Assessing Systems Adaptability to a Product Family

Mika Korhonen and Tommi Mikkonen
Institute of Software Systems
Tampere University of Technology
P.O.Box 553, FIN-33101 Tampere, Finland
{mika.korhonen, tommi.mikkonen}@tut.fi

Abstract

In many cases, product families are established on top of a successful pilot product. While this approach provides an option to measure many concrete attributes like performance and memory footprint, adequateness and adaptability of the architecture of the pilot cannot be fully verified. Yet, these properties are crucial business enablers for the whole product family. In this paper, we discuss an architectural assessment of one such seminal system, intended for monitoring electronic subsystems of a mobile machine, which is to be extended to support a wide range of different types of products. This paper shows how well the assessment reveals possible problems and existing flexibilities in assessed system, and this way helps different stakeholders in their further decisions.

Keywords: *Software product lines, product line architecture, adaptability, assessment*

1. Introduction

A pragmatic way to develop product families is to base them on a well-established seminal system. The possibility to use the seminal system as a basis for estimating many non-functional properties, like performance and memory footprint, for instance, is a definite advantage of such an approach. Another definite advantage is the available real-life experience on the required verification and validation effort. With precise input, the plan of the transition to product line architecting can also be more accurate.

What remains open with this approach, however, is the adaptability of the architecture of the seminal system to become a product line architecture (PLA) [3] for a range of related products. While concrete measurements obtained with the real system can be used to study the scalability of the system, issues related to variability needed for different products requires a different approach. Then, the focus is placed on intended modifications, and their relative cost and effort.

The goal of this paper is to introduce a real-life case where we were faced with the challenge of transforming a single product to a product line architecture. As a tool for estimating the adaptability of the architecture, we assessed the architecture with respect to several quality properties that reflect adaptability needed in product line phase.

The rest of this paper presents the discussion structured as follows. Section 2 describes the product architecture as it had already been implemented. Section 3 introduces the procedure we used for assessing the adaptability of the system. Section 4 discusses the input created for the assessment, and the evaluation of the input data in the assessment. Then, Section 5 provides the conclusions we made as the result of the assessment. Finally, Section 6 concludes the paper with some final remarks.

2. Pilot product

The environment and architecture, discussed in the following, are assumed to be applicable for the whole product line.

2.1. Environment

The physical environment of our application consists of mobile machines. Such machines include a wide range of products, e.g. vehicles

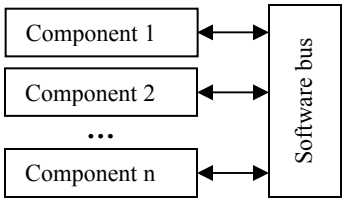

Figure 1. Architecture based on a software bus

including cars as well as construction, agriculture, military, and forestry machines, airplanes, ships, and railway engines, for instance. Many mobile machine types have some kind of engine, breaking system, transmission, suspension controlling, or weapons systems. These communicating subsystems often provide extra data from their status and current activity. In some cases subsystems contain control interfaces that can be accessed through some communication interface.

In addition to extremely complex individual subsystems, many (if not all) mobile machines contain some kind of a communication network (or networks) that connects the subsystems together. Information from the communication bus can be traced and collected for further use if needed.

In the pilot system, controller area network (CAN) [2] is used as a communication network within the vehicle. This network is widely used in automotives and factories and nowadays also in ships and buildings. In general, the main area of CAN networks is reliable real-time applications.

The main goal of the prototype application is to make it possible to trace and receive data from various data sources of the physical environment. An auxiliary goal is to decrease the amount of received data, so that only necessary data is stored for the later use.

2.2. Software architecture

The main philosophy of the architecture reminds physical device environment. In the software architecture the physical communication network and associated devices have been implemented with equivalent software modules. The resulting architecture is a mix of three well-known patterns; mediator, strategy and observer patterns [1]. In this paper, the combination of these three patters will be called *software bus* [4].

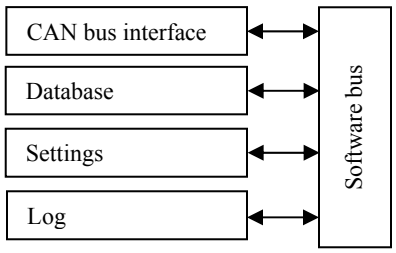

Figure 2. Available application framework

Selected architecture is an obvious result from the point of view of the problem domain, where similar elements can be found. The CAN network acts as a software bus and subsystems in CAN network can be compared to functionalities (components) of the application that are communicating to each other through software bus.

In our implementation the software bus routes extensible markup language (XML) messages, which eases the debugging and specification work, and decreases the dependency of different components, compared to communication with object (or component) references. The abstract architecture of such a system is illustrated in Figure 1, with components representing different kind of functionalities.

2.3. Component configuration

We will refer to a collection of different components as a *configuration*. The different features needed in different subsystems will then be implemented via configurations, and added on top of a default configuration, that contains the basic infrastructure for the real applications (Figure 2).

The default configuration has already been used as a basis for a real application. The application contains the components of the default configuration extending it with user interface, and data monitoring components. This application configuration is illustrated in Figure 3.

3. Assessing system architecture

Architecture assessments are a well-established mechanism for studying the adaptability of software systems. In the simplest case, one lists the intended modifications (or best estimates), and then, the modifications are

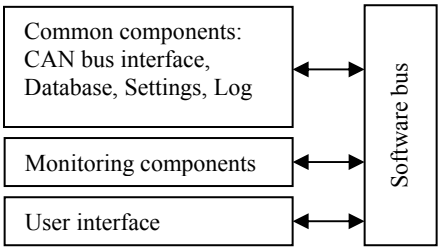

Figure 3. Implemented seminal system

reflected on the system architecture to reveal supported and problematic modifications.

In this section, we will introduce the practice of software architecture assessment. In addition, we will also give an insight to the assessment procedure that we chose to organize.

3.1. Software architecture assessment: An overview

Assessing software architectures is a practical necessity for ensuring that the designed architecture meets its functional and quality requirements. For instance, if there is a requirement for fault tolerance, the architecture should enable its implementation with e.g. replication. If no support can be found, it becomes questionable whether the requirement can be satisfied.

There are two basic categories of architecture assessments, those based on questionnaires and those based on metrics. Assessments based on questionnaires assume that the assessment team collects information regarding the expectations of the stakeholders of the system. This can be implemented with series of questions answered by the stakeholders. Questions can be predefined and reused in many different assessments virtually unmodified. However, with more detailed and domain sensitive questions, it is possible to obtain more accurate information. For instance, a question can be formulated as a scenario that defines a detailed change in the functions of a system. Answering this question then gives an idea about the flexibility of the system with respect to that particular function.

Measurements used for assessing software architectures contrast questionnaires in the sense that they are not about raising questions about the system but about producing answers to them. Usually they imply the development of simulations, prototypes or test environments that can be used for measurements. Moreover, there are some conventional metrics that can be applied at the architectural level. In general, however, the problem of measurements often is the lack of reliable data to measure.

3.2. Creating change scenarios

The assessment we decided to carry out is loosely based on available scenario-based architecture assessment methods [5], [6]. However, as we only had a limited amount of working time available, we decided to carry out the assessment in an agile fashion in two half-day meetings. Both meeting were attended by same people, which ensured smooth continuation during the second day. The number of participants in the assessment was ten, including two assessment experts, one architect, two software developers responsible for implementing the system, and one project leader. In addition, a group of software architecture experts participated.

The first meeting, lasting three hours, was used for introducing the assessment setting and the architecture of the system. This time, however, would not have been enough if the architecture of the system had not been familiar to the participants at an abstract level.

After the first meeting, all participants were given a task to think about different scenarios for identifying whether the architecture could be used as a basis for a software product line or not. One week was allocated to this task. The total number of scenarios that we were able to identify was 46.

The second meeting, lasting four hours, again gathered all the participants to a meeting. During the meeting, we went through the scenarios one by one, aiming at finding the different topic areas that the scenarios implied. Obviously, as scenarios were originally derived in isolation, many of them were combined for eased handling. Also, many of the scenarios were such that they could fit to several categories. Finally, when we were satisfied with the scenarios and their grouping to different topics, we reflected the scenarios to the actual architecture in order to detect any mismatches.

4. Evaluating scenarios

Input for the assessment was a collection of potential future scenarios that the architecture should adapt to. In the following, we describe the most important scenarios grouped into four categories; adaptability, configurability, reliability, and performance.

4.1. Adaptability

The purpose of adaptability scenarios was to assess how well our architecture would adapt if something were totally different in some specific product of the product line.

4.1.1. Scenario: Architectural requirement changes

This scenario covers a range of different detailed scenarios. For instance, we could consider the introduction of new message type to the software bus for some new special communication needs.

This kind of change would be derived e.g. from performance requirements. For instance, the requirements on processing of messages obtained from the software bus or memory footprint could be changed so that new types of protocols are used. For instance, XML might be replaced by a native protocol for performance reasons.

Also the internal architecture of the database component might be forced to change for some performance restriction, with the exception that we may have configurations where no database is present at all.

4.1.2. Scenario: New tasks, not only tracing

The system may be extended to cover high priority controlling of different tasks. One of these tasks could be controlling the physical devices that are connected to the communication bus. This would lead to a more complex interface to the CAN bus and would introduce a new component that would do the controlling calculations. Usually controlling calculation adds some real time requirements for the application, and which is however not expected from the application, until now.

4.1.3. Scenario: Changes in the user interface

Localization was considered as a necessity for an application of this type. This would cover both changes in the language as well as changes in the units used to describe e.g. tolerances (inches vs. millimeters). In addition, this may require changes on the amount of data to be shown on the user interface (UI).

If real-time UI updating is required, then the current report style UI philosophy is going to change and leads to some changes in the UI component. However these changes are not causing other changes in the other components of the system.

4.1.4. Scenario: Infrastructure changes

Different processor types may be needed when the tasks performed by the system change. This may depend on the type of vehicle we are developing. In addition to processor types, it is possible that the operating system or component technology is upgraded or altered. Similarly, when dealing with a totally different type of a product (e.g. a boat), it may be a necessity to change the infrastructure considerably.

4.2. Configurability

Configurability scenarios have been included in the assessment to see how easily it would be possible to introduce new functionalities to the application environment. Configurability is also discussed because the functionality changes in the assessed architecture are handled with the changes in the used component configuration.

4.2.1. Scenario: Multiple configurations in the same system

There could be multiple configurations that can be used in one machine, depending on e.g. the needed functionality or selected user profile. Additional concerns can be raised in cases where some selected configuration does not work. In situation like this the application should probably replace the failing configuration with the latest functioning configuration.

Changes in the configuration controlling causes at least changes to the software bus that maintains the active functional components. Configuration controlling might also introduce some new components to the software bus.

4.2.2. Scenario: Automatic configuration

In some cases, the application probably should configure itself automatically. In particular, this

covers issues where a special purpose hardware unit is added, and the application could react to that. This would result in a plug-and-play like configuration management. No such features have been included in the system in the current phase but maybe in future.

This kind of functionality can be easily added when it is needed.

4.2.3. Scenario: Configuration control upgrades

The option of using remote configuration facilities was proposed instead of in-place configuration that was implemented in the current system. In addition to creating a configuration, the scenario also covers a feedback mechanism that enables the transmission of the configuration status remotely. In the long term, this may also lead to runtime configuration controlling, where new configurations are loaded on the fly, not only when the vehicle is executing its startup sequence.

These and other configuration management requirements can be handled by some configuration management component or by the software bus.

4.2.4. Scenario: Erroneous configuration

When new components are introduced, there can be problems due to interacting components. Therefore, extra facilities are needed for checking the compatibility of different components included in the same configuration. This can obviously be checked either off-line when creating the configuration or on-line immediately before taking the configuration to use.

Similar problems can be encountered at run-time as well. However, this is much more difficult to detect, as the underlying hardware and the user can affect the run-time behavior.

Similarly to previous configuration scenarios this function can be separated to an independent functionality that uses configuration controlling capabilities of the software bus.

4.3. Reliability / Robustness

As mobile machines are usually embedded systems, reliability and robustness are important properties for them. In order to ensure that our architecture does not risk these properties, we decided to assess this issue separately from the rest of the system.

4.3.1. Scenario: Communication network problems

When using a remote connection to monitor a vehicle, it is possible that the external communication network fails. This will result in bursts of data to be transmitted, while potentially downgrades the options of controlling the vehicle. Obviously, the vehicle itself cannot recover from such problems.

Additional problems can be found from security. It may be a necessity to introduce a cryptographic functionality to secure the information the vehicle transmits.

These features can be located in the communication component or some other component that is working tightly with the communication component.

4.3.2. Scenario: Erroneous data

The reception of faulty commands from the user or from the CAN bus (or from other communication buses) can be a problem. Then, if overlooking the commands is not an option, the system should either panic or preferably return to the previous configuration that was functioning in accordance to specifications.

The best solution to recover from this problem would be the introduction of a component that monitors the correctness of communication messages between different devices and sensors attached to the bus.

4.3.3. Scenario: Breakdown

The most obvious problem falling in this category is the failure of the monitoring application, potentially leading to the corruption of the associated database or other ongoing tasks. The easiest way to improve this aspect would be the introduction of a scan at the beginning of the boot sequence that would rebuild the database and rescue the interrupted tasks in a case of problems. Also other resources should be monitored and rescued if possible and needed.

A runtime error in the computation of the monitoring application is a potential problem that we have not addressed in the current application. This could be improved by introducing a feature that would monitor the execution. Monitoring could be based on e.g. minimum and maximum values of a certain measurement (like condition monitoring). In XML-based communication, one

validity checking could be based on XML validity checks.

4.3.4. Scenario: Out of memory or disc space

When generating the database, it is possible to run out of memory or disk. A manager module could be added that monitors the amount of available memory and disk. Problems should obviously generate some kind of an indication to the user, potentially using remote communications link.

4.4. Performance

Performance of the prototype system has been acceptable. However, we had not studied the different implications of new configurations and tasks before the assessment. Therefore, we chose to treat performance as a topic of its own.

4.4.1. Scenario: N-fold increase on number of measurements

When the number of measurements increases, a more efficient processor or some distribution mechanism is required. The latter is eased by the use of software bus as the architecture of the system, as this component can be easily used for distribution of tasks as well.

4.4.2. Scenario: Configuration uses too much processor time

A potential problem in the system is that the current system optimistically tries to execute all its tasks. With a more dynamic configuration scheme, a situation may occur where too many tasks are configured for execution. For such cases, the introduction of a priority management feature may become a necessity.

5. Assessment results

The most challenging problems we were able to identify in the assessment were related to configuration. Online configuration controlling would require changes in some parts of the core of the application framework, as this option was simply overlooked in the first version of the system.

The additional configuration controlling component is also needed to the product line, if the configuration should automatically adapt to the current physical environment. This additional component is also the solution if the application should control multiple separate configurations, e.g. normal, test, and minimum configurations.

An additional source of problems was the handling of error situations. It is unclear to us, what part of the system extends to this kind of an error surveillance mechanism and how widely it should be used in the final product line. This category also contains an automatic surveillance mechanism that could be called as condition monitoring for the application itself. Both error handling and surveillance mechanism belong to the further studies. It might be that these problems become the biggest challenges in the next assessment phase.

Even if this assessing concentrated on the scenarios in various categories, it seems that almost all of them can be handled with some new or existing functional components. It also seems that, all the features are located in some specific components and are not spread all over the components of the system. Therefore, it is not the architecture that should be modified to host the upgrades, but the individual components that take care of actual executions.

The base architecture does not depend on the normal functional requirements, which makes it possible to add various types of features without changing the architecture. This architectural independence of the change of functional requirements makes it a flexible environment for product prototyping. Of course there are some side effects; one of them is unnecessary waste of processor time in the software bus.

The benefits that come from a stable architecture during prototyping can be transferred to product line development. In this case, the prototype architecture can be used as a base architecture for the product line architecture because prototype has been developed for architecture testing, not for functionality testing. That is also why the architecture has been quite well tested against possible requirement changes in the prototype developing time.

Functionalities that have been implemented in the prototype developing phase advances the PLA developing. In the point of preliminary analysis it seems that implemented functionalities are also the commonalities of a great amount of incoming products of the product family. Available functionalities fasten the first product line product

delivery time, because only product specific functionalities have to be implemented.

Even if the configuration control seems to add some implementation and some reparations has to be done on the core of the PLA, the current architecture supports it at an ideological level. The current architecture makes it possible to change the configuration between runs. This limitation can be changed to in advance with some minor framework changes, so that the application framework also supports runtime configuration changes.

The control of the runtime configuration is another aspect of this problem, and it can be separated from the application framework to separate functional component, as previously told.

As a result, the current architecture does not change a lot. Only few additions has to be implemented on the PLA, and some new configuration functionalities has to be implemented to some component that can be used in products that require dynamic configuration controlling e.g. automatic adaptability that could be one part of the further studies.

6. Conclusions

In this paper, we have introduced a case study on analyzing whether or not an architecture designed for a pilot product can be used as a basis for a product line. As a tool for analyzing this adaptability problem, we used a software architecture assessment. The purpose of the assessment was to study whether the architecture of a pilot system can be adopted as a basis for a product family. Adaptability, configurability, and scalability of the architecture, the most important properties to be studied when considering the generalization of a pilot system to a product family, were all included in the assessment.

As a conclusion, the scenario-based evaluation methods (e.g.[5], [6]) work well also for this type of an assessment, even if they may not be originally designed for such purpose. However, the challenge is to be able to find the right scenarios to evaluate. This requires intimate information on future products and software evolution of existing ones, which we did not have in detail. Therefore, some of the issues we chose to assess were derived from potential products via the process of abstraction, addressing e.g. general adaptability requirements. In fact, we believe that such concerns should be addressed in any case when transforming single product architecture into product line architecture. Moreover, the experiences of this paper can also be seen as a part of a check list that should be used when transforming single product architectures to product lines.

For this particular case, the experiences from the assessment have provided both software developers and associated vehicle manufacturers improved understanding on the structure of the pilot system, as well as on future products that would fit this product family.

References

[1] Gamma, E., Helm, R., Johnson, R., and Vlissides J. *Design Patterns*. Addison Wesley, Reading, MA, 1995.

[2] ISO 11519-1 Road Vehicles — Low Speed Serial Data Communications — Part 1 Low Speed Controller Area Network (CAN), 1994-06

[3] Jacobson, L., Griss, M. and Jonsson, P. Software Reuse: Architecture, Process, and Organisation for Business Success, Addison-Wesley-Longman, May 1997.

[4] Jokinen, J., Järvinen, H.-M. and Mikkonen, T. Incremental introduction of behaviors with static software architecture. *In Proceedings of the 2002 International Conference on Software Engineering Research and Practice (SERP'02)* (Eds. H. R. Arabnia and Y. Mum), pp. 10-16, CSREA Press, June 2002.

[5] Kazman, R. Abowd, G., Bass, L. and Clements, P. Scenario-based analysis of software architecture. *IEEE Software* 13(6):47-56, 1996.

[6] Kazman, R., Klein, M., Clements, P. ATAM: Method for Architecture Evaluation. PA: Carnegie Mellon University, Software Engineering Institute, Report CMU/SEI-2000-T-04. 2000.