Report on architectural decisions and assumptions in the LASCAD system

Paco van Beckhoven, Mats Stijlaart, Floris Velleman,
Olav Trauschke, Robin Kulhan, Ardavan Ghaffari,
Nico de Groot, Bouke Grevelt, Spiros Tzavaras

March 13, 2016

"Whilst understanding fully the pressures that the project team were under to achieve a quick and successful implementation ... knowing that there were so many potential imperfections in the system [6, p. 4002]"; a statement that unfortunately summarizes the London Ambulance Service (LAS) case to quite a frightening degree of accuracy. What architecture was used during this project and to what extend could things have been different if changes were made to the architecture? In this document we report on the architectural analysis that has been done in regards to the LAS case.

The authors assume that the reader is familiar with the LAS case. If this is not the case, then we advise reading our previous report "Failure of the London Ambulance Service, In-depth analysis of the quality assurance and tender process for the London Ambulance Service incident and how modern methodology might have saved the project".

The first chapter discusses the dominant decomposition used in the LAS case. It also talks about how the dominant decomposition relates to the organization. The following chapters analyze three major failures in the system from an architectural point of view. The chapters show an analysis of where the applied architecture helped and what changes in architecture the project could have benefited from.

Architectural Decomposition

Dominant System Decomposition

The documentation we analyzed makes no direct mention of the design methods and architectural patterns applied in the LAS case. Therefore, in determining the architectural decomposition certain assumptions will be made based on the information that was provided by the documentation.

We believe that the dominant decomposition of the architecture was component-based. The different components in the system (as shown in Figure 1, appendix A) were created by different companies:

- The AVLS system was created by Datatrak.
- The CAD software was created by Systems Options.
- The RIFS system was created by SOLO.
- Network infrastructure was delivered by Apricot.

This does not directly provide evidence for a component based architecture, as members of different companies could have worked together in teams with a single responsibility (for instance a service in a service oriented architecture). However, we have found no evidence of the existence of such teams, which is something we believe would have been mentioned in the official inquiry report [6] if it had existed.

We also considered expert based composition to be the dominant decomposition due to each company working on a part of the system of which they were experts. However, as with a service oriented architecture, one would expect to see teams with members from different companies in an expert based decomposition. The Computer Aided Dispatch (CAD) software for instance required expertise from all companies as it had to interface with all the different systems. If an expertise based composition had been used, multiple teams with a specific expertise would have worked on the CAD software. As SO (which was an expert on GIS systems [7] but not an expert on integration projects of this size [6, p. 3078]) was solely responsible for the CAD software, we conclude that the expert based decomposition was not the dominant decomposition in the LAS project. We thus find it reasonable to assume that a component-based architecture has been used.

Although we believe that at top level the dominant decomposition of the architecture was component based, we have identified other decompositions that were dominant within specific components. This is in line with the content of a paper written by Kramer and Wolf where they stated that "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 "[3, p. 26]. In the following paragraph we will look at the dominant decomposition of the architecture in the file server component.

File server

The LAS architecture contained a central file server, where all the data from the different components of the system, such as the location of the ambulances, was stored. Within this component, we identify the shared data pattern as the dominant decomposition. When this architectural pattern is used, it should be taken into account that the storage may be a performance bottleneck and a single point of failure [4, chapter 13]. The crash of the system on November 4th 1992 shows how the file server was a single point of failure in the LAS architecture as a memory leak in the software on the file system lead to a complete lockup of the system.

System Communication

As mentioned in the Architectural Decomposition, we concluded that the component-based architecture was dominant. This aspect strongly returns in Figure 1 located in appendix A, which shows the communication lines between the different entities in the system.

We believe the component-based decomposition brought some advantages, such as the possibility to replace components. LAS decided to use hardware from the previously failed project [3, p. 8]. We assume this requirement is one of the reasons a component-based architecture was used. A component implementing a certain interface would be introduced to encapsulate the old hardware and corresponding software. This gave LAS the possibility to easily replace the legacy hardware as only the implementation of the interface would have had to change. During the tender process the consortium decided to change the supplier for the AVLS system [6, p. 3073]. Suppose that SO already started on the development of the CAD system, then the use of component interfaces reduced the amount of extra effort (changes flowing through the system). It is important to note that this advantage only applies if the interfaces are well defined. If implementation details are reflected in the interface one still needs to change the components that use this interface when switching to a new implementation. Another limitation on interfaces is that programmers need to reconsider them when new functionality is introduced, resulting in the loss of the purpose of having interfaces at the first place.

Another advantage of the component-based decomposition is the ability to replace certain components with stubs. A problem was the readiness of the RIFS hardware [6, p. 3074]. The implementation of the integration with the RIFS was blocked as the hardware was not completely finished. As different components depended on this functionality it was a crucial part of the system. In order to prevent this from blocking the work on other components SO started with the component interface and created a dummy implementation. This allowed them to continue to work on the other components instead of just waiting until the hardware was finished. We believe this also helped when LAS decided to go for a phased implementation of the system. By replacing components that were not ready or had a low priority with stubs SO could focus on the most important components.

We suspect SO created a deployment view in which components and their relation were specified. This components overview allowed developers of a specific component to identify how components were related to each other. For example, if a change to a component interface was made one could quickly identify which other components required a change. Additionally it could have been beneficial for the planning of the test process. For example, to identify which components could be tested separately.

One of the problems of the LASCAD system was that the AVLS systems frequently did not transmit the (correct) location of ambulances. This resulted in incorrect allocations by the system such as not sending the closest vehicle to an incident [6, p. 4022]. "The Datatrak system as it was installed was likely to be as reliable as any AVLS system ever could be in an urban environment such as London at that time" [6, p. 3133]. The knowledge that the AVLS's did not always submit (correct) locations was part of Datatrak's expertise. SO which depended on the location data had no knowledge of this problem. This was a consequence of the distribution of the work between the different companies. The work was distributed in such a manner that the companies worked isolated from each other, resulting in a lack of alignment between SO and Datatrak. SO's dependency on Datatrak's data should have indicated that knowledge sharing should have taken place prior to development on this component. This lack of knowledge could have been identified with a (technical) use case diagram in which the AVLS would be modelled as one of the actors. Assuming SO would have made this diagram, it would still not contain the alternative scenario in which no or inaccurate data would be provided. However, if they used this diagram in their communication with Datatrak, Datatrak could have pointed them on the missing scenario. This diagram should have been part of the process to facilitate communication between the involved parties.

When SO decomposed the architecture in different components a problem arose, which was not identified until the system ran in production. There was no clear specification on the input validation that had to be done for each component. As stated in the official report "There had been no attempt to foresee the effect of inaccurate or incomplete data available to the system (late status reporting/vehicle locations etc.)" [6, p. 4001]. The logical view of the 4+1 model would have helped as this problem could have been identified with the use of sequence diagrams on a component level. Using a sequence diagram one can trace the user input from the client through the different components. The fact that no validation step was present should have indicated the lack of validation.

Backup

"The LAS wanted adequate fallback commensurate with the need to maintain a constant high level of service" [6, p. 5038]. The idea was to implement a backup for the file system, consisting of two fallback file servers [6, p. 3132]. There is no information about when the decision was made to create a backup system but we assume this was at the start of the project. This could have been during the requirements elicitation when there was a phase dedicated to specifying the systems requirements [6, 2018a].

However, "the fallback to the second server was never implemented by Systems Options as an integral part of this level of CAD implementation. It was always specified, and indeed implemented, as part of the complete paperless system and thus arguably would have activated had the system actually crashed on 26 and 27 October 1992" [6, p. 4041]. This has lead us to assume that the fallback servers and the usage of printers were not thought out in the architecture. "The failure of the fallback procedures arises as a consequence of what was believed at the time to be only a temporary addition of printers. The concept of the system was that it would operate on a totally paperless basis. Printers were only added, as a short term expedient, in order to implement at least a partial system at the originally planned implementation date of 8 January 1992" [6, p. 4040].

The problem with the fallback servers and the addition of printers could have been prevented with a deployment view on the system's architecture. The deployment view could have shown how the different hardware components of the system should have been connected to each other. This could have helped to spot defects in the system when the printers were added. Besides a better deployment view, a better logical view on what should happen when the file servers failed and how they should have interacted could have contributed to better testing of the file servers. The logical view could have been represented by a sequence diagram for instance. Unfortunately, we did not find a deployment view or a logical view in the literature. However, Figure 1 in Appendix A may have been developed before the project and is much like a deployment view, but it does not mention the file servers. The view in the appendix may have been used to facilitate the communication between the different companies developing the software, which may decrease the understanding of the purpose of the backup system in the development teams.

This lack of understanding of the backups purpose, may have been the reason the system was delivered without the backup having been tested completely. "The testing phase shows that there was no attention for the backup system or worst-case scenarios" [6, pp. 3085,3132]. "There is no record of the backup having been tested and there can be no doubt that the effects of server failure on the printer-based system had not been tested. This was a serious oversight on the part of both LAS IT staff and SO and reflects, at least in part, the dangers of LAS not having their own network manager" [6, p. 4041].

Critical analysis of the architecture, if it had been done correctly, might have helped to find out the purpose of the backup which could have not been clear for SO. If flaws in the architecture had been found, they could have been solved and a better backup system may have been developed.

Besides architectural views that could have helped prevent the failure of the system, better usage of a project management methodology could have made sure a full integration test was accounted for. This could have helped in finding the defects in the backup that lead to its failure. At the start of the project the decision was made to work with PRINCE, because this was a government standard. But none of the members of the LAS team nor SO had prior experience with PRINCE [6, pp. 3068,3078]. This decision was clearly not well thought out.

We can conclude there was an architectural decision to have a backup system in place, but there were no architectural views that could have prevented defects regarding printers and fallback servers in the system. The fallback servers were also not executed properly in the implementation phase and not tested properly during the testing phase. In the process, only partial integration tests were performed. We think testing the system as a whole would have avoided the failure of the backup. Finally, we think more architecture views, more critical analyses, and better use of PRINCE would have helped to find (and solve) defects before the development started which could have helped to avoid the failure of the backup system.

Graphical User Interface and Performance

We found that decisions related to the Graphical User Interface (GUI) and its performance influenced the project. The end-users had to cope with errors presented on their workstations [1, p. 704] and a slowly responding GUI [6, p. 3126]. The fact that end-users were not able to do their work as intended eventually lead to a snowball effect that caused the disaster on the 26th and 27th of October 1992 [6, 1017w].

The used literature did not mention non-functional requirements for the GUI, nor the consideration of performance issues. We believe the issues in the GUI were related to the lack of relevant architecture. There is no information that SO included alternative cases in the architecture, such as the presentation of errors in the GUI

For instance, usage of architectural views to facilitate discussion and validation of the GUI with stakeholders (ambulance staff and call takers) would have made identification of GUI related problems more likely. There are views and methods which could have contributed here, such as user interface views. Examples of these kind of views are wireframe views and the Paper-prototyping method [8][5].

The wireframe views would have allowed SO to "show user flows between views or pages [8, p. 7]" and help the stakeholders to understand the "content and information hierarchy as much as the the structure, functionality and behaviour" [8, p. 9]. The structure of the GUI and the flow between the different screens could have been explained or pitched to the stakeholders. This would allow SO to identify confusing screens and flows that were unclear and therefore sensitive to human mistakes.

The Paper-prototyping method could have been used as an interactive way for SO to find a more optimal and suitable structure for the layout with the ambulance staff [5, p. 345]. These methods may have changed the process in such a way, that there would be more communication on the non-functionals between SO and the stakeholders. It may also have removed the lack of ownership the users experienced when using the system [6, 1007o].

Systems Options chose to use a programming tool that offered increased implementation speed at the cost of execution speed [6, p. 3128]. Our opinion is that this choice is rather remarkable. First, as the LAS was the largest ambulance service in the world at the time, the LASCAD would require high performance. Secondly, the system developed prior to the failed case was abandoned after performance criteria could not be met [6, p. 2017].

Although the performance issues were addressed by LAS, these issues intentionally received little attention by SO [6, p. 3128]. For example, the slow response of the software was one of the most reported issues [6, p. 3126]. This can be traced back to a series of poor design choices.

LAS had two options available: either use a graphical presentation or a text based presentation. LAS chose to use a GUI and then SO used Visual Basic to produce the screen dialogues as mentioned in the official report [6, p. 3128]. Visual basic was at that time an uncommon choice for mission critical systems:

"Visual Basic is primarily used for prototyping and development of small, non-mission critical systems. The performance speed of visual basic applications is not fast. Filling screens with Visual Basic applications takes time measured in several seconds. In order to overcome this CAC staff preloaded all the screens they were likely to use at the start of a shift and used the Windows multitasking environment to move between them as required. This placed great demands on the memory available within the workstations, thus reducing performance, and led to a surplus of "clutter" on controllers' screens" [6, p. 3128].

Satisfying a quality attribute such as response time for a mission critical system like LASCAD is crucial. The architecture was lacking design methods. This could lead to the negligence of non-functional requirements. We think that if SO would have applied a goal-oriented architecting (GOA) approach, the problem with slow response times could have been alleviated to some extent. In this approach candidate designs are explored, evaluated and selected based on trade-off analyses in consideration of non-functional requirements [2, p. 92]. SO should have attributed more weight to performance using GOA when a choice was made between a graphical and textual interface, or proposed a different GUI language/framework that mitigated the risk of performance issues. But it seems ease of operator use was favored instead.

Conclusion

We found that the dominant architectural decomposition used in the LAS case was component based. A major advantage of this architecture was that LAS had the option to only use some of the components when it became apparent that the full system did not function as expected. We believe a major drawback of this architecture can be found in a lack of communication between the teams working on different components. An example of this is the LASCAD software not being able to deal with imperfect ambulance location data, even though the fact that location data can never be perfect is common knowledge for an AVLS developer.

The previously mentioned decommissioning of part of the project after it was clear that the complete system did not work as expected is the only example of changes in the architecture during or after development. We found that more architecture would have likely been beneficial here as the backup solution was not changed to fit the new system architecture and therefore did not function when it was needed. Overall we conclude that additional architecture would likely have prevented some of the major malfunctions in the project.

We come to conclude that additional architecture would have improved the project result. Specifically the use of:

- Sequence diagrams and a deployment view; this would have improved inter-team communication.
- A wireframe view and paper prototyping; these would have likely improved the usability of the GUI and implementation guidelines.
- Stating that only 'proven' technologies could be used would have likely improved GUI stability.

References

[1] Paul Beynon-Davies. “Human error and information systems failure: the case of the London ambulance service computer-aided despatch system project”. In: *Interacting with Computers* 11.6 (1999), pp. 699–720.

[2] Lawrence Chung et al. “Goal-oriented software architecting”. In: *Relating Software Requirements and Architectures*. Springer, 2011, pp. 91–109.

[3] Jeff Kramer and Alexander L Wolf. “Succeedings of the 8th international workshop on software specification and design”. In: *ACM SIGSOFT Software Engineering Notes* 21.5 (1996), pp. 21–35.

[4] Rick Kazman Len Bass Paul Clements. *Software Architecture In Practice,* 2012.

[5] Carolyn Snyder. *xt*. Morgan Kaufmann, 2003.

[6] London (United Kingdom); South West Thames Regional Health Authority. *Report of the inquiry into the London Ambulance Service*. 1993.

[7] WINGS GIS by Systems Options Ltd. `http://web.archive.org/web/19970630110747/http://www.systems-options.co.uk/techsumm.html`. (Visited on 03/08/2016).

[8] UXPin. *The Guide to Wireframing*. `https://studio.uxpin.com/ebooks/guide-to-wireframing`. (Accessed on 08/03/2016).

Appendix A - Viewpoints

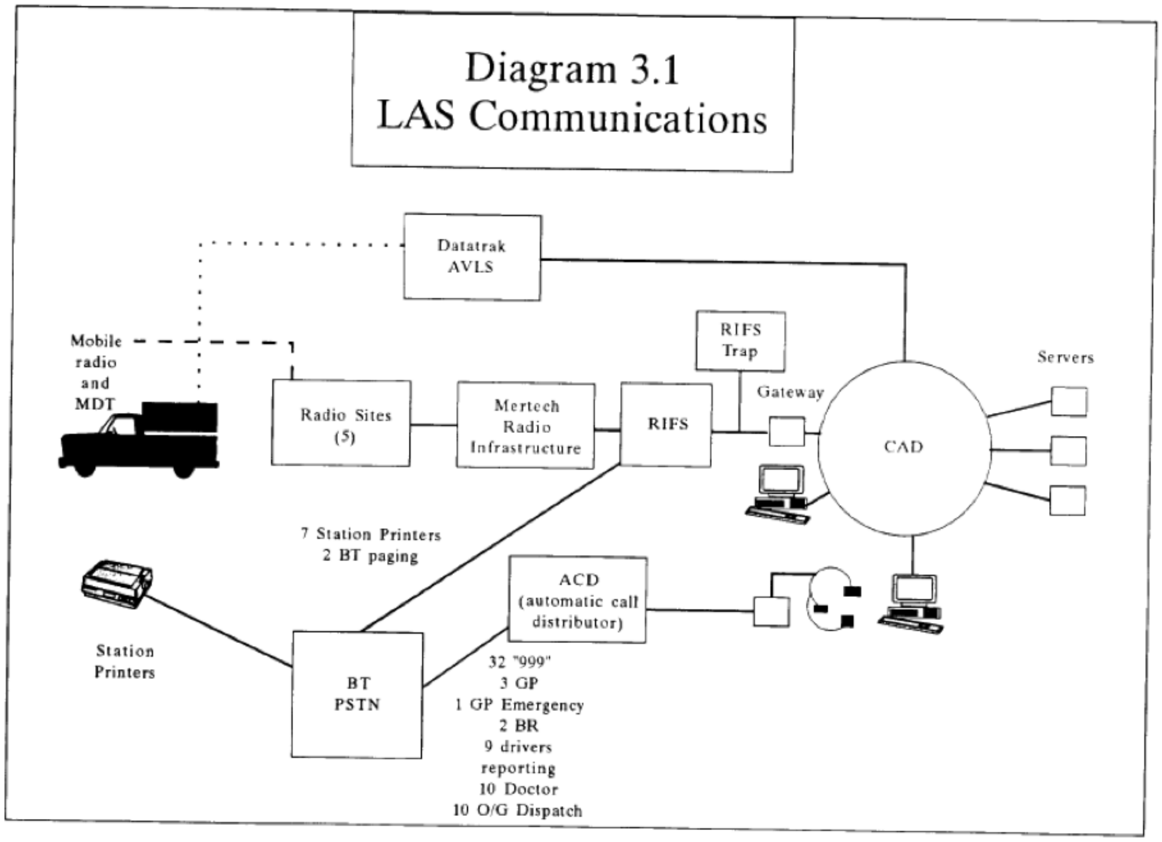

Figure 1: Diagram 3.1 from Report of the inquiry into the London Ambulance Service [6, p. 27]

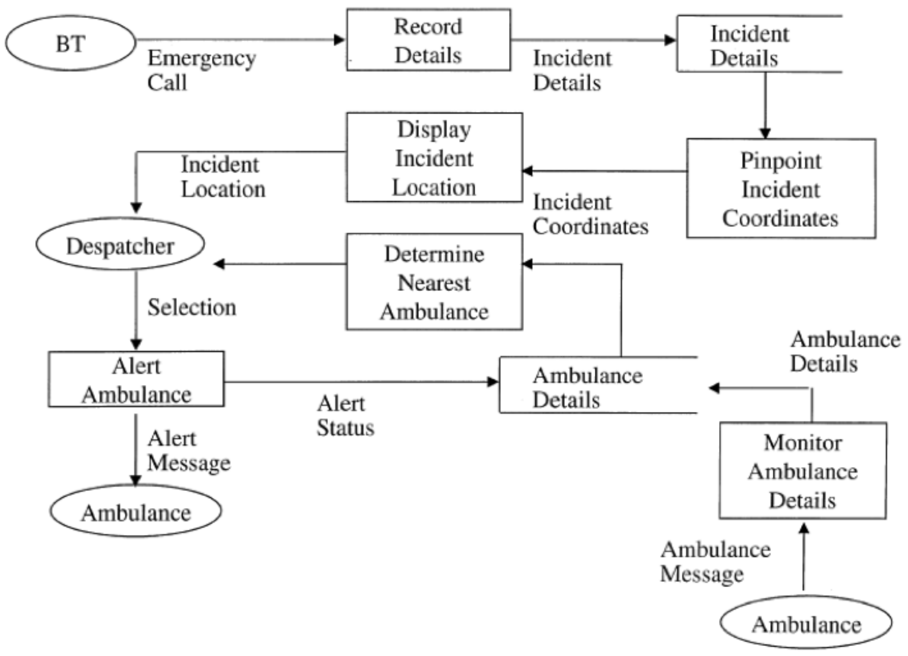

Figure 2: From Human error and information systems failure: the case of the London ambulance service computer-aided despatch system project [1, p. 707].