Succeedings of the 8th International Workshop on Software Specification and Design

Jeff Kramer and Alexander L. Wolf

Program Chairs

1 Introduction

The 8th International Workshop on Software Specification and Design (IWSSD-8) was held at Schloss Velen, Germany, in March 1996. In order to foster informed and fruitful discussions, the workshop was an invitation-only event of limited size. Based on formal submissions, approximately 60 people were selected and invited. Like its predecessors, IWSSD-8 maintained the principle that the accepted papers should serve as background material for the workshop. Therefore, the workshop did not include formal paper presentations, but rather provided an opportunity to engage in real work, with intensive discussions focussed around major themes. Each theme was discussed in a separate working group directed by a Working Group Chair who organized their group members so as to discuss the research issues of that particular theme. This year the themes selected were Requirements Engineering, Design Engineering, Software Architecture, and Concurrency/Distribution.

IWSSD has established a tradition of using "case studies" as a focus for individual working groups. These case studies, supplied in advance to participants, have proved to be a fruitful way of working. Evidence of this can be seen most clearly in the "succeedings" or workshop reports which have followed previous workshops. It was decided that for IWSSD-8, in order to provide common ground between the themes, a single common case study should be used. The "Report on the Inquiry into the London Ambulance Service" was selected, with each theme drawing on it in a manner appropriate to their own interests and concerns.

The London Ambulance Service (LAS) is briefly summarized below and discussed in the first paper appearing in the proceedings. The case study was presented in a plenary session at the beginning of the workshop and the findings of the different working groups presented and discussed at the end of the workshop, again in a plenary session. In order to make best use of the time available, working group members were asked to prepare for the workshop by familiarizing themselves with the case study and the major issues in their area relevant to that case study. We believe that this format made it both attractive and rewarding for people to attend, and was a major reason for the success of this workshop.

J. Kramer is with the Department Of Computing, Imperial College of Science, Technology and Medicine, 180 Queen's Gate, London SW7 2BZ, United Kingdom (e-mail: jk@doc.ic.ac.uk).

A.L. Wolf is with the Department of Computer Science, University of Colorado, Boulder, CO 80309 USA (e-mail: alw@cs.colorado.edu).

1.1 A Common Case Study: the London Ambulance Service (LAS)

Anthony Finkelstein and John Dowell

The LAS despatch system is responsible for: receiving calls; despatching ambulances based on an understanding of the nature of the calls and the availability of resources; and, monitoring progress of the response to the call. A computer-aided despatching system was to be developed and installed in the central ambulance control room which included an automatic vehicle locating system (AVLS), mobile data terminals (MDTs) in ambulances and supported automatic communication with ambulances. This system was to supplant the preexisting manual system.

Immediately following the system being made operational the call traffic load increased. The AVLS could not keep track of location and status of units. This lead to an incorrect database so that (a) units were being despatched non-optimally (b) multiple units were being assigned to some calls. As a consequence of this there were a large number of exception messages and the system slowed down as the queue of messages grew. Un-responded exception messages generated repeated messages and the lists scrolled off the top of the screens so that awaiting attention and exception messages were lost from view. Ambulance crews could not (or would not) use their MDTs and used incorrect sequences to enter the status information. Ambulance crews were frustrated and, under pressure, were slow in notifying the status of their unit. The public were repeating their calls because of the delay in response. The AVLS no longer knew which units were available and the resource proposal software was taking a long time to perform its searches The entire system descended into chaos (one ambulance arrived to find the patient dead and taken away by undertakers, another ambulance answered a "stroke" call after 11 hours—5 hours after the patient had made their own way to hospital).

The CAD system was partly removed and aspects of its function (notably despatch decisions) were performed manually. This part-manual system seized up completely 8 days later. The back-up server did not work since it had not been fully tested. Operators used tape recordings of calls then reverted to a totally manual system. The Chief Executive of the LAS resigned. A summary of this form cannot do justice to the range of problems identified by the inquiry. Key points which emerged were: the software was incomplete and effectively untested; the implementation approach was "high risk"; inappropriate and unjustified assumptions were made during the specification process; there was a lack of consultation with users and clients in the development process with knock-on consequences for their "ownership" of the resulting system; the poor fit of the system with the organizational structure of the ambulance service. Subsidiary to these points but nevertheless important were the poorly designed user interfaces; lack of robustness; poor performance and straightforward bugs or errors.

Though outside the scope of IWSSD, there is a very strong message in the report about the attempt to change working practices through the specification, design and implementation of a computer system.

1.2 Comments

The summaries produced by the Group Chairs of each of the working groups are presented below. It is interesting to consider the way in which each group satisfies the following aspects:

1. **Consensus.** Did the working groups come to any consensus as to which methods/techniques/tools should be applied in dealing with systems such as the LAS? Would these have mitigated the risk in its design, implementation and deployment?
2. **Integration.** What information is expected (required) from other working groups and is expected to be produced (provided) to other working groups?
3. **Omissions.** What research needs to be conducted to help in the development of such applications the future?

We hope that readers will benefit from the reports of the working groups. In addition, Daniel Jackson produced an excellent version of the LAS requirements as a focus for discussions in the Architecture group. This is presented in the appendix to these succeedings.

1.3 Further Information

The IWSSD-8 Proceedings, containing a paper on the LAS case study, thirteen full papers, and twelve position papers, is available from the IEEE Computer Society Press, order number PR07361, ISBN 0-8186-7361-3.

The printed version of the LAS inquiry report is available as ISBN 0-905133-70-6. An electronic version is available at either of the following two URLs.

ftp://ftp.cs.city.ac.uk/pub/requirements/lascase0.9.ps.gz
ftp://ftp.cs.colorado.edu/users/iwssd8/las.ps.Z

2 Requirements Engineering Group Report

Group Chairs: Kevin Ryan and Sol Greenspan

The Requirements Engineering (RE) working group viewed itself as having been called in as consultants to "do the job right" the second time around. We started by spending some time sharing our understanding of the LAS requirements, which helped us revisit and reformulate our views on what are the main RE issues. We then discussed how the research efforts of participants related to the LAS project. In particular, we asked the question "How would your approach have helped avoid the disasters that happened?". We eventually worked together to define (elements of) a requirements methodology that could have been used to improve the outcome of the project.

2.1 LAS Requirements

We found it necessary to rise above the fact that the root causes of failure of the LAS project appeared to be organizational. Certainly, protection of personal and business interests, management ineptness and neglect were overwhelming factors. However, RE deals with the boundary of these organizational problems and the formulation of technical solutions. At a minimum, all parties involved should have had an idea of what RE activities they should have been engaged in.

For the LAS project, there was no clear requirements process or product. There was nothing resembling a clear, well-understood, predictable, repeatable way to do the needed requirements activities: elicit, state elaborate, and validate requirements; communicate requirements to developers/integrators; use the requirements to control the procurement process; understand whether requirements were being satisfied; make design choices/tradeoffs based on the requirements; respond to changing requirements after the system goes into service.

Some LAS requirements take the form of high-level needs and objectives. The citizens, as represented by their government, want the best possible service for certain kinds of emergency transport. A clear understanding of what is "possible" and what is "best" is a prerequisite for further consideration of the problem. The users of the service and other stakeholders must decide what they really want, what they are willing to pay for, and how they plan to manage their needs over the long term. The LAS developers seemed to focus on problems, such as poorly handled incidents and how to avoid criticism for them, which might have been some use, but is certainly not a replacement for an understanding of what is the best service possible in the circumstances.

The high-level requirements for LAS include getting appropriate resources (ambulances and crews) to the scene of an incident in the shortest possible time; providing service in a cost-effective manner; using "good judgment" at all times (possibly combining human decision-making with expert systems); keeping information accurate and readily available, and never "dropping the ball" (i.e., never losing track of a case in progress and never failing to fulfil an obligation). To minimize delays to patients, information had to be obtained quickly, the right equipment sent to the right place, calls not interrupted or dropped, and appropriate staffing levels maintained.

One of the key requirements is to get an appropriate resource (e.g., ambulance) to the scene of an incident in the shortest possible time. Focusing first on the "shortest possible time", it is clear that this requirement needs a lot of serious clarifi-

cation, elaboration and negotiation. If ever there is a death that could have been avoided by an ambulance getting to the scene a bit earlier, then people will say this requirement has not been met. Putting an ambulance on every corner might help avoid such accusations but would obviously be too costly. Any attempt to balance response time against resources could come under attack. Remember the Ford Pinto case where Ford decided a certain amount of deaths (due to gas tanks that exploded during collisions) were acceptable because further reductions would cost too much. In the LAS case, a set of standards (called ORCON) had been developed. They state what percentage of responses must be under 3 minutes, under 10 minutes, and so on. These standards represent the acceptable "level of service" for any LAS system.

These LAS requirements demonstrate several key issues that are not addressed by previous requirements work. For one, as implied above, the requirements are actually a negotiated result among many stakeholders, including the operators, the system developers, representatives of the various technologies (e.g., communications systems, user interfaces, databases), financial interests, and the public, both as a general group whose satisfaction is sought and as victims of incidents whom the LAS is intended to serve. All of these stakeholders probably have a contribution to and stake in the response-time-related requirements.

Secondly, the early LAS requirements included a demand for complete accuracy of all information at all times. This is clearly impossible in a situation where information capture depends on phone calls, manual input and an untried vehicle location system.

A further set of issues surrounds the notion of what it means to "meet" the requirements. It is not possible to design the system in a way that will guarantee a priori the satisfaction of the requirements. For example, there is no way to simply put the ORCON tables into the system and have the system abide by them. Meeting such requirements is an operations-time phenomenon. Observations at runtime will be necessary to measure the performance of the system, and changes to the system (or re-negotiation of the requirements) are to be expected. What representation of requirements should exist to play a role here?

Requirements of this type, which could be considered "service-oriented", translate into system-level requirements: the communication system should be reliable; the ambulance tracking and scheduling system should be optimal; information entry, transmission, record-keeping, display, etc. should be effective. System-oriented requirements include human agents in the system and so are not met by technology alone but translate into competency and performance requirements on phone and ambulance staff, among others.

Other sources of requirements arise from society at large, such as needs to obey laws, physical limitations and other "business rules."

Due to space limitations, we will not go on with an elaboration of the LAS requirements, but it was clearly a good common case study from our point of view and provided the context for the useful discussions that followed.

2.2 Requirements Issues

The participants of the RE working group were asked what they thought were the main RE problems of the LAS, considering the above context. Among the answers were:

- LAS did not concentrate enough on nonfunctional requirements (NFRs). We need to identify/express/integrate them into the requirements. This led to (yet another?) discussion of whether there is really such a thing as a nonfunctional requirement. (This debate is still going on in post-workshop email discussions.)
- The problem is bad process. We need to decide what requirements should be placed on the development process itself to help ensure that the system requirements are satisfied?
- The problem is selecting requirements for actual implementation. We must make a rational stepwise selection based on cost/benefit and risk. The importance of making such decisions has been emphasized for years, somehow not being considered "interesting research" but now there appear to be some practical approaches available.
- The problem is managing and tolerating inconsistency and uncertainty. We need to detect logical inconsistencies but also to act in their presence. Several participants approached this problem by structuring requirements into "viewpoints", itself an active subarea of study (e.g., there is a workshop later this year on it). Consistency/uncertainty within viewpoints and between viewpoints can be treated as separate problems.
- The problem is knowing admissible and not admissible states under various assumptions. We know that for some types of systems, such as aircraft or missile control, such analysis is of clear benefit and some fairly mature tools exist. For other systems, characterizing the states of a system might be harder and be intertwined with, and dominated by, other issues.
- The problem is that many/most requirements are "soft" i.e., not absolute. We therefore require decisions about how, and to what extent, the requirements are to be satisfied using, for example, the selection mechanisms mentioned above.
- The user interface requirements were not understood. We must pay more attention to user interfaces. Serious problems were encountered by LAS, such as the operators being unable to prevent essential information scrolling off the screen when it was critically needed.

Based on these requirements issues, the discussion proceeded to techniques that would help remedy these problems. Not all possibilities could be discussed in the available time, but

there was a surprising—and encouraging—degree of agreement both on the issues and on a large number of the solutions.

2.3 Proposed Methodology

Rather than drag the reader through a blow-by-blow description of the discussions and debates, it is more useful to cover the points of agreement, which were folded into the form of a methodology that was presented back in the plenary session. The basic steps of the methodology were as follows:

1. *Identify stakeholders.* Stakeholder viewpoints need to be identified to seek out requirements, to maintain traceability to requirements sources, and to deal with changing requirements. Application of work on viewpoints provides tactics for managing consistency and uncertainty.
2. *Describe scenarios.* First, describe "normal" scenarios, then exceptions. Scenarios are at first a technique for requirements elicitation, also a tool for requirements validation, and later used as test cases against the operational system. Scenarios were suggested as the best way to deal with the fact that requirements are incomplete and imprecise. One cannot expect complete test suites, and in many cases, there may only be informal statements of what requirements are being tested for. However, scenarios can be designed to raise confidence about particular aspects of the system. For example scenarios from each stakeholder's viewpoint can be used to check interactions with other viewpoints. There was recognition of the similarity to 'use cases' in OOA.
3. *Identify services.* Many (or most?) systems can be viewed in terms of the services they provide to the users/customers. These services should be identified and described separately. e.g. LAS provides emergency services and also non-emergency transport services. Each type of service has parameters, variations, etc. Understanding the system in terms of services, as distinct from the processes and subsystems that provide them, seems very important as a way of separating requirements from design/implementation.
4. *Global constraints.* Not all requirements can be associated with services, functions, processes, or the like. Some requirements are global, which can mean that they are system wide, or if they are not, it isn't yet clear to which part of the system they should be allocated. We called them "global constraints" to avoid the functional/non-functional debate, although they clearly include what is usually referred to as nonfunctional requirements.
5. *Acceptance tests.* An important technique for expressing requirements is to say what needs to hold or occur for the system to be considered acceptable.
6. *Hazard analysis.* A system like LAS is judged, to a large extent, by how well it avoids hazardous situations. For example, if a phone call is dropped, someone waiting for an ambulance may not get one. What if an ambulance breaks down or gets diverted or lost in a radio blackspot or goes to the wrong accident? The system must be robust enough to recover from such hazardous situations and not allow problems to snowball. Thorough hazard analysis will seek to anticipate such situations and either prevent them occurring or recover from them rapidly.
7. *Staged introduction.* Most significant systems can not be introduced all at once. What services should be offered by when and what capabilities are needed at each stage? This consideration and the preceding one point to a spiral-like approach.

2.4 Some Conclusions

The elements of the methodology reflected some consensus among the participants on a set of practical problems that needs to be addressed by the RE community. While LAS may sound like a particularly bad case, the feeling was that many, if not most, efforts suffer from similar problems, which are neglected due to the lack of standard techniques and trained personnel.

The discussion was notably different from those at past IWSSD working groups on RE. Past emphases were on (1) representation methods in support of completeness, consistency, etc., and (2) management of requirements documentation. Instead of presentations on "yet another notation" or "yet another tool", this year's interactions concentrated on practical concerns that could contribute to the successful procurement of systems.

The RE discussions seemed nearly separate from the other three sessions' concerns. In fact, we declared that we didn't really care how the other groups distinguished themselves from each other (e.g., how are design and architecture different?). However, we recognized that there are important relationships between RE and each of the other areas. We mention two:

- By some definition of "Design", it is an activity that seeks to put together a set of interconnected components that meet the requirements. What requirements information does this activity need and in what form? RE should be responsive to the other activities by knowing what information they need. We challenge(d) the Design working group to tell us.
- LAS seemed to be a member of category of similar systems that involve communications, emergency operators, resource allocation and critical response requirements. By some definition of "Architecture", there ought to be some standard "architectures" for these systems that provide a framework for composing systems of this type. Given an appropriate architecture, it might be possible to streamline the requirements (and implementation) process because we would be formulating requirements within a framework that

can be more clearly and effectively analyzed and more directly mapped into an operational system.

RE continues to bring together an eclectic set of skills and research areas. Interestingly, some of the needs for RE that are pretty generic, such as document structuring, cooperative working, and prototyping tools. Right now these fields are advancing all on their own, independently of RE. This is to be welcomed and besides, it helps us focus on the distinctive RE issues, which the RE working group at IWSSD-8 believed to be those represented above.

3 Design Engineering Group Report

Group Chairs: Debra Richardson and Wolfgang Emmerich

3.1 Scope of Design Engineering

The Design Engineering Working Group (DEWG) felt it important to first set forth a scope for design engineering. We define design engineering as the "discipline for producing designs based on fundamental engineering principles". Essential are those principles that are complementing the craft of designing, which has been the basis of design practice in the past.

Design engineering is concerned with representing and reasoning about designs and designing. By so doing, design engineering treats design as a first class object, whereby design is described as a product. There must be a syntax for denoting it, semantics for describing it, and a basis for reasoning about it. Design should be engineered (described, defined, measured, reasoned about, evaluated, compared, improved).

An important part of design engineering are the tools used for designing. These include mechanical tools, such as design representations, design methods, and software tools for assisting in their use, as well as intellectual tools, such as design principles, guidelines, and patterns.

3.2 LAS CAD Case Study

In studying the Computer Aided Despatch (CAD) System of the London Ambulance Service, the DEWG considered what problems in the CAD system could have been avoided by "proper" design engineering, where "proper" means technology that is state-of-the-art, industrially relevant for the application, and workable in the organizational context. It was brought up repeatedly that in the CAD system development the nontechnical problems overwhelmed the technical ones. In fact, there were few explicit mentions of design problems in the study, although it was clear that the design developed was inadequate. The technical problems for design engineering included incomplete requirements, insufficient analysis, no user participation, development of a hardware-oriented design, and the lack of any design method. The main problems, however, were managerial, which included unrealistic planning, bad communication, low skills, poor organization and responsibility, and no schedule change. Thus, it is unclear whether any technological solution could have conquered this root cause. Our conjecture was that improved design engineering probably could not have conquered the problems but it could have mitigated them. So, we pressed on to consider potential solutions.

First, it is clear that both customers and contractors of complex systems must be better educated. The CAD customer lacked an understanding of the nature of the product being bought. We need to get customers to ask: What is desired? How would the customer know it if he saw it? How would she spot trouble or recognize an inadequate approach? Moreover, no contractor in their right mind should have bid this job in the time frame allowed and for the money committed. The CAD system was a much bigger job, and a reputable contractor would have known that it couldn't be done.

The CAD system had performance and load requirements that were critical and needed to be brought out early and considered throughout development. These non-functional requirements must be explicitly represented. Moreover, they must be prioritized—that is, rated by importance—since it is possible that they may be contradictory. Then, the design should be annotated to describe how these requirements are addressed, tracking requirements throughout design and related activities, and thereby relating requirements to designed components that satisfy them.

The failure to meet requirements would have been identified earlier if the development process had consisted of incremental, coordinated development activities. In particular, the design should have been developed in an evolutionary fashion, using design analysis and simulation to provide feedback and refine the algorithms. In addition, as the system was modified, the design should have evolved as well. Design rationale should have been captured so that when problems arose the design could have been more easily modified. In addition, a user participatory design process, where stakeholders remained involved during design, would have ensured usability and ownership of the final system.

On a similar note, the system should have been developed and deployed in stages rather than in one single delivery. A path for system evolution from the current practice would have enabled the users of the system to become familiar with new technology gradually rather than having to adjust to a new, automated system in a day. It would have also enabled field testing of the system in increments and supported falling back to a previous working system (possibly the non-automated system) in the case of massive failure. Incremental field testing would have identified overly complicated interfaces and inadequate procedures that could have been improved for the next version. Better algorithms for ambulance dispatch could have been added incrementally. Overall, London might have

ended up with a more workable solution, as well, such as a human-machine partnership where the machine proposes and the human disposes, with the human in the loop and hence always available as a backup.

Basing the design of the CAD system on an appropriate architectural style could have helped CAD to succeed. An architectural style provides design guidance and a framework for inexpert design team. The team starts with a proven design, has design guidance for a staged implementation, and appropriate tools for design simulation and analysis. An architecture would have brought improved design reuse options and facilitated extension of existing systems (changing buy/build decision). Design engineering handbooks stating proper architectural styles for particular problem domains would help organizations make the proper design decisions.

Design method engineering assumes that a single design method is not suitable for complex systems but rather that component-specific design methods must be chosen. After selecting an initial decomposition possible based on the selection of an architectural style, guidelines would be used to select different design methods for different components. These might include formal methods, such as Z for data-oriented components, Statecharts for process-oriented components, as well as the development of prototypes, particularly for the user interface. The different methods must be integrated and the artifacts composed, raising issues of consistency management.

Due to the organizational issues involved, we recognized that any improvement to the design process would have to be evolutionary rather than revolutionary. To ease the course, it would be necessary to integrate new design methods with those with which the developers are already familiar. Design methods would have to be engineered to be usable by developers (rather than researchers, for instance). Finally, the methods would have to be appropriate for the domain (in this case, command and control systems).

Finally, there was clearly an inadequate testing process in the CAD development. An acceptance test plan connected by consistency relationships to requirements should have been developed. In terms of design engineering, it is clear that the system should have been designed for testability and that the design should have been "tested" before being implemented. Although the DEWG clearly recognized that this is an area in need of further research and development, there are current practices that could have been used, such as developing a test plan along with the design, designing built-in tests for critical components, performing design reviews, and analyzing the design for consistency. This would have identified potential defects earlier in the process, facilitated the elimination of faults before deploying the system, and enabled testing of the non-functional requirements.

3.3 Design Engineering in Context

Much of the DEWG conclusions can be summarized by putting design engineering in the context of the other specification and design activities. In particular, the DEWG sought to stipulate what the design engineering activity needs from the other activities to enable higher quality designs. Figure 1 shows this context by displaying the artifacts of concern to design engineering and the traceability relationships between them. Thin lines denote consistency relationships between artifacts; thick arrows denote the data flow of artifacts between the various activities; and thin, dashed arrows show control flow between activities.

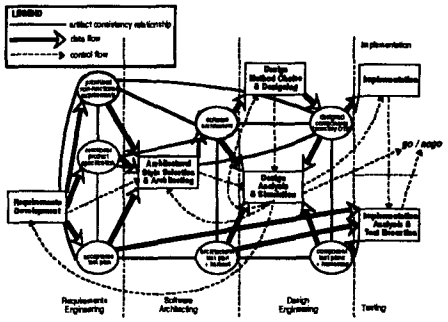

Figure 1: Design Engineering Artifacts and Process.

3.3.1 Requirements Engineering

It is important that requirements engineering not only provide a functional specification of the product, but non-functional requirements must also be provided. These include requirements that address load, performance, safety, security, maintainability, and the like. Non-functional requirements may even be more important than functional requirements. Adding a new function to a design can often be achieved by adding a new module, whereby adding a new non-functional requirement may change the architectural style of the whole system. To address safeness, for instance, a system will have to be built upon a database system or a transaction monitor to recover from hard- or software failures.

Another critical input from requirements engineering is an acceptance test plan, which complements the requirements. Such a test plan provides a means for the procuring organization to accept or decline the delivered product. An acceptance test plan also enables the developing organization to gauge the requirements.

3.3.2 Software Architecting

A high quality design cannot be achieved without a suitable architecture. As discussed above, an architecture should be developed from an architectural style that is appropriate for the non-functional requirements and the application domain.

Again, the software architecture should be accompanied by a test plan that supports testing the architecture for satisfaction

of the non-functional requirements. Furthermore, tools for analyzing and simulating the architecture are essential so as to enable early detection of inappropriate design decisions.

3.3.3 Design Method Selection and Designing

We note that there are not only dependencies between architecture and design artifacts, but also dependencies between architecture artifacts and the selection of design methods. An architectural style that suggests a typical information system architecture, for instance, has certain implications on the design notation: Appropriate methods should be chosen for database schema design as well as for reports and forms definition. This dependency has important consequences for method and tool integration. Design method engineering may only commence after the architectural style document has been completed. Since tool selection and/or construction depends on the method engineering result this will even start later. A desirable integration of architecture and design tools is complicated since the architecture tool may have been used for the production of the architectural style definition before it can be integrated with the design tool that has been selected or constructed.

In parallel with designing is the development of a component test plan and the required test harnesses.

3.3.4 Design Analysis and Simulation

Finally, a critical activity in design engineering is reasoning about the design. This includes both analysis of the design representation and simulation of the design behavior.

3.4 Design Engineering Research Agenda

In light of the case study, we also weighted what problems are not addressed by state-of-the-art and hence what issues are raised to redirect research priorities. Our top priorities included:

- Design analysis methods
- Design simulation methods
- Design method definition
- Method integration
- Experience representation (architectural styles handbook, method and process engineering handbook)
- Traceability of non-functional requirements
- Integration of off-the-shelf components
- Reliability of off-the-shelf components
- Basic infrastructure for integrated tools and environments

4 SW Architecture Group Report

Group Chairs: Paul Clements and Daniel Jackson

The Software Architecture Group was charged with understanding the architectural implications of the London Ambulance Service case study. In particular, how to specific requirements for a system manifest themselves in the architectural decisions that must be made by the system builders?

How are architectural decisions made? Why are systems structured the way they are? Where does the choice of components and their interaction and coordination mechanisms come from? Can there ever be an architectural guidebook or handbook in which we can look up a given system's problem statement and find a worked-out architectural design? Such a handbook seems to be a goal towards which many in the field are striving. Several working group participants explicitly suggested that a worthwhile goal for architecture practitioners would be to capture and structure information so that design decisions need not be constantly re-invented.

Towards this end, we tried to understand the requirements for the London Ambulance Service (LAS) system to see where they seemed to levy requirements for the architecture. Our primary source of requirements information was the requirements specification written by Daniel Jackson before the workshop, and circulated to the working group participants. The specification is given as an appendix to these succeedings.

Also circulated to working group members before the workshop was a candidate architectural description for the LAS system, written by Paul Clements. This is reproduced below. The intent was to crystallize a set of architectural decisions so that they could be discussed and debated, and yet leave enough information unbound so as to facilitate further creativity. Further, many of the working group members use specific architectural representation techniques; a common architecture was intended to let them show off their favorite languages in a way that invited comparison.

4.1 Candidate Architecture

The candidate architecture circulated to working group members comprised three sections: A set of components or modules, a layering scheme for programs within those components that controls which facilities programs may avail themselves of, and a process structure. These three structures together define the architecture.

1. **Components: The static structure**
   (a) There is a "map server" that returns the sector of a given location, and the distance and approximate driving times between two given locations.
   (b) There is an "ambulance server" that returns the status of a given ambulance.
   (c) There is a "hospital server" that provides the location of each hospital, as well as its resources, and certain capacity and status information (such as being the destination of previous calls).
   (d) There is a "lot server" that provides the location of each lot.

(e) There is an "ambulance dispatcher" that chooses the best-suited ambulance(s) to respond to an incident, and chooses the best-suited hospital(s) as the destination(s).
(f) There is a user interface module, which manages the dialogue between the system and its human users.
(g) There is an incident database, which stores incident records.
(h) There is a "function driver", which takes user input (via the user interface module) and responds appropriately: updating appropriate servers and databases, dispatching ambulances, selecting hospitals, notifying the user(s) of its dispatch decisions (so that hospitals can be notified), etc. This "function driver" is partitioned into subcomponents as appropriate.
(i) There is an "incident tracker", which tracks incidents and makes sure their status migrates towards resolution in a timely manner, and raises warnings if not.
(j) There is a "processor monitor" that makes sure each processor in the system is working properly, and causes migration off of a mal-functioning processor.
(k) There is an operating system, with clock facilities, and facilities for spawning and killing processes, and for inter-process communication and synchronization.

2. **Interaction of components**

The architecture is layered:

- *Layer 1:* Function drivers, user interface module, incident tracker, processor monitor
- *Layer 2:* Ambulance dispatcher
- *Layer 3:* Servers and databases
- *Layer 4:* Operating system

Software at one layer is only allowed to make use of (i.e., call or assume the existence of) software at the same or any lower layer.

3. **Dynamics**

The reliability monitor is implemented as a set of processes. There is one process per pending incident, created and destroyed as needed.

The system shall be implemented on a three-processor system. The software shall be allocated across the three processors initially. The processor monitor is implemented as a trio of processes that runs on all three. In case of processor failure, software function shall be migrated off the mal-functioning processor and start up on the remaining processor(s).

4.2 Architecturally Relevant Requirements

After preliminary discussion about the scope of the task, and indeed, about the scope of the term "software architecture" itself, the working group took as its first goal to understand which of the LAS requirements had architectural ramifications. Towards that end, we identified the following relevant aspects of the problem:

- the system is expected to be long-lived and extensively modified
- the system is fundamentally a command-and-control system
- the system is advisory (in that it advises actions for humans to take) rather than automatic (in direct charge of resources)
- the system allocates and manages resources, which are finite
- the system relies on decentralized and unreliable information
- the system provides a time-critical service
- incidents are widely distributed
- the system is safety-critical
- the system must incorporate legacy hardware and software
- the complexity of the system suggests a shadow deployment strategy
- the system must be usable and user-friendly

This is an incomplete list, but provided enough material to start proposing and discussing architectural alternatives.

For example, the kinds of modifiability that we expected implied certain key encapsulation decisions, leading to the identification of particular components. For instance, a map server could encapsulate all of the details about how geographic locations are translated into distances and/or directions for reaching an incident site. A hospital server could encapsulate all of the changeable information about a hospital, etc.

The perceived need for a shadow or staged deployment suggests a constraint for the automated system's architecture to mirror that of the manual system it is replacing, so that components may be brought on-line in an orderly manner. It also suggests an interface capability between the old system and the new.

The existence of unreliable data suggests the need to install resource tracking and incident tracking monitors that provide warnings if incidents are not resolve in a timely manner.

The identification of a particularly relevant domain—in this case, command and control—suggests that the system might profitably borrow previously-demonstrated architectural solutions to similar problems.

The existence of finite resources suggests a design that manages the allocation and freeing of those resources.

Time criticality suggests that real time will be one of the resources that the system will have to manage, perhaps by way of checking to make sure that functionality is delivered within specific deadlines.

The use of legacy hardware implies certain structuring decisions that will encapsulate the legacy components, as well as interface design decisions to accommodate their design.

4.3 Conclusions

4.3.1 Reusing solutions of similar problems

The notion of "similar problems" played a key role in our discussion. We appealed to Michael Jackson's notion of problem frames to see if we could position the London Ambulance Service system appropriately, and see if such a positioning would shed any light on an architectural solution. Jackson's problem frames seek to identify the nature (as opposed to the domain) of a particular problem by describing the requirements for systems usually associated with each particular class of problem. For example, Jackson makes a distinction between systems with "biddable" (in the sense of doing one's bidding) resources as opposed to automatic, dedicated resources. The difference is biddable resources may be instructed or requested to perform an action, but cannot be compelled to do so. Ambulance drivers, for instance, may be told to report to a certain location, but they may not get there for any number of reasons. Since the distinction between biddable and dedicated resources was one that the designers of the London Ambulance Service system apparently failed to make, and since that distinction was a primary factor in the failure of the system, and since Jackson's problem frames identified it as a key discriminator, we felt that the approach was quite promising.

Much of the work in software architecture is oriented towards providing an architectural guidebook: Given a problem, the guidebook would show solutions to that or similar problems that have worked well in the past. The guidebook would prevent re-invention, false starts, and misguided effort. Whether or not such a guidebook could really ever exist is open to debate; however, if it can, an approach to framing the problem, such as Jackson's, will be a necessary first step.

4.3.2 Non-behavioral Requirements Drive Architecture

We observed that by and large, it was the non-behavioral requirements for the system that drove many of the architectural decisions: its long life, its modification or evolution pattern, its performance and distribution and safety-critical requirements, and so forth. It might be argued that since architecture is largely concerned with decomposition of systems into parts, architecture is what largely addresses such issues. System behavior or functionality can be achieved in many ways, but the decomposition and interaction among the parts is what primarily affects changeability, performance, fault tolerance, and so forth.

4.3.3 Case Study Usefulness

Our group observed that the case study was a mixed blessing. While it focused our discussions, so much of the necessary requirements decisions were not available, and so it was difficult to be precise about design alternatives that would solve the problem. We therefore underscored Fred Brooks' observation (made in The Mythical Man Month, 20th anniversary edition) that is better to be wrong than vague: If you are wrong, someone will tell you. If you are vague, you may be wrong but nobody will notice. Architects require specificity (or at least specifically bound possibilities) in order to work effectively. As we progress towards our conceptual goal of a guidebook, we will need to understand how to state our problems in terms that allow identification of previously-used solutions, how to represent those solutions and tailor them to the specific instance at hand, and how to evaluate the results and verify that we indeed made the correct architectural choices.

5 Concurrency and Distribution Group Report

Group Chairs: Gerald Karam and David Rosenblum

The members of the Concurrency and Distribution working group used the workshop time to: (1) explore issues arising from the examination of the common case study, the London Ambulatory Service Computer Aided Despatch system, and (2) identify research areas from this exercise that represent problems worth investigating. The approach taken by the group was as follows: (1) review each participant's perception of the issues in the case study, (2) identify the main points and then try to answer a set of fundamental questions around the case, and (3) hypothesize research problems.

The first step of reviewing the case study was to frame its relationship to the working group's interests. In particular, the LAS system did not possess much in the way of concurrent software that needed to be developed, but rather had many distributed, concurrent components that needed to function under the management of a centralized reactive system. Thus the group largely explored the case from this perspective.

5.1 Group Member Views

The following presentations were made by the primary participants on the LAS problem (in the order of presentation).

A. Silva: Reported on an object-oriented technique he is using that approaches the problem by separating it into abstractions, followed by a separation of concerns between the application domain (functions) and technical issues (e.g., concurrency control), followed by the identification of work products, particularly those that can be reused. One significant observation with regard to the LAS application was the need

to define the degree of consistency needed for the interleaved activities of the LAS components.

D. Rosenblum: The LAS system is a reactive system, thus techniques such as Statecharts and Petri Nets would be of use in analysis of the system behavior. Specialized testing such as load testing and run-time checking may have been useful in identifying the major technical problems earlier, however it is not clear that even if this testing had been done, that it would have substantially impacted the success of the project.

U. Buy: The problems were not due to the failure of the usual concurrency problems; e.g., deadlock or starvation. Instead they were faulty specs, failure to consider errors in concurrent system objects, inadequate testing (although some testing had revealed errors that were nonetheless ignored by management), and poor management. The system could have been successfully built using off-the-shelf methods and sound software engineering practice.

A. Fekete: The application is a distributed system with a centralized, reactive subsystem providing the main control. A state transition model would be suitable for modeling purposes. However the most careful model could not avoid disaster caused by unrealistic assumptions (such as the requirement for precise knowledge of vehicle locations, and the manual intervention). However, the understanding brought out by a good model can help to isolate assumptions and provoke discussion. A. Wolf noted that it may be difficult to separate assumptions from design decisions.

R. Kurki-Suonio: This application really needed an operational versus a properly-oriented specification technique. Modeling of the system and environment were needed to tease out assumptions; i.e., by having to think about the whole system. The logical properties versus the efficiency (performance) concerns were not dealt with adequately.

A. Lakas: LAS failed due to an inadequate specification and poor testing, although the testing may not predict successful deployment. He advocated an approach based on separation of concerns: functional versus non-functional, where the functional concerns would be specified first, and then extended with assumptions and non-functional requirements.

S.-C. Cheung: Proposed a technique of compositional reachability analysis that would integrate analysis of hierarchical components for a comprehensive analysis. The major challenge with respect to the LAS problem was to fold in errors, inaccuracy, delays, and issues such as crew frustration (human factors) into the analysis. One could imagine transitions with delays and inaccuracies in the model. From this analysis a cause-effect diagram could show problem areas.

L. Semini: Presented a property-based technique using a coordination language (like Linda) and formal refinement method, using LAS as an example. The refinement method required manual intervention to transform the specification into an implementation.

G. Karam: Wrapped up the discussion by relating how some companies confront the design of complex systems such as the LAS. A particular example were various incarnations of a towed array sonar system built for the Canadian Navy. The company that developed the most complex version had been successful because: (1) it had talented engineers, (2) it used a methodology that emphasized risk reduction though simulation and prototyping, and (3) the engineers believed in this approach and applied it all times.

5.2 Discussions

At the end of the presentations and ensuing discussion, the group proceeded to the second phase of work. In this step, assumptions of the LAS system were made in order to frame responses to the question: How could the techniques of concurrent and distributed systems helped a reasonably skilled company in successfully completing the LAS system?

The following are the main comments that were made by the primary participants on the LAS problem (in the order of discussion).

L. Semini: Techniques for reliable communications could have helped the specific technical problem of fading mobile communications channels, and the ensuing errors this introduced. Also, the user community could be involved in the human interface design to eliminate the operator problems in the application.

S.-C. Cheung: The risks, behaviors, and procedures in the manual LAS system should be modeled as inputs to the design of the automated system.

A. Lakas: Most any concurrency analysis and modeling technique could be used to get the requirements well studied; e.g. simulation. Still unclear was how formal methods could be used to help represent and study this problem. G. Karam remarked that a formal method for validation seemed appropriate because of the safety critical nature of the system.

R. Kurki-Suonio: Recommended against any emphasis on formal methods, instead the focus should be on understanding the reality of the components to be used. For example, prototypes for the user interface and the mobile communications subsystem. These would then be inputs to modeling and design.

A. Fekete: Advocated a semi-formal approach using a tool for the design of reactive systems, probably using a state-based model. Problems in the environment need to be taken into account. Analysis at the functional level would proceed first, and then tuning for performance. (There was an open discussion on the role of performance engineering as part of the design process, and it was felt that it should be addressed as an integral part of the concurrency model, starting from rough approximations and progressing to better estimates as the details of the design become exposed.)

U. Buy: The original problem had no solution given the limited skills, budget, and logistics of the situation; i.e., no credible engineering company would have accepted this contract,

as is. He felt that with proper expertise, software engineering procedures, and/or good experience, a reasonable company could have solved this problem given a fair budget. In particular, simulation, rapid prototyping and the use of operational analysis models (e.g., Petri Nets) would have assisted in better understanding the problem.

D. Rosenblum: There needs to be a domain expert in every part of the development process. Also, the field testing and roll-out of the system should occur in an incremental, phased approach in smaller geographic areas.

A. Silva: Recommended decoupling logical and physical views, and tailoring analysis to the different components. Reuse of existing communications mechanisms and frameworks would be helpful.

5.3 Summary of Conclusions

The discussion was then summarized for presentation to the final plenary session of the workshop. The summary conveyed the following main points:

No technical solution was likely under the existing social and financial constraints by a reasonable, professional engineering firm (this firm would have modern software engineering processes and/or domain experience in safety critical, reactive systems). Therefore if we consider the actions of such a firm under reasonable constraints, the following methods could help to solve the LAS application. (We have assumed the basic problem is solvable with the equipment specified; alternatively the engineers would have to provide new ideas to the system architecture).

The fundamental approach is to minimize risk:

1. Perform early analysis of environmental characteristics:
   - identify people and components that form the system elements;
   - use prototyping, simulations to obtain measurements of poorly understood behaviors in system elements;
   - produce a gross-level performance model of the system.
2. Construct an operational model of the system
   - would use uncertainties and time delays from step (1);
   - explore functional behavior in detail;
   - integrate and validate a refined performance model resulting from the specification of functional components.

The last step of activity in the working group was to identify research problems. There was considerable discussion that lead to a specific proposal that would address what was felt to be the main gap in techniques that would have helped the LAS application. Specifically it was felt that existing concurrency and distributed system modeling approaches were deficient in their support for risk analysis. The proposal, which was also presented at the final plenary session of the workshop, is that risk analysis be integrated into state/transition-based methods. In particular:

1. Allow for the association of numeric quantities or intervals (delays, costs, probabilities, errors, uncertainties) with states and transitions.
2. Allow for the "coloring" of states and transitions to distinguish between failure and valid modes.

With these extensions, analysis techniques would identify transition paths leading to the expansion of error quantities as potential problem situations for the engineer to analyze. For example, the uncertainty of some value getting progressively larger in perhaps an unbounded way may represent an error in the system, or perhaps a concern if there is an assumption about stability in the value.

It would be especially desirable if mathematical characterizations of these notions of risk could be established, much as we have mathematical definitions of deadlock, starvation, boundedness, etc. We would hope that the formality of the mathematical representation would allow a wide variety of analyses to be done in the area of understanding risk in a general sense in concurrent, distributed systems.

6 Acknowledgements

We gratefully acknowledge the work of the Program Committee and the Working Group Chairs who contributed a great deal of effort to the preparation of the workshop. We thank Anthony Finkelstein for suggesting the LAS case study, making the material available on-line and, together with John Dowell, for presenting the case study so competently. We would also like to express our gratitude to the Communications Directorate, South West Thames Regional Health Authority, for allowing us to use the LAS material. Finally we would like to thank our General Chair, Wilhelm Schäfer, for the helpful and efficient way in which he organized and ran the workshop.

A System Requirements Description

Daniel Jackson

A.1 Purpose of Description

The following is an attempt to characterize some of the requirements of a computer-aided despatch system. It is loosely based on the report of the inquiry into the London Ambulance Service catastrophe, but because that document provides hardly any technical information, almost all the details

are invented and thus suspect. Nevertheless, it might still provide some basis for a discussion of architectural questions in the design of such a system.

The structure of the document roughly follows the scheme proposed by Michael Jackson in "Requirements & Specifications: A Lexicon of Software Principles, Practices and Prejudices" (Addison-Wesley, 1995). It attempts to obey the methodological principle of "description before invention", by recording domain properties before considering the observable behavior of the actual system. The "requirement" is seen as a constraint in the real world that the system should bring about, and is considered prior to the "specification", which describes the interface of the system.

The document is very incomplete and has not been carefully checked.

A.2 Purpose of System, Scope of Description

The purpose of the system is to automate the handling of emergency calls and the allocation of ambulances to incidents. This description considers only the software that executes at Central Ambulance Control. It ignores mapping software, communications systems, mobile terminals and the vehicle location system. Instead, information about vehicle location and map details are assumed to be provided in a form specified below.

A.3 Domain Description

This section documents properties of the subject domain; in this case, these properties are closely related to those of the environment in which the system operates.

A.3.1 Entities

- *Call:* A call is a telephone call to the emergency service, identified by the caller's phone number and the starting time of the call. This means that a call might not be associated with a single caller: the operator may talk to many parties in succession.
- *Ambulance:* An ambulance is any vehicle sent to an emergency scene by the ambulance service and may thus include supervisor cars, fire engines and emergency trucks. An ambulance is identified by its license plate, not by its team or by its equipment.
- *Incident:* An incident is some kind of accident or emergency that the ambulance service may respond to. This is the only reason for dispatching ambulances; we are ignoring the use of ambulances for non-emergency transportation. The notion of incident is nebulous, since the cause of an incident and its extent may not be precisely defined. So we shall interpret the notion of incident in terms of the service provided: an incident occurs at some location, and continues to be the same incident until the incident is deemed to have been resolved. The granularity of locations may thus determine whether two incidents are the same.
- *Location:* The physical terrain of London is divided into locations, so that any point belongs to exactly one location. Different floors of a large building may be regarded as belonging to different locations. The notion of location is used to determine where incidents occur, and not for locating ambulances as part of resource allocation.
- *Sector:* The set of locations is partitioned into sectors for the purpose of allocating resources. A sector, for example, may be served by a single hospital and a single ambulance station. Sectors are likely to be physically contiguous, although this may not be necessary. For example, the locations in an area may be divided into sectors according to whether the land use is residential or industrial areas.
- *Lot:* A lot is a place from which ambulances are dispatched. Lots may be at ambulance control stations, hospitals or anywhere else that ambulances are stored temporarily.
- *Hospital:* A hospital is identified with its emergency room: it is a sink for the ambulance service, and not a point from which ambulances are dispatched. If an ambulance can be dispatched immediately following a delivery of a patient to an emergency room, it is viewed as being dispatched from an adjacent lot. Not all hospitals need have this facility.
- *Resource:* A resource is any capability to provide medical services. It may consist in special equipment, or in the expertise of emergency team members. Resources are associated with hospitals and ambulances.
- *ResourceClass:* Resources are grouped into abstract classes representing their function, such as "operating rooms", "stretchers", "respirators". The set of classes is not fixed.
- *Time:* A global notion of time is assumed: we shall ignore all problems of clock inaccuracies and inconsistency. A moment in time is perhaps identified by a date, and the time of day in hours and minutes.

A.3.2 Non-entities

Some candidate entities are disqualified methodologically. The notion of a "team", for example, is not easily made precise. The entities given above are "designated" in the sense that it should be easy to determine by observation in the real world whether something is indeed an entity of a particular type. No such recognition rule could be found for the notion of "team", in contrast, say, to "hospital".

Another reason for dismissing candidates on methodological grounds is that we only model "atomic" individuals as entities. Many entities that might have been included in a traditional ER analysis become entity properties. The set of available resources, for example, is not modeled as an entity in itself: it is not an individual that has identity. A sector, on the other hand, can be regarded as an entity, even though it "consists" of locations: the point is that it has an existence in its own right too.

Other candidates are ruled out simply by being regarded as beyond the scope of the requirements. The notion of ambulance location, for example, is clearly important in the actual LAS system, but we have decided to ignore the tracking of ambulances en route. The notion of "injured party" is presumably not important even in the actual system: it is simply not the job of a CAD system (currently at least) to track the treatment of patients.

- *Team:* The emergency teams may not stay together, so should not be regarded as entities. Their members are treated like other resources (such as operating rooms and cardiac equipment).
- *Ambulance Location:* The position of an ambulance on its way to and from an incident is not modeled. A more realistic system would keep track of ambulance locations independently of incident locations.
- *Incident Class:* Unlike resources, incidents are not grouped into classes. The system will thus not support automatic association of resource classes with incidents.
- *Injured Party:* The purpose of the system is not to track the treatment of individuals, so the treatment of particular people is not relevant. An incident creates a need for resources; how these are allocated to people is handled manually at the scene of the incident and by triage in the hospital.
- *Calling Party:* The identity of callers is not regarded as any more important than other details of a reported incident, such as the owner of the property on which the incident occurred. This means, for example, that there will be no support for identifying a thread of calls from a given party and giving it special treatment.
- *Phone:* Likewise, the phone from which an emergency call is made is not regarded as fundamental, even though the system is provided automatically with information about the directory number and location of the caller.
- *Severity, Priority, etc.:* Abstract notions such as these are regarded either as properties of entities or as implementation details. Attributes of entities are modeled as monadic predicates on entities, and have no existence as entities in their own right.

A.3.3 Events

Only events that are observed externally by the system and are not under its direct control are listed here, so we omit irrelevant events (hospital treats patient), system-generated events (dispatch ambulance), and undetectable events (incident happens).

The events are grouped by entity; when an event involves several entities, we pick the one that constrains the time-ordering of the event. For example, **arrive_location** involves an ambulance and a location, but we classify it under the ambulance entity since arrival and departure alternate for an ambulance but not for a location. The secondary entity is given as an attribute of the event; the primary entity is implicit.

The events of each entity are followed by a regular expression showing the allowable orderings.

- *Ambulance Events*

```
arrive_location (l: Location)
leave_location (l: Location)
arrive_hospital (h: Hospital)
leave_hospital (h: Hospital)
park_at_lot (l: Lot)
leave_lot (l: Lot)
decommission
commission_at (l: Lot)

(commission_at
(park_at_lot leave_lot
arrive_location leave_location
[arrive_hospital leave_hospital]
)*
decommission)*
```

where **commission_at** means an ambulance is made available for service, and **decommission** means an ambulance goes out of service.

Note that this grammar makes explicit a very unreasonable domain assumption: that ambulances never break down en route.

- *Resource Events*

```
apply_at_location (l: Location)
apply_at_hospital (h: Hospital)
release
commission_for_hospital (h: Hospital,
                         c: ResourceClass)
commission_for_ambulance (a: Ambulance,
                          c: ResourceClass)
decommission


((commission_for_hospital
(apply_at_hospital release)*
decommission)
|
(commission_for_ambulance
(apply_at_location release)*decommission))*
```

where **release** means a resource is released and available for another incident.

Note that a resource cannot be applied at a hospital and a location without being decommissioned in between.

- *Call Events*

```
received (t: Time)
classify (i: Incident)
reclassify (i: Incident)


(received classify reclassify*)
```

- *Incident Events*

```
assess (where: Location, when: Time,
        needs: ResourceClass -> Nat)
reassess (where: Location, when: Time,
          needs: ResourceClass -> Nat)
```

A human operator performs the classification and reclassification of calls, and the assessment and reassessment of incidents, based on information recorded during the call. The form of this information is a *design* and not a domain description issue: the problem is to find information that enables these incident properties to be most readily and accurately inferred. I don't know whether or how to model the presence of this information in the domain description.

A.3.4 Entity Properties

An entity property may be static (not changing over time) or dynamic, and may belong to a single entity or relate several entities. All entity properties are either observed through

the occurrence of events, or are given by some data assumed available to the system.

Entity properties obtained from events are regarded as being *defined* by those events, so they need not be designated in their own right. For each event, the change to the entity properties will be given.

Then, there are indicative assertions that should be added. These fall into two categories: redundant assertions that follow from the definitions and which serve as useful checks—for example, that an ambulance cannot be at a location and a hospital at once, and that a resource cannot belong simultaneously to two resource classes—and those that add information to the description—for example, that an ambulance will only park at a lot to which it was previously assigned when commissioned.

Entity properties are modeled relationally rather than logically. So a property that logically would have been a monadic predicate becomes a set. To model whether a resource is available, for example, instead of a predicate

```
available_pred : Resource -> Bool
```

we declare the set

```
available: set Resource
```

which is simply the set of resources for which the predicate is true. The motivation for this is two-fold: first, expressing properties in this manner makes it much more convenient to state constraints between properties (using relational operators); second, it allows constraints to be checked for consistency.

- *Topology (static, from given data)*

```
loc_sector: Location -> Sector
lot_sector: Lot -> Sector
loc_hospital: Location -> Hospital
```

- *Resource Classification (dynamic, from Resource events)*

```
class: Resource -> ResourceClass
```

- *Resource and Ambulance Placement (dynamic, from Resource/Ambulance events)*

```
res_hospital: Resource -> Hospital
res_ambulance:
   Resource -> Ambulance available: set Resource
serving_from: Ambulance -> Lot
```

- *Call Recording (dynamic, from call event)*

```
received_at: Call -> Time
```

- *Ambulance Tracking (dynamic, from ambulance events)*

```
at_location: Ambulance -> Location
at_hospital: Ambulance -> Hospital
parked_at: Ambulance -> Lot
```

- *Call/incident Classification (dynamic, from call/incident events)*

```
reported: Call -> Incident
where: Incident -> Location
when: Incident -> Time
needs: Incident -> (ResourceClass -> Nat)
```

A.4 System Description

This section describes events and entity properties that are not observable in the domain, because the events are system-generated, and the entity properties require knowledge of these new events.

A.4.1 Events

```
dispatch_ambulance (a: Ambulance,
                    dest: Location)
```

the key function of the system is to generate this event.

```
notify_hospital (rs: set Resource,
                 h: Hospital)
```

notify hospital that resources should be put aside for the imminent arrival of casualties of an incident.

```
instruct_ambulance_to_move
   (a: Ambulance, from, to: Lot)
```

to optimize resource placement, the system will generate instructions to move ambulances that are in service between lots.

```
instruct_resource_to_move
   (r: Resource, from, to: Ambulance)
```

likewise, resources can be moved between ambulances (but not between hospitals and ambulances).

A.4.2 Entity Properties

- *Derived Ambulance Tracking*

```
sent_to: Ambulance -> Incident
serving_at: Ambulance -> Incident
delivering_to: Ambulance -> Hospital
```

- *Resource Allocation*

```
allocated: Resource -> Incident
```

  when an ambulance is dispatched to an incident, its resources become allocated to that incident, and similarly for hospital resources.

A.5 Requirement

The basic requirement is that the system generate events to optimize the delivery of resources to incidents (by taking resources to patients and by taking patients to resources). It is not at all clear what the measure to be optimized should be. An arbitrary (and not totally implausible) measure is the average time taken to resolve an incident (dispatching of ambulance to delivery at hospital), ignoring the time spent at the incident itself.

It is tempting to cast the requirement directly in terms of the Call and Incident events, but this would be a mistake. The behavior of the system can only be judged given some knowledge of how dispatch and resource movement events subsequently effect the motion and availability of ambulances and resources. This knowledge cannot be formalized, of course, since the domain is "biddable" and not "controllable": an ambulance driver can choose to ignore an instruction, for example.

A.6 Specification

The specification of the system is a description of its interface. Unlike the description of domain properties, coming up with the specification involves a considerable degree of invention. The choice of interface should not be equated with the client's requirements; a client that places significant constraints on the interface is participating in the design of the system (and is likely to have forced premature design decisions). For these reasons, our specification must be even more speculative than our domain description.

A.6.1 Simplifying Assumptions

This specification is very crude and does not match the domain description properly. It also makes a number of further simplifying assumptions.

- No attempt is made to optimize the distribution of ambulances by predicting demand. Ambulances are sent to incidents as needed, and returned to their originating lots.
- The LAS CAD system runs at a single central location. Although its implementation is likely to be distributed (because there are several operators), we shall ignore complications that would arise from having the central control station interact automatically with local stations. Instead, the central control stations generates dispatch orders that are routed to appropriate local stations, and perhaps receives ambulance location events from local stations, with the local stations regarded as part of the system's environment and not components of the system itself.

A.6.2 Basic Strategy

The interaction of the system with its operators is clearly a fundamental design issue. We shall make the following naive design decisions. When a call comes in, an operator enters details into the system. This is a `call_report` event. The record of the call cannot be changed subsequently. At the same time, on receipt of a call, the operator keys in certain incident details. These elicit a query on a database that generates candidate incident records. If none match the call (in the operator's judgment), or no incident record is retrieved by the system, a new incident record is created (as a form filled in by the operator). If an incident report does match the call, the operator can update it according to the new information given by the caller. This may include not only adding resource needs (because the incident is worse than first expected), but also reducing needs, either because of reassessment of the event, or because information in the new call suggests that a previous call was incorrectly matched to an incident.

The system automatically batches up incident reports and presents them to a supervisor, who determines whether dispatch orders and hospital notifications that are generated automatically are appropriate.

The system will track incidents and raise warnings if ambulance location events (and other incident tracking events) indicate that the incident is not being attended to as anticipated.

A.6.3 Context

The context of the system defines the form of its interface: i.e., how it interacts with the environment.

- *Incoming Events (detected)*
  - ambulance locations
  - ambulance status changes (e.g., arrived at incident, arrived at hospital)
- *Outgoing Events (generated)*
  - ambulance dispatch orders
  - hospital notifications
- *Operator Interaction*
  - call reports
  - incident queries
  - incident reports
  - incident updates

A separate view of the system handles slow-changing updates to the map of London, division into sectors and locations, the allocation of resources to hospitals and ambulances and the acquisition of new ambulances.

A.6.4 Non-functional Properties

- *Security:* The updating of incident reports and the vetting of dispatch orders should be carefully controlled.
- *Resilience to Failure:* Very high availability of obviously necessary. The system should produce a paper trail so that operators can easily switch to manual operation. In the event of failure, restart time should be small.
- *Reliability/robustness:* Loss or corruption of incident reports, or failure to track the handling of an incident correctly, is very serious.
- *Maintainability:* It should be possible to replace the user interface without undermining the structure of the system. Scheduling algorithms should be replaceable. The system should be amenable to a variety of extensions to its function (such as dynamic tracking of ambulance position, rule-based matching of incident reports, and so on).