Characterising Software Platforms from an Architectural Perspective

Ulrik Eklund[1],[2], Carl M. Olsson[2], and Marcus Ljungblad[2]

[1] Volvo Car Group, Sweden, ulrik.eklund@volvocars.com
[2] Malmö University, Sweden

Abstract. With demands of speed in software development it is of interest to build on available software platforms that incorporate the necessary non-competitive functionalities and focus the development effort on adding features to a competitive product. This paper proposes that we move from an API-oriented focus and instead suggest four architectural concerns for describing software platforms as more relevant.

1 Introduction: An Empirical Reflection

Using three empirical cases from the embedded domain, this paper reflects on the challenge for future research for new product and feature development based on existing platforms and the architectural concerns this brings. The need for this is motivated by the current trend in software engineering towards extending and integrating existing platforms to further the supported functionality [1].

1.1 Case 1: Platform Development in a Heterogeneous Domain

Home automation is attracting increasing attention from commercial actors such as energy suppliers, infrastructure providers, construction companies, third party software and hardware vendors. As there are no accepted reference architectures or software platforms we see multiple vertical solutions where companies strive to support the whole chain - from the sensors and devices to gateways and servers, with whatever dedicated software that is of particular interest to the company. This creates a situation where it is difficult to avoid lock-in for third party service developers and customers, which subsequently may limit their willingness to commit and develop services for each specific platform. One example of this problem is in a major project with one of Europe's largest energy suppliers (henceforth referred to as EnergyCorp), where the desire is to overcome the problem of heterogeneous technologies. In this case, three concerns stand out from an architect's and developer's perspective:

- What assumptions can be made about the components supporting the API?
- What limitations regarding the data and communication paths can be inferred from the API? I.e. what guarantees does a developer have when changing state on an actuator?
- What are the run-time dependencies and functionalities not attached to a device initially? This is not obvious since the exposed API is a composition of data from several sources within the platform, and the contextual information about where a device is located is added by the user.

1.2 Case 2: Implementation based on an Existing Platform

Volvo Car Corporation has a strategy to use the AUTOSAR [2] architecture and platform as a basis for the software in new vehicle platforms. To use a platform-based approach is not only deciding on the infrastructure configuration, but also to maintain a selection of functionalities the platform shall support when developing new innovations enabled by software. This caused a need for the architects to:

- Define the scope of the platform that is easy to communicate to managers and developers and roadmapping the platform to meet long term demands.
- Configure the selected subset of basic software in the AUTOSAR platform.
- Focus on platform definition and evolution rather than integration of separately developed applications towards model year changes.
- Separate slow evolving platform development from fast application development in general, and to allow differentiated lead-times for novel features.

1.3 Case 3: Adaptation of a Platform to a Different Context

The Open Infotainment Labs is a previously published case [3][4] of a prototype development of an in-vehicle infotainment systems in cooperation between Volvo Car Corporation and EIS by Semcon. The project had two goals: First, to establish whether it was possible to do feature development with extremely short leadtimes from decision to implementation compared to present automotive industry standard, from a nominal leadtime of 1-3 years to 4-12 weeks. From an architect's perspective there was a need to:

- Analyse the Android platform to identify additional needed services and behaviour when used in a car compared to a mobile phone.
- Break down any change or addition to the platform to fit as a single user story in the product backlog (an effect of the project using Scrum).
- Focus on platform development and deemphasize application development.

2 Research Challenge

The case of EnergyCorp suggests the difficulties to see, on an API level alone, how combined heterogeneous hardware and software from several platforms complement and contradict each other from an architectural perspective. For AUTOSAR, both long and short term product management is concerned with how the functionalities provided by a platform are combined and extended to realise new customer-discernible features. In these situations, it is desirable to scope the platform both in terms of what domain-specific services/functions it provides and what commodity components it contains to build products and product families. Finally, when developing on top of an Android platform it is important to separate the concerns between what services/functions are available and how these services are used. It is impractical for an architect or product manager to incorporate updates in platform backlog user stories in terms of API

changes, instead a higher abstraction of the platform capabilities is necessary. Overall, this hints towards a working hypothesis that the API-level may not be appropriate for discussing architectural concerns of software platforms.

Our challenge for future research therefore becomes how to characterise software platforms from an architect's perspective and what views or dimension are important to capture. Found factors of relevance include building products based on a combination of multiple platforms (case 1), long-term product planning (case 2), or addressing various non-functional properties (case 3). In the four subsections below, we put this working hypothesis to initial scrutiny by reviewing which architectural concerns stand out in our three empirical cases.

2.1 Infrastructure

According to [5] a platform infrastructure consists of the operating system and commodity components, with database management and GUI being two examples. In case 1 and 2 this concern is present, as application developers often have to make implicit assumptions about the available hardware and software services available to be able to efficiently build applications on top of the platform. It is thus important that the infrastructure is communicated properly. Based on our three cases, this should from an infrastructure perspective include at least a list and description of the available hardware devices and commodity components.

2.2 Run-time Dependencies

Simply being aware of the platform infrastructure, however, does not enable application developers to build efficient applications (case 1 and case 2). The type of communication, asynchronous or synchronous, between application and platform components is not easily discernible from the API as functionalities may be physically separated with respect to dependencies on data. An API request to modify a device's state says little about consistency guarantees among the platform components. Highlighting platform communication paths as well as their respective guarantees is thus needed.

2.3 Functionalities

Functionalities are those domain-specific services, attributes, and operations that are part of the platform which can be used for feature and application development. In an embedded system they are usually an abstraction of available hardware sensors and actuators. In a car in case 2, functionalities could be status and control of the interior lights, which can be utilised in development of various convenience features.

2.4 Construction Principle

There are two fundamental approaches to platform construction: monolithic vs. componentised platforms [6]. These affect the construction principle that guides system design and architecture. The monolithic type of platform have a static

structure for every instantiation and variation is achieved by variation points in the components. In the componentised platform, the platform instantiation allows for a creative selection and configuration of components and most of the tailoring towards specific products is achieved through different component configurations.

3 Conclusion

This paper has presented three cases of typical platform usage; development of a new platform for a domain, product family implementation based on an existing platform, and adaptation of an existing platform to a different need. In the discussion of our empirical cases, it becomes clear that none of the main concerns are appropriate to discuss on an API level, which lends further support to the working hypothesis we started from, i.e. that there is a need to establish a suitable level of abstraction. To this end, we identified four suitable architectural concerns to describe software platforms: infrastructure, run-time dependencies, functionalities and construction principle.

Acknowledgements We would like to thank the following supporting organizations: VINNOVA, Energimyndigheten, EnergyComp, ResearchGroup, Volvo Car Group, and Semcon.

References

1. Evans, D.D.S., Hagiu, A., Schmalensee, R.L.: Invisible Engines. MIT Press (2006)
2. AUTOSAR: Technical overview v2.2.2 (aug 2008)
3. Eklund, U., Bosch, J.: Introducing software ecosystems for mass-produced embedded systems. In: Proceedings of the International Conference on Software Business. Lecture Notes in Business Information Processing, Cambridge, MA, USA, Springer (2012) 248–254
4. Eklund, U., Bosch, J.: Using architecture for multiple levels of access to an ecosystem platform. In: Proceedings of the ACM Sigsoft conference on Quality of Software Architectures, Bertinoro, Italy, ACM (2012) 143–148
5. van der Linden, F., Bosch, J., Kamsties, E., Känsälä, K., Obbink, H.: Software product family evaluation. In: Proceedings of the Software Product Line Conference. Volume 3154 of Lecture Notes in Computer Science., Springer (2004) 110–129
6. van Ommering, R.: Roadmapping a product population architecture. In: Proceedings of the International Workshop on Software Product-Family Engineering. Volume 2290 of Lecture Notes in Computer Science., Bilbao, Spain, Springer (2001) 51–63