Architecture for Large-Scale Innovation Experiment Systems

Ulrik Eklund and Jan Bosch
Software Engineering Division, Dept. of Computer Science & Engineering
Chalmers University of Technology
Göteborg, Sweden
Email: ulrik.eklund@chalmers.se, jan.bosch@chalmers.se

***Abstract*—Business and design decisions regarding software development should be based on *data*, not opinions among developers, domain experts or managers. The company running the most and fastest experiments among the customer base against the lowest cost per experiment outcompetes others by having the data to engineer products with outstanding qualities such as power consumption and user experience.**

Innovation experiment systems for mass-produced devices with embedded software is an evolution of current R&D practices, going from where innovations are internally evaluated by the original equipment manufacturer to where they are tried by real users in a scale relevant to the full customer base. The turnaround time from developing and deploying an embedded product to getting customer feedback is decreased to weeks, the limit being the speed of the software development teams.

The paper presents an embedded architecture for realising such a novel innovation experiment system based on a set of scenarios of what to evaluate in the experiments. A case is presented implementing an architecture in a prototype in-vehicle infotainment system where comparative testing between two software alternatives was performed.

***Keywords*-embedded software; software architecture; product development**

I. Introduction

Innovative ideas for embedded products are typically collected and prioritized during the roadmapping and requirement management process as part of the yearly release cycle, which usually is determined by manufacturing concerns of the hardware. Feedbacks on innovations from real customers are collected only on new product models, if collected at all.

Since more and more embedded products also are connected (e.g. [1]), it is conceivable to develop, deploy and measure usage on new software in iterations which lengths are determined by the speed of the software development teams instead of the setup of the manufacturing process, going from years to weeks. Such an innovation experiment system would utilise feedback from real users in a scale comparable to the entire customer base. This requires an embedded architecture in each product together with an infrastructure capable of collecting and analysing the data.

The driver for having such an innovation experiment system is that business and design decisions should be based on *data*, not opinions among developers, domain experts or managers. The company running the most experiments among the customer base against the lowest cost per experiment outcompetes the others by having the decision basis to engineer products with outstanding customer experience.

The main contribution of the paper is architecture to support innovation experiment systems for mass-produced devices with embedded software. The architecture consists of a set of scenarios and the mechanisms that implements these. In addition, a case implementing the architecture and performing comparative testing between two software in the context of prototype vehicle HMI is presented.

II. Context and Problem Statement

The approach of having the entire R&D process as an innovation experiment system has been applied to areas where software is provided as a service in a cloud-computing environment, an example being Intuit [2]. Driving the R&D process as an innovation experiment system in the embedded domain is a novel approach, and the architecture for the products must facilitate the short development cycles of defining experiments, develop software, deploy it and evaluate the results.

A. Innovation Experiment Systems

Developing software in an innovation experiment system is different from development approaches for traditional embedded software. First, it frequently deploys new versions focusing on continuously evolving the embedded software in short cycles of 2-4 weeks, as seen in Figure 1. Second, the design decisions are based on customer usage data throughout the entire development process. Third, the goal of the development process is to test as many innovations as possible with customers to enhance customer satisfaction and, consequently, revenue growth. R&D in this context can best be described as an experiment system for new innovations [2].

The innovation experiment system focuses on incremental innovation according to the framework defined by Henderson and Clark [3]. The short iterations of experiments are aimed to refine and extend established designs. Improvement occurs in individual software parts, but the underlying design concept remain mostly unchanged [3], even if there is nothing that prohibits the evaluation of e.g. architectural innovations as well.

978-0-7695-4827-2/12 $26.00 © 2012 IEEE
DOI 10.1109/WICSA-ECSA.212.38

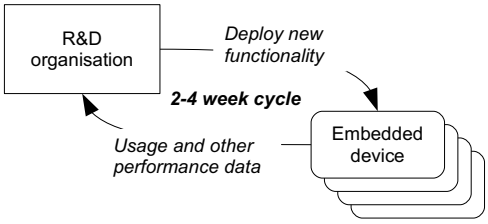

Figure 1. Overview of the Innovation Experiment System with the iteration of experiments.

B. Mass-Produced Embedded Systems

Today original equipment manufacturers (OEM) of products with embedded software range from focusing on efficient manufacturing of products with the software as difficult necessity to seeing software as a key business differentiator. Software is often an enabler for new innovations in embedded systems, for example in cars [4], and marketed innovative features are often realised by software. In many embedded domains the amount of software increase exponentially over time [5].

A common development approach for embedded systems is using a traditional stage-gate process, where the gates are driven by decisions on investment in the manufacturing of the product, i.e. driven by the hardware. The finalisation of software artefacts often correspond to process gate progression, e.g. user requirements, system requirements, software architecture, component requirements, and software implementation, i.e. a waterfall process even if the artefacts are updated as the project progresses.

We define the domain of mass-produced software-intensive embedded systems by four characteristics:

- Deep integration between hardware and software for significant parts of the functionality
- Strong focus on manufacturing aspects of the product in the development (e.g. by development process gates)
- Strong supplier involvement in the form of subcontractors
- Some elements of the system realise safety-critical functionality

Examples of mass-produced embedded products include cars and trucks, washing machines and other home utensils, sewing machines, printers and copying machines [5]. We will hereafter give general examples from the automotive industry since cars are arguably the most complex product of this category, both in terms of conflicting requirements and subcontractor relationships.

C. Research Problem

Innovation experiment systems are a new concept in the embedded domain and thus there are no architectural blueprints available, either from the research community or in industry. The research presented in this paper aims to explore implications for the architecture of the embedded software when products are built as a large-scale innovation experiment system. The research question is thus:

> What are the software architecture principles to realise a large-scale innovation experiment system of mass-produced embedded systems?

III. Goals of an Innovation Experiment System

The desired goal is for development teams to deploy updated or new software and collect data based on real customer usage instead of opinions inside the R&D organisation. Ideally these data would be directly related to OEM income revenues, such as increase in sold products or purchases of related services, but a direct correlation might be difficult to establish for products which have a one-time payment from customer to manufacturer. The next best thing is if the data can be connected to the perceived customer value of product. This would create a lasting value for customer since the product would feel "fresh" longer and indirectly affect brand perception and loyalty.

The simplest experiment cycle would be:

1) The team(s) defines the experimental problem and produces new software which implements a feature thought to improve a user perceived quality. In this case there is just one factor with two levels, present and new implementation.
2) The response variable(s) are selected and implemented
3) The software is deployed to a statistically relevant number of devices, with unmodified devices being the control.
4) The software is run for a period of time.
5) The data is uploaded and analysed through the infrastructure.
6) The development team draws conclusions about the new software and decides whether to develop the new software design further, keep it as it is, or to drop it and revert to the unmodified software.

A. Experiment Scenarios

The initial focus of the innovation experiments would be on user interaction, but should be extended to other things, e.g. power consumption or precision in control systems. The experiments supported by the architecture could for example answer the following about real world usage, i.e. these are potential experiment response variables:

- How long does it take to ...
- Which of ... is most often used/accessed/...
- Identify behaviour that is not intended, e.g. menu selection followed by "back" indicates that the user made a mistake.
- Are there any features that are not used?
- Be able to evaluate competing designs based on the answers above, i.e. A/B testing (AKA. split testing).

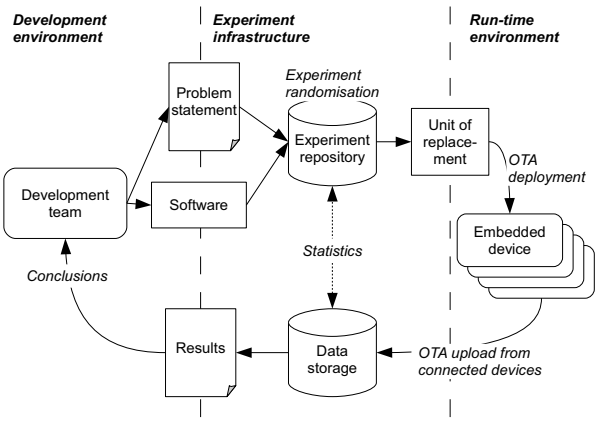

Figure 2. The infrastructure enabling the innovation experiment system.

Depending on the embedded domain there are other data that are interesting to get measurements on. In the automotive domain some examples could be real-world fuel consumption, how often active safety systems are activated to avoid accidents, or electrical power consumption in different driving situations. The architecture should support comparative or A/B-testing, where qualities of two or more, different application delivering similar features are tested against each other.

IV. Architecture for Experiments

The embedded devices is only one part of the innovation experiment system, the other two being the development environment and the experiment infrastructure, as seen in Figure 2. The experiment infrastructure allows developers to deploy new software and collect data how it behaves in areal-world settings being used by actual users. The infrastructure support deployment of software experiments and collection of data over-the-air on a scale comparable to the entire customer base, for an automotive developer this means devices in the order of 10^5. The infrastructure supports with automated randomisation and factorial designs [6] sufficient to draw statistical conclusions from the experimental scenarios.

A. Unit of Replacement

The unit of replacement is constrained by what the infrastructure can handle both in terms of deployment, e.g. size, and in terms of distinction between versions (it becomes a problem of numbers with devices in the order of 10^5 and unique units of replacements in the order of 10^2). But the size of the unit of replacement is also constrained by the microcontroller architecture, e.g. through the size of separately flashable sectors in an automotive electronic control unit (ECU). In Android the unit of deployment is typically the app, which is supported by e.g. the Android marketplace. In the download framework of the ROBOCOP

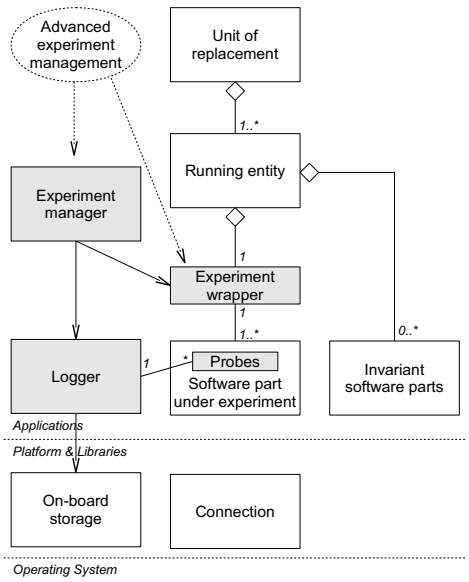

Figure 3. The architecture for managing experiments on-board.

architecture it is the component [7]. In the Automotive Open System Architecture (AUTOSAR) the software components are integrated for each ECU before delivery [8].

In many cases the unit of replacement is larger than what is desired upon from an experiment perspective, with the experiment ideally only encompassing a single user-initiated task or use case, while the unit of replacement can encompass many user tasks or features. In these cases the small part included in the experiment will be updated but all other parts are kept invariant.

The embedded devices must allow over-the-air deployment of new applications as well of updating the necessary parts of the software running the experiments.

B. Experiment Management Architecture

The experiment manager architecture, seen in Figure 3, supports the deployment of multiple experimental software parts to the same device and autonomously controls when to run which experiment, even allowing for local A/B-testing. Measurements and analysis is done on-board in real-time. The experiment scenario to be answered is implemented on the embedded device (i.e. how long does it take to ...)

The experiment software, besides the part under experimentation, consists of 4 parts: The replaceable software unit experiments deployed to each device, seen in Figure 3, consists of several parts besides the application being experimented upon, and the underlying embedded platform:

- The experiment manager which is responsible for deciding which experiment(s) to run, running the experiment, and assemble and analyse the collected data.
- The wrapper for the parts under experimentation, which enable the manager to decide in run-time which soft-

ware part to run, allowing for A/B testing or to revert to the default software part.
- The probe software fused with the application(s) that is evaluated.
- The logger which utilises the on-board storage.
- The on-board data storage provided by the embedded platform.
- The connection provided by the platform, e.g. 3G mobile connection.

The last two parts are part of the general embedded platform and is not specific to the innovation experiment system. The data logger can be general for all experiments and don't need to be updated in each experiment cycle, while the wrapper and probe software is very dependent on the application. The experiment manager is specific to the type of answers wanted. The actual probing can be done either through sampling or event-driven logging, depending on domain and type of application to be evaluated.

Embedded systems have a layered architecture, and the experiment manager thus needs to be in the same layer as the observed application, otherwise it does not know about what it is observing. If placed in a lower layer it can only draw conclusion about the hardware devices like "button is pushed" but don't know the meaning of a pushed button.

Collected and analysed data is stored on-board to be up-loaded for further analysis in batches, e.g. when the connection is not utilised for other things. This can be implemented as a push, i.e. the data set is uploaded when the measurement manager decides sufficient data is collected or the storage buffer is full. Or as a pull, the data is uploaded when requested for by the infrastructure.

C. Safety pattern

If the application experimented upon is potentially safety-critical the architecture must include mechanisms to mitigate any risks. If the experiment should run out-of bounds of what is considered safe it must be disabled and a fall-back, safe, version of the software application runs instead.

The safety pattern to satisfy this, seen in Figure 4, consists of 4 parts, and is based on well-known safety architectures for embedded systems [9]:

- The application(s) involved in the experiment.
- The monitor who evaluates that the application stays within safe boundaries.
- The fail-safe software, which operates within safe boundaries.
- The safety executive which controls the setup.

V. Case: Vehicle HMI

The case implementing an embedded architecture for an innovation experiment systems was a development project of a prototype to establish a proof-of-concept for some radically different development strategies compared to current software development in the automotive industry. The system

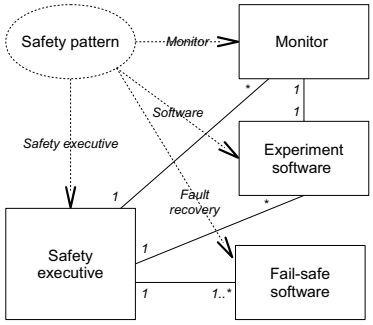

Figure 4. The safety pattern when the application software involved in the experiment is potentially safety-critical.

was an infotainment system based on an open platform, Android. The project was executed in an industrial setting, but the resulting system was not intended to go into mass production and be sold to customers.

The primary goal of the project was to establish whether it was possible to do feature development with extremely short lead-times from decision to implementation compared to present industrial projects, from a nominal lead-time of 1-3 years to 4-12 weeks. The short lead-times were accomplished by a small development team using Scrum from a consultancy firm with automotive software experience, which had a supplier relationship to Volvo car Corporation as product owner. Working software was continuously validated in "real" environments, i.e. the infotainment system was installed in both a driving simulator and real test cars and users evaluated the system during the project.

A. Experimentation

A user story in the first sprint covered measurement/logging how the user uses the system with the purpose to provide input to backlog and future sprints, in terms of tuning of current features and new ideas. In a subsequent sprint an A/B experiment was defined evaluating two layouts of the start screen of the infotainment system, implemented as two different launchers in Android. The system was mounted in a vehicle and test drivers were requested to perform some common task with the intent to measure which launcher "worked best".

Even though the test sample was too small to draw any conclusions, 7 drivers in total, the test drives showed that the on-board innovation experiment system worked as intended and collected the required data, which was then analysed off-board.

B. Architecture

The first generation of the system implemented a simplified experiment architecture from Section IV-B. The system used a logger in the same layer as the observed application, in this case the launcher, both residing in the Applications layer of Android, as seen in Figure 5. The data from the

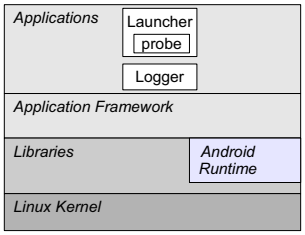

Figure 5. The experiment architecture of the prototype infotainment system.

logger was stored in a text-file with a batch upload of the data pulled by the developers. The logger kept track of the user's actions by storing different strings in the text-file, describing the actions that the user has performed, such as adding widgets to the workspace or starting an application. The logger was initiated from within the Android launcher at startup by creating the logger variable and call the constructor of the generic logger-class available on the Open Infotainment Labs platform. Since the software was only deployed to a single vehicle the management of the experiment was done off-line, i.e. only a single software variant was deployed at any time, and no wrapper was used.

The probe software consists of a single line of code at the appropriate place(s) in the launcher application, which defines the string to be stored in the log file, e.g:

```
2011-09-21, 10:14:20, DEBUG, Wdgt,
Analog clock started at screen: 3
```

The off-board analysis software (written in Python) determined the following based on the logged data:

- Calculates the time spent in each launcher screen
- How many applications are installed?
- What apps are launched?
- Which widgets are installed in which launcher screen?

VI. Conclusion

Innovative ideas for embedded products are typically collected and prioritized during the roadmapping and requirement management process as part of the yearly release cycle, which usually is determined by manufacturing concerns of the hardware. Feedbacks on innovations from real customers are collected only on new product models, if collected at all. We argue that business and design decisions regarding software development should be based on *data*, not opinions among developers, domain experts or managers.

Innovation experiment systems is an evolution of current R&D practices, going from where innovations are internally evaluated by the original equipment manufacturer to where they are tried by real customers. Since more and more embedded products also are connected, it is conceivable to develop, deploy and measure usage on new software in iterations which length is determined by the speed of the software development teams instead of the setup of the manufacturing process, going from years to weeks.

Since innovation experiment systems are a new concept in the embedded domain there are no architectural blueprints available, either from the research community or in industry. The paper explored necessary architecture principles to realise a large-scale innovation experiment system of mass-produced embedded systems. The result was a novel embedded architecture based on what to evaluate in the experiments, together with how the embedded architecture fits into a larger infrastructure. The evolution of such an architecture is described, going from simple measurements on a single software part to composite experiments involving different software parts involved in comparative A/B testing.

A case implementing the architecture in a prototype in-vehicle infotainment system where A/B testing was performed is also presented.

Acknowledgement

This work has been financially supported by the Swedish Agency for Innovation Systems (VINNOVA) and Volvo Car Corporation within the partnership for Strategic Vehicle Research and Innovation (FFI).

References

[1] T. Koslowski, "Your connected vehicle is arriving," *Technology Review*, Jan. 2012. [Online]. Available: http://www.technologyreview.com/business/39407/

[2] J. Bosch, "Building products as innovation experiment systems," in *Proceedings of the International Conference on Software Business*, ser. Lecture Notes in Business Information Processing. Cambridge, MA, USA: Springer, 2012.

[3] R. M. Henderson and K. B. Clark, "Architectural innovation: The reconfiguration of existing product technologies and the failure of established firms." *Administrative Science Quarterly*, vol. 35, no. 1, pp. 9–30, 1990.

[4] M. Broy, "Challenges in automotive software engineering," in *Proceedings of the International Conference on Software Engineering*. Shanghai, China: ACM, 2006, pp. 33–42. [Online]. Available: http://portal.acm.org/citation.cfm?id=1134285.1134292

[5] C. Ebert and C. Jones, "Embedded software: Facts, figures, and future," *Computer*, vol. 42, no. 4, pp. 42–52, 2009. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5054871&isnumber=5054856

[6] D. C. Montgomery, *Design and Analysis of Experiments*, 3rd ed. Wiley, 1991.

[7] J. Muskens, M. R. V. Chaudron, and J. J. Lukkien, "A component framework for consumer electronics middleware," in *Component-Based Software Development for Embedded Systems*, ser. Lecture Notes in Computer Science. Springer, 2005, vol. 3778, pp. 164–184. [Online]. Available: http://www.springerlink.com/content/f172600116688114/

[8] AUTOSAR, "Methodology," AUTOSAR development partnership, Auxiliary document 068, 2011. [Online]. Available: http://www.autosar.org/download/R4.0/AUTOSAR_TR_Methodology.pdf

[9] B. P. Douglass, *Doing Hard Time: Developing Real-Time Systems with UML, Objects, Frameworks, and Patterns*. Addison-Wesley, 1999.