and so on.

In a software-intensive system the requirement is located in the problem world, at some distance from its interface with the machine. The lift-control machine can monitor the request buttons and sensors, and it can switch the motor current for the doors and for the hoisting mechanism; but the requirement is about the arrival of the lift car at a floor, the entry and exit of passengers, and the travel of the car to a floor that has been requested. To take another example, consider the control of a complex traffic intersection with lights, controlled pedestrian crossings, and vehicle sensors built into the road surface. The control computer can monitor the vehicle sensors and pedestrian request buttons, and it can switch lights on and off; but the requirement is about orderly, safe and efficient traffic flow—that is, about the relative movements and positions of vehicles and pedestrians.

Three Elemental Model Roles

Three distinct models, playing three elemental roles, can be clearly distinguished in the development of a software-intensive system.

- First, the *requirement* itself demands description. In the traffic intersection problem: pedestrians must be given enough time to cross the road and early enough warning when vehicle flow will be restarted; simultaneous vehicle flows must not cross each other; the lights must be timed so that all vehicles in a crossing can leave the crossing area before transverse vehicle flow is permitted; and so on.
- Second, the given *problem world properties* must be carefully analysed and described. These will include: the road layout, with the positions of the lights, pedestrian request buttons, and sensors; the properties of the sensors in relation to nearby vehicle presence and movement; expected volumes of traffic on the various routes through the crossing; minimum and maximum vehicle speeds; driver reaction times; pedestrian walking speeds; and so on.
- Third, the *machine specification*, specifying the behaviour of the machine at its interface with the problem world. This is the specification to be directly satisfied by the software that will be written, expressed in terms of phenomena directly monitored and controlled by the machine.

Dagstuhl Seminar Proceedings 06351
Methods for Modelling Software Systems (MMOSS)
http://drops.dagstuhl.de/opus/volltexte/2007/863

The relationship among these three models is clear. To ensure that the developed system fulfils its purpose the entailment

$$\textit{specification, world properties} \vDash \textit{requirement}$$

must hold. That is: if a machine satisfying the specification is installed and operated in a problem world satisfying the stated properties, then the requirement will be satisfied.

Problem Decomposition

As always, complexity demands some form of decomposition. Adopting the problem-oriented view of development, we decompose the problem into *subproblems*, each with its machine, problem world and requirement. This decomposition is not a process of refinement. In spite of the identification in the preceding section of the overall requirement as a subject for a model, we do not begin with a single exact but abstract requirement statement and proceed to refine it into a modular, equally exact, but more concrete statement, preserving the abstract properties of the single statement. Instead, we begin with an inevitably loose statement of the overall requirement, and proceed, often in a highly iterative fashion, to identify parts of the overall requirement that can be first treated as distinct subproblems and subsequently composed into a whole.

Ideally, we hope that the identified subproblems will fall into problem classes for which we have well-understood solution methods and even standard solution designs. This kind of development is similar to the practice of normal design in established branches of engineering, in which a product is developed as an assemblage of components of familiar types.

Fundamental to the problem-oriented approach is the recognition that we are decomposing the problem rather than the software. This means that each subproblem is thought of as having its own machine, problem world and requirement. The subproblem machines will, eventually, be composed—perhaps after substantial modification and transformation—into the final software product. The subproblem worlds are drawn from the problem world of the original whole problem by selecting just those parts that are relevant to each subproblem and identifying their relevant properties. The subproblem requirements are identified as parts of the original loosely stated overall requirement.

In a highly simplified illustration of such a decomposition for the lift-control problem, we may recognise these subproblems:

- 1: *Lift Service*. The subproblem requirement is to provide lift service for users in accordance with priority rules set from time to time by the building manager. The subproblem world includes the users, the lift equipment, the request buttons, and the building manager.
- 2: *Lift Safety*. The subproblem requirement is to ensure safety by monitoring the equipment for faults and, if a fault is detected, applying the emergency brake which locks the lift car in the shaft and prevents it from falling freely. The subproblem world includes the lift equipment and the emergency brake, but not the users or the request buttons or the building manager.
- 3: *Lobby Display*. The subproblem requirement is to maintain a visual display in the ground floor lobby showing the current position of each lift and the outstanding requests for each floor, allowing impatient users waiting at the ground floor to predict when a lift will become available to service their request. The subproblem world includes the display itself, the lift car, the floor sensors, the impatient users, and the request buttons, but not the emergency brake, the lift doors, of the building manager.

Further decomposition of these subproblems gives:

- 1a: *Edit Priority Rules*. The subproblem requirement is to support editing of the priority rules by the building manager. The priority rules are reified in a machine-readable data structure, perhaps held in disk storage. The reified rules and the building manager form the whole subproblem world.
- 1b: *Priority Lift Service*. This is the original Lift Service subproblem with the building manager replaced by the reified priority rules.
- 2a: *Maintain Lift Safety Model*. The subproblem requirement is to maintain a reified machine-readable model of the lift equipment behaviour sufficiently faithful to allow faults to be reliably detected in good time. For example, the model might include a representation of all floor sensor states, from which it would be possible to detect that a sensor is stuck on or off; and it might include timing records of lift travel from which it would be possible to detect that upwards travel is gradually becoming slower, indicating a possible impending motor failure. The subproblem world contains only the reified model and the lift equipment that is its subject matter.
- 2b: *Lift Safety Using Model*. The subproblem requirement is to monitor the safety model continually and to apply the emergency brake as soon as a significant fault is detected. The subproblem world includes the reified model and the emergency brake.
- 3a: *Maintain Lobby Display Model*. The subproblem requirement is to maintain a sufficiently accurate reified model of the changing position of the lift car and the changing set of outstanding service requests. The subproblem world includes only the reified model and the lift car, floor sensors and request buttons, but not the visual display itself.
- 3b: *Lobby Display from Model*. The subproblem requirement is to maintain the visual display, obtaining the relevant information from the reified model of the lift car position and outstanding requests. The subproblem world includes only the visual display and the reified model.

Reified Models

The reified models introduced into subproblems 2 and 3 are models in a special sense. They can be thought of as local variables of their respective undecomposed subproblems, maintained by the undecomposed machines so that information about the problem world can be accumulated, processed, and made available when necessary. They are *analogic* models, so called because their purpose is to support an analogy between their values and the present and past states and behaviours of the problem world parts that they model. It is this analogy—for example, between the current position of the lift car and the value of a variable *lift_pos*—that makes the model useful. By contrast, the models we considered earlier are *analytic* models, which are formal descriptions intended to be used in the development process.

Reified models appear everywhere in software systems. A database in a library administration system is a reified model of the library's books and members; objects in an object-oriented program are often—but not always—models of entities in the problem world.

It is helpful to decompose a problem that uses a model into two subproblems: one to build and maintain the reified model, and one to use the model. By doing so we obtain the usual advantages of separating the writers and readers of shared state. A more important advantage is that we expose the fundamental concern in such a problem: the need for the model to provide a sufficiently accurate analogy to the parts of the problem world that are its subject matter. An analogic model of a part of the physical world is necessarily imperfect. The machine which builds and maintains the model introduces unavoidable time lags in setting its values, and the values themselves are often highly imperfect approximations to continuous physical quantities; further, in solving the subproblem that builds and maintains the model we must use analytical models that are themselves imperfect formalisations of their subject matter.

To deal adequately with a reified model in a critical system it may be necessary to consider several analytical models. Consider, for example, the lobby display problem, decomposed into its two subproblems. We must consider: an analytic model of the lift car and outstanding requests; an analytic model of the reified analogic model from the point of view of the subproblem in which it is maintained; an analytic model of the reified analogic model from the point of view of the subproblem in which it is used; and an analytic model of the visual display. If this subproblem were an important part of a critical system—which, of course, it is not—we would be obliged to reason carefully about all of these models and their relationships. For example, to show that the state of the visual display corresponds to the states of the lift car and requests, we must rely on something like the implication:

$$(\text{Model corresponds to Lift\&Requests} \wedge \text{Display corresponds to Model}) \\ \Rightarrow \text{Display corresponds to Lift\&Requests}$$

It is not clear that sufficient precision can be achieved here for the implication to hold formally. Nor is it clear that the development can proceed by refinement: we may be unable to state the condition “Display corresponds to Lift&Requests” clearly enough until the conjuncts “Model corresponds to Lift&Requests” and “Display corresponds to Model” have been elaborated and analysed in their respective subproblems.

Non-deterministic Models for Approximation

In subproblem 2a an analogic model is maintained of the behaviour of the lift equipment, to enable detection and diagnosis of faults that may justify application of the emergency brake. The analytic model of the lift equipment in the problem world properties must therefore describe how faulty equipment conditions are related to the equipment behaviour observed by the subproblem machine.

For example, the time for the lift car to rise from one floor to the next is observable in terms of the time between the lower floor sensor opening and the upper floor sensor closing. If this time should be approximately four seconds for healthy equipment it may be supposed that an actual time of five seconds or more indicates an existing or incipient fault. The analytic model of the lift equipment may therefore contain, in effect, the statement

$$\text{rise\_time} > 5\text{s} \Rightarrow \text{rise\_time\_fault}$$

However, straightforward use of the property thus stated inevitably leads to a specification that cannot be implemented precisely: it is impossible for any machine to distinguish perfectly between a time > 5s and a time ≤ 5s. The difficulty can be overcome by stating the non-deterministic property

$$(\text{rise\_time} > 5.2\text{s} \Rightarrow \text{rise\_time\_fault}) \wedge (\neg\text{rise\_time\_fault} \Rightarrow \text{rise\_time} < 4.8\text{s})$$

and using it in the derivation of the machine specification. The interval [4.8, 5.2] in which the value of rise_time_fault is not determined provides the tolerance necessary for a formally satisfiable specification.

Different Analytic Models of a Problem World Part

A problem world part common to two or more subproblems will typically need a different analytic model in each subproblem. This difference is inevitable, because the role of the problem world model is to capture those properties that relate the requirement to the machine specification. In effect, the problem world properties expressed in the model are those on which the machine *relies* to achieve satisfaction of the requirement.

So, for example, the analytic model of the lift equipment in subproblem 1b, which provides Priority Lift Service, are just those necessary for the machine to monitor and control movement of the lift car and doors. This is possible only if the equipment is in a healthy state: if the motor has burned out, or floor sensors are stuck, or the hoist cable has broken, it is impossible to provide the required lift service. In this subproblem, therefore, the machine relies on the causal chains by which switching on the motor results in movement of the lift car, arrival of the car at a floor results in a change in sensor state, and so on. These are the properties described in the analytic model.

By contrast, the analytic model of the same lift equipment in subproblem 2a is concerned with potential faults and their manifestations at the machine interface. This is a different model from the model used in subproblem 1b. It does not rely on healthy behaviour of the lift equipment, but it does rely on the relation between the patterns of changes in floor sensor states and the healthiness or faultiness of the equipment.

By separating the two models we ensure the clearest possible description both of those lift equipment properties that are considered necessary to the provision of lift service, and of those that are necessary to diagnose equipment faults. It is worth noting the relationship between the two models. If we say that lift service relies on *healthy* equipment, and that fault detection distinguishes *some_fault* from *no_fault*, then we must have:

$$\textit{no\_fault} \Rightarrow \textit{healthy}$$

but we do not need:

$$\textit{healthy} \Rightarrow \textit{no\_fault}$$

There may be faults, such as excessive rise_time, that do not in themselves falsify the conditions on which lift service relies, but are nonetheless evidence of incipient failure that justifies application of the emergency brake.

Modelling by Successive Approximation

A fundamental difficulty in software-intensive system development is a dissonance between the non-formal problem world and the effectively formal nature of the machine and its specification. Any formal problem-world model is inevitably imperfect: the non-formal world itself can always produce a counterexample.

One way to understand the use of different models of the same problem world part is to think of it as an approximation technique. The first, crude, approximation is the model used in the lift service subproblem: here the equipment is always healthy. The second approximation is the combination of this first model with the model used in fault detection: if the equipment is not healthy then some fault can be detected.

Of course, a more elaborate approximation structure might have been chosen, in which there are more than two models. For example, if the failure of one particular sensor can be detected, then in the absence of any other fault it would be possible to provide a reduced lift service in which the lift car never stops at the affected floor. It might be appropriate to combine this model with the healthy model, or to keep it as a separate model.

Conclusion

The focus in this paper has been on relationships among models arising from problem decomposition. For brevity, we have ignored the large topic of composition, in which the decomposed subproblem solutions are reassembled into a solution of the whole original problem.

In the limited context of problem decomposition it is apparent that there can be no single model of the problem world. Different subproblems are concerned with different parts of the problem world, and where they are concerned with common parts they need different models of those parts. Each model plays a particular role in a reasoning structure within its subproblem, where it captures those problem world properties on which the subproblem machine relies to guarantee satisfaction of the subproblem requirement. The relationships among the subproblems are reflected by relationships among their models, especially in the matter of approximating the unbounded potential behaviours of the non-formal problem world.

Acknowledgements

The ideas of the problem-oriented approach underlying the discussion in this paper have been developed over several years in cooperation and joint work with a number of colleagues, including Jon Hall, Ian Hayes, Daniel Jackson, Cliff Jones, Robin Laney, Lucia Rapanotti, and others. The ideas have also been further developed by work carried out by some of these colleagues independently. None of them, of course, is responsible for any defects in this brief paper.