Fraunhofer
IESE

Openness as an Architectural Quality

Authors:
Taslim Arif
Thorsten Keuler
Jens Knodel
Matthias Naab
Dominik Rost

IESE-Report No. 075.12/E
Version 1.0
November 2013

A publication by Fraunhofer IESE

Fraunhofer IESE is an institute of the Fraunhofer Gesellschaft.
The institute transfers innovative software development techniques, methods and tools into industrial practice, assists companies in building software competencies customized to their needs, and helps them to establish a competitive market position.

Fraunhofer IESE is directed by
Prof. Dr. Dieter Rombach
(Executive Director)
Prof. Dr.-Ing. Peter Liggesmeyer
(Scientific Director)
Fraunhofer-Platz 1
67663 Kaiserslautern
Germany

Abstract

Openness of software systems constitutes a new way of collaboration between software development organizations – on the one hand the open system producer and on the other hand the extension producer (i.e., the consumer of the open system).

Open system producers share information on internal concepts and mechanisms and provide architectural documentation about them in order to enable third party producer to extend the open system. The extensions themselves are typically unknown at development time of the open system. The extension developed third party organizations utilize the mechanisms either at configuration time or at run time (without requiring the open system to be modified or recompiled). Such contributions are either system extensions (e.g., Eclipse plugins or Apps) or adapters to enable interoperation with other systems (e.g., machine2machine communication).

In this report we define the term "open architecture" and present a conceptual model for openness within a software ecosystem (see Section 1). We characterize the organizational roles involved the software ecosystem and discuss the cooperation between open system producers and system extension producers (see Section 3). Furthermore, we elaborate on concepts for integration of open systems and extensions.

Key contributions of this report is a classification schema (see Section 4), which helps organizations in characterizing the situation of their systems and gives guidance to development organizations on whether or not to invest into openness and what consequences to expect.

Examples from the commercial vehicle domain characterize sample situations of development organizations (Section 2) and show how the schema to classify open system can be applied (see Section 4).

The foundations of this report are basis for a follow-up report, which discuss engineering principles to achieve openness in software systems.

Keywords: software architecture, open architecture, classification of open systems, commercial vehicles, openness, software ecosystems

Project: Digitaler Nutzfahrzeugtechnologie (DNT)
Commercial Vehicle Technology (CVT)

Table of Contents

1 Introduction

1.1 Project Context

Digital Engineering for Commercial Vehicles is the main subject of the Fraunhofer "Innovation Cluster" that Fraunhofer ITWM and IESE have initiated in cooperation with regional companies.

The current project activities of the institutes together with industrial partners from the commercial vehicles industry form the basis of the Innovation Cluster "Digitale Nutzfahrzeugtechnologie (DNT)".

The Fraunhofer Innovation Cluster "Digital Commercial Vehicle Technology" is funded by the European Union and the state of Rhineland-Palatinate in the context of the Ziel 2-Programm Rheinland-Pfalz.

This work has been conducted in context of the project DNT2 in work package "7-Integration von Fahrzeugen in die IT-Welt".

1.2 Purpose of the Report

The purpose of this report is to define the term "open architecture" and delineate the term with related terminology. The report starts with sketching openness in general and then focuses on the role of open architecture in particular.

The intended audiences of the report are:

- Members of the *DNT2 project*: This report provides the definition of open architecture and serves to achieve a common understanding of project-related terms.
- *Software architects*: This report introduces a conceptual model for open architecture and provides a classification schema where development organizations can relate to characterize the situation of their own systems
- *Commercial vehicle software engineers*: This report provides examples from the commercial vehicle domain to illustrate open architectures and its underlying concepts.

1.3 Openness in Software Engineering

Openness is a term first coined in the field of thermodynamics, where it describes a system that continuously interacts with its environment. With the open source movement the term was adopted in software engineering, and

many instantly still think about open source when they hear the term openness or open system. However, openness and open system have a great variety of different meanings and also for open source there are more aspects than just the possibility to browse the source code of a system (cf.[1]). In the following some aspects to consider for openness are explained.

Availability vs. Transparency vs. Participation: Matt Zimmerman outlines the different aspects of openness in open source as availability, transparency and participation [2]. He understands availability as the possibility for everyone to access the source code of a system, transparency as the possibility to witness the development of the project in terms of discussion forums, source code history, bug reports, etc. and participation as the possibility to get involved in the development of a system. Although openness in this sense targets open source exclusively, the different aspects are also relevant for open architectures.

Static vs. Contracted vs. Public: An aspect that is more directly related to the notion of openness that underlies this report is the possibility to extend the product with additional components. For this, the following classification can be given. A *static* system does not support this possibility and cannot be changed after it has been shipped to the customer. As such, it can be considered as simply closed. Other systems may offer the possibility to extend a product with custom components, but limit this possibility to known organizations. Therefore we refer to this variant as *contracted*. The variant with the highest degree of participation is public. Here, an organization creates an open product and allows any other organization or person to produce additional components for it. It is also possible to bind this to certain regulations but in general the producers of extensions are unknown at the time of the finalization and shipping of the product.

Free vs. Charged: Openness, especially with regard to open source is often associated with being available without charge. Despite that this might be the case for some open systems this cannot be considered as the standard. Hence, open systems can be classified according to their price model. We can distinguish open systems that are freely available and such, for which license fees have to be paid. For charged systems we can distinguish systems that have to be paid for use, i.e. once for a certain period of time, or pay per use, where a certain amount is due for every time the system is used.

The meaning of openness is highly dependent on the context. The following list provides an overview of different common perceptions of openness.

- *Open Content*: The term open content was created by the Open-Content-Initiative and describes content of any form that is freely available and may be distributed, changed and used without conventional restrictions. The in-

itiative created corresponding licenses under which such content can be published. As such, open content is directly related to open source.
- *Open Source*: Open source primarily refers to software, which source code is freely available. Other related aspects have been explained above.
- *Open Data*: Open data refers in general to the availability and right to process data of any kind. This is specifically related to the absence of conventional copyright regulations to promote the creation of innovative ideas and content.
- *Open Standards*: For standards it is a prerequisite to be open (i.e. available to the public) to a certain degree to achieve relevance. Therefore the term might be considered as redundant. Standards not being open would not be used and thus lose their validity. Open in this case is independent of potential royalty-fees that users might have to pay to the creator of the standard, which is a common practice. However, in the combination of "open" and "standard", "open" is often more specifically defined by additional rights, like royalty-free usage, for example.
- *Open Protocols*: When an open standard defines a (communication) protocol, it can be referred to as open protocol. The same aspects apply then analogously.
- *Open API*: Open API is a term that describes the possibility of systems to interoperate with webpages or services in the internet. There for interfaces are created and made available that can be used by application developers to access data or functionality of the provider.

In this report, we focus on systems with an architecture that supports the incorporation of extensions. We refer to such architectures as *open architectures*. Consequently, we define systems that possess such an architecture as *open systems*. We will use the term in this sense throughout the report. In the following section, open architectures are defined and explained in detail.

1.4 Open Architectures

We define open architectures as:

Openness is the property of a system architecture to enable added-value services by incorporating third party contributions (typically unknown in advance) while retaining essential qualities. Explicit mechanisms for openness are established at development time and third party contributions utilize such mechanisms at runtime. The decision to open a system architecture is always based on business rationale.

Thereby it is important to note that:

Contributions are either system extensions (examples like Eclipse plug-ins or Apps) or adapters to enable interoperation with other systems (examples like machine2machine communication).

In this definition, some aspects need to be emphasized:

Open architectures allow the incorporation of third party contributions. We refer to such contributions as *system extensions* or simply *extensions*. They contribute additional or enhanced system elements like data, functions or user interfaces, to generate a benefit for the user of the system.

Such extensions are typically created by third party organizations. This is a central part of the business model that underlies the idea of open architectures. It includes the idea that a business ecosystem evolves, in which multiple independent organizations contribute to the same product. Such an ecosystem shall create a situation in which the system producer, third party contributor, and the user all benefit from the same product. There is no pure technical motivation to choose this type of architecture. In the typical case open this architecture style implies the need for an increased effort investment for creation or maintenance of such a system. Only with the possibility of an increased income on the basis of the corresponding business model, such a decision becomes plausible. This is reason why the definition states that the decision to open a system architecture is always based on business rationale.

In consequence the contributors and the concrete extensions are typically unknown during the creation of the open system. Open system producers will normally first create an open system and then third party organizations will create extensions for it. This means that mechanisms for openness, that allow the adding of contributions, are established in the system during development time. These mechanisms are then utilized by extensions, after the system has been delivered to the customer, i.e. during build- or runtime.

Openness as an architectural quality has trade-offs with other quality attributes and thus constitutes a challenge in their achievement. It is therefore the responsibility of the architect to make such relations in a concrete project setting explicit and to ensure the retaining of other essential quality attributes.

Although openness contains some similarities with interoperability, there are foundational differences. Interoperability targets two independent and fully operational systems that are integrated to generate a benefit. Openness in contrast targets one focus system, the open system that is capable of incorporating extensions that have been created specifically for that purpose and are not operational without the open system. However, openness can be utilized for interoperability, with the help of the extensions that can establish the interoperation with another system.

2 Example Scenarios Characterizing Openness

Open systems can be of different types (e.g. Vehicles, Desktop Applications, Cloud Platforms etc.). In order to characterize open systems, three representative scenarios are presented below.

In the first scenario a company producing agriculture machinery offers its machine to be extended by their customers or external software companies. A software company extends the machine with one add-on software component. As a result customers of the machines get more value added machines than before. The machine producing company, the external software company and the customers – all get benefits from this newly created ecosystem by complementing one another.

In the second scenario the Eclipse platform (a desktop application platform) is described, which is open to their users, who can arbitrarily extend it. A software company makes one business process modeler as extension to the Eclipse platform. The customer company of this extended product further extends it to fulfill their business goals.

In the third scenario SalesForce.com (cloud platform for collaboration with customers) is open to be extended. A customer organization of Force.com identifies the need to extend it to integrate their Microsoft Outlook application to centralize all information in a single location. They outsource the task to another software company. The software company builds the Outlook Integration for Force.com. Now the customer company of Salesforce has their email system integrated to the force.com. It makes the customer company more productive in collaborating with their customers as they eliminated the need to constantly switch the tools they work with.

2.1 Agriculture Scenario

A well-known company in the agriculture domain manufactures machines to cultivate and harvest the land. It provides products and services to transform and enrich land. Their portfolio comprises a variety of tractors, machines for planting and seeding, grain harvesting and so on. Their Customers are usually the farmers, who buy this equipment for usage on their own land. Another common setting is, that certain organization buy different agricultural equipment and offer it on a pay-per-usage basis to farmers with smaller fields, for whom buying all the equipment would be too expensive. Farmers often employ field workers to work for them in the field and to operate those machines.

The manufacturing company plans to introduce open architectures in their new generation of agricultural machines. They plan to create a new market around their products, create additional income and new business opportunities. To make this possible, they open their machines for extensions with additional functionalities, data and user interfaces.

They expect that the open architectures will create possibilities to make their machines better integrated with their environment, like with other machines, information systems and business processes of their customers. They hope that additional functionalities will be created that make their products even more attractive for customers. As they open the machine data for further processing, it could additional advanced data visualization techniques could be developed. For example, usage statistics or usage history can be visualized.

They expect the machines can be better integrated with the surrounding machines, information systems and processes if they allow additional software components to be installed into their machine platforms. It would open opportunities for their machines to be extended with new functionalities. Some of them can be visualizing the data produced by the machine. For example usage statistics or usage history can be visualized the way customers' want. Data can be sent to other information system to be processed further. Data includes current location, working history of the day, current status etc. Even a task schedule can be uploaded into the machine to be carried out by machine without any more user input. But this of course means they have given access of the control of the machine available to the extension components too.

The usage and status data of these machines include the location of the machine, current operating status, usage history and so on. It can be used as valuable data for tracking, reporting, diagnosis, and even for financial calculation. The company understands that this data is a valuable information and it can enrich the value of the machine in many ways. The company offers the data to be processed by the customers' systems in various contexts. For example one farmer might want to know whether his contractor or worker is currently working in the field or how much the worker worked in the whole day etc. With this kind of additional functionalities, the company wants to create additional values of their machines and get a competitive advantage over their competitors.

To make the machine data available for further processing, the company plans to build an application platform in their machine with access to machine operation data in read only manner and where additional software components can be deployed. The machine data they have given access, can be used by the farmers to better control their field remotely and also for different reporting or financial calculations.

2.1.1 A software Company Makes the *Cloud-Loader* Add-on for the Machine Platform

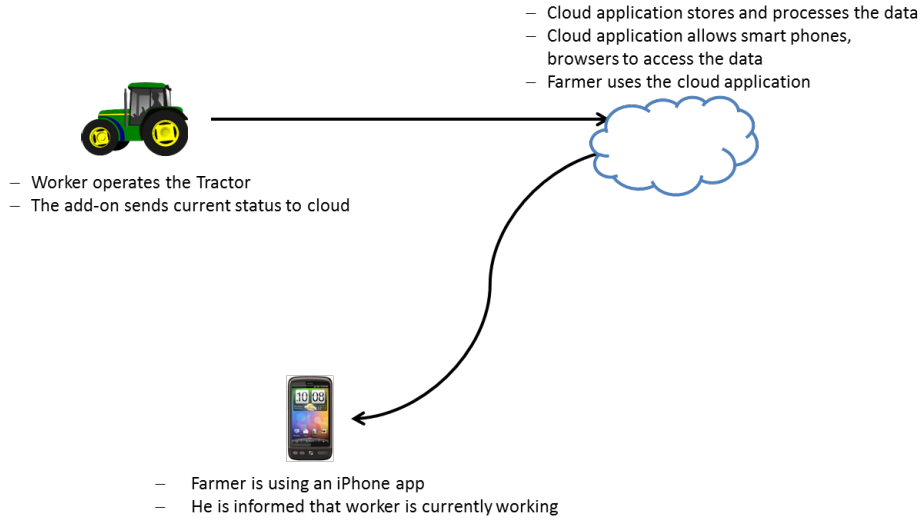

Figure 1 : Machine Data Accessible through Cloud Platform or Mobile Devices

A software company builds software in the agriculture domain. As the tractor manufacturer made the platform of their machines available for externals, the company found potentials to build add-ons to upload the machine data to a cloud application. This cloud application can be used by the contractors/ farmers to monitor their machines in real-time.

The software company developed the *Cloud-Loader* add-on for the machine platform. The *Cloud-Loader* add-on periodically sends the location and status of the machine to the cloud. At the end of daily work the operator can also upload the status report (what is done, what is still left, problems faced and so on) to the cloud. The cloud application stores all those status reports, processes and makes it available as web services for the contractors/ farmers to access through iPhone app or some browser.

The software company's plan is to sell the *Cloud-Loader* add-on and the cloud services to the farmers and contractors. They can also provide consultancy services to install the add-on in the machines, to set up the cloud environment for the farmer.

The tractor manufacturer benefits from this external add-on as they do not need to put effort to build such a helpful feature. The tractor manufacturer and the software company both are complementing each other and achieve mutual benefit.

2.1.2 Farmer Gets Updated Instantly

Consider a farmer who owns several tractors of the above mentioned manufacturer. He decided buy one of the new models, because he wanted to try out the possibilities that an open architecture has to offer. He expects that he can easily upgrade his machine with new features that better support his business process and make field work more efficient. He has some employees working for him and now wants to track their farming activities, to be able to make quick decisions if they run out of schedule.

He checked the store of the machine manufacturer where all the different extensions are listed for sale. There he finds the above mentioned software company who offer an add-on for the tractors that sends machine operation data to the cloud which is accessible in real-time from his home via a cloud service. As it is not very expensive he buys the software company's add-on and cloud services to access the progress and field data.

After installing the *Cloud-Loader* add-on, he instructs his workers to upload the daily task status at the end of the day through it. The Cloud-Loader itself sends the location and the current operation status to the cloud periodically. The cloud application makes the location and the current operation status available as web service directly without any processing. But from this information it processes the usage statistics and makes that available too as a web service.

The farmer uses his mobile device to check whether his workers have already started to work that day and what the current progress is. He also performs financial calculations at the end of the month based on the usage history of the tractors, working hours of workers and so on provided by the cloud application.

2.2 Eclipse Desktop Application Scenario

Eclipse is a desktop application platform that uses plugins to provide functionality within and on top of the runtime system. This architecture is different compared to other desktop software where functionalities are rather built-in the system. The plugin architecture of Eclipse allows extending Eclipse in any desired way to the environment. Eclipse provides the Rich Client Platform for developing general purpose applications.

2.2.1 A Software Company Develops Eclipse Plugin

Consider a company that develops software to enhance the collaboration between business analysts, SOA architects and IT. They develop the application Business Process Modeler for business analysts based on the Eclipse platform.

The software supports BPMN modeling, sharing among all the team members, processes reviewing, BPEL visualizing and document generation.

The software company is using Eclipse because it provides a proven robust platform to build desktop applications. The platform already provides a runtime, modeling framework, visualization and user interaction support. So it reduces the development effort greatly. And as the software company is using the Eclipse platform, their software can easily also be extended with other software components based on the customers' needs. The software company uses the modeling framework and visualization functionalities of eclipse and builds the business process modeler as an Eclipse plugin.

2.2.2 Customer Uses Eclipse Plugin and Extends It

Consider a company that buys the Eclipse based BPM software of the above mentioned software company. They chose this application to support their business project management tasks because they wanted a cheap and highly flexible solution, which was easily adaptable to any changes.

For effort estimation of their business modeling tasks, this customer company was using expert judgement before. They are now planning to track and save all their actual effort data and then analyze this data for future effort estimation. They want the process to be automatic with minimum interaction with the workforce they have. They asked their in-house IT department to investigate the possibilities. The IT department found that they can extend the software with additional plugins to automatic tracking of effort, storing and analyzing for future estimation tasks. Now the customer company has an integrated system that supports all their business needs regarding business modeling and it makes the company satisfied because of their choice of Eclipse based software.

2.3 Cloud Service Scenario

The cloud provides different levels of services, namely the software as a service (SaaS), platform as a service (PaaS) and infrastructure as a service (IaaS). In the PaaS model, additional software components can be deployed by the PaaS customer. The PaaS can be for generic purpose use (for example Google App Engine) or it can be domain specific (e.g. Force.com). In both cases customers have the opportunity to deploy their software components and extend the provided functionalities of the platform.

2.3.1 Salesforce.com Provides Extendable Force.com Platform

Salesforce provides cloud apps and platform for the social enterprise. It helps to connect to customers and employees. No hardware and software need to be

installed to use the services. If somebody is just connected to the internet, she can use the services.

The salesforce.com provides a broad range of services. The sales cloud is a SaaS service that helps the sales representative the opportunity to collaborate with company experts, complete customer profile, managing marketing campaign etc. This SaaS service allows the companies to interact with their customers by not only call center but with other social networking channels. Users can also join in the conversation, can do video calls (through FaceTime etc.) and can get prompt response.

Along with these services Salesforce also provides a PaaS platform known as Force.com. It allows the users to create additional applications that integrate into the main salesforce.com application and are hosted on the salesforce.com infrastructure. Salesforce provides a marketplace named AppExchange for selling and buying apps. With Force.com Salesforce created a highly innovative platform that attracts many additional customers to their main product. Offering the possibility to create extensions enhances their product with high flexibility, constantly added features and generates additional income through usage fees and consultancy services. Salesforce.com allows additional or modifications of user interfaces, user defined fields can be added or new tabs can be added for example.

2.3.2 A Customer Company Identifies the Need to Extend Force.com

Consider one of the customers of the Salesforce.com. They are using Salesforce to contact and collaborate with their customers. They use Outlook and Exchange as their internal email system. Often, information is exchanged internally but also with customers, which is then not available within the Salesforce platform. However, the customer company expects that having all relevant information centralized in a single location would greatly improve the efficiency in their business processes. The company contacts a software company to build an extension app for integrating outlook with Force.com.

2.3.3 A Software Company Builds the App for the SalesForce.com Customer

Consider a software producing organization specialized in building apps for Force.com. As a start-up they decided to specialize on Force.com as it already has an established customer base to which they can easily offer their product. Technically, Force.com provides an existing infrastructure which they could simply reuse, which greatly decreased their initial development effort.

They build one app for the above mentioned company to integrate Outllook to Force.com. They extend the collaboration history user interface of Force.com.

Now sales persons can see all the collaboration data from Outlook and Salesforce in one place. It makes them more productive during collaboration.

3 Conceptual Model of Open Architectures

In the following sections, a conceptual model for open architectures is introduced. It serves as a foundation for the engineering approach, which is described in section 4. The conceptual model includes foundations of openness as an architectural quality, business motivations of the main roles, the organizational settings around open systems, a characterization of open systems and extensions, as well as their integration and a delineation of openness from other quality attributes.

3.1 Foundations of Openness

In section 1.4 the definition of open architectures has been introduced and some basic implications been explained. In the following, the main aspects that constitute the conceptual model of openness are explained in a short overview. The subsequent sections provide more detail on each of these aspects. Figure 2 provides an overview.

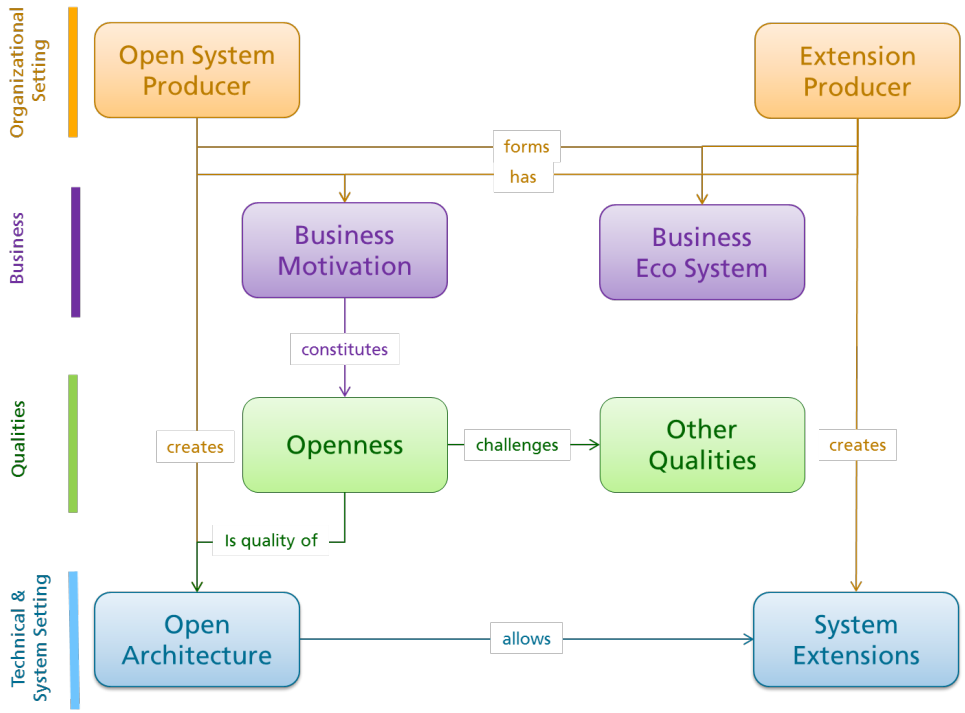

Figure 2. Foundational Aspects of Openness.

Figure 2 summarizes the definition of open architectures and shows the main aspects covered in the conceptual model, which is explained in the following sections. The aspects include the *organizational settings*, covered in section 3.2, the *business motivations*, covered in section 3.3*, openness and other qualities*, covered in section 3.6, as well as the *technical and system setting*, covered in section 3.4 and 3.5.

As described in section 1.3 apart from other meanings of openness, in this report we cover openness as a quality attribute of architectures, i.e. open architectures. The main idea of open architectures is systems that have an open architecture (i.e. open systems) allow the incorporation of system extensions. These system extensions offer value added services to the open system. This means, that the system is able to provide additional benefits to the user by integration of extensions. This can include enhanced hardware, additional data, additional features, an enhanced user interface or additional process steps. Open systems can be of various different types like vehicles, mobile devices or distributed systems.

System extensions are always created for a concrete open system. Unlike interoperability for example, where independent and fully functional systems are integrated, system extensions depend on the open system, which operates them and makes them usable. Therefore, the integration effort of open systems and system extensions is in the majority of cases very low. System extensions are typically contributions of third party organizations and thus still unknown at the point in time when the open system is created.

Openness as a quality can have possible conflicts or tradeoffs with other quality attributes a system has to achieve. It is crucial for architects to ensure that other qualities are retained when introducing an open architecture. Introducing an open architecture is achieved by creating openness mechanisms during development time of the system. These mechanisms are then utilized by system extensions after the system has been delivered to the customer.

Concerning the organizational settings, three main roles can be identified: the open system producer, the extensions producer and the customer. In a typical setting these roles are taken by different organizations. Each of these organizations has a business motivation concerning openness and open systems. An open system producer makes the decision to create this kind of system only based on business considerations as from a technical view, it only creates challenges. Business considerations can include obtaining an enhanced product with increased flexibility and customizability, an increased visibility and new opportunities for additional income. Extension producers can benefit in several ways from open systems. Apart from the technical infrastructure they can use that significantly reduces the effort to produce a fully functional and sellable feature, they can benefit from the image of the open system producer and

have a customer base at hand instantly. Customers may choose open systems over conventional ones because of the high number of available additional features and the increased customizability that may help to adapt the system to changing requirements.

The following sections will explain each of these points in greater detail to provide a conceptual model for the classification and characterization of open systems and architectures.

3.2 Business Motivations for Open Systems

The decision to create an open system is always based on business considerations. This means, the business benefits must outweigh the increased effort investments for creating and maintaining an open system. The following sections list and explain considerable factors on which the decision to create an open system may be based. But also possible motivations for the creation of system extensions and factors that may lead a customer to choose an open system over a conventional one are given.

3.2.1 Business Motivation for Creation of Open Systems

An organization producing software intensive systems may either migrate an existing system to an open one or create an open system from scratch. In any case a significant investment is necessary for the development and maintenance of such a system and is even potentially higher than for conventional systems. Thus, the economic benefits must outweigh the additional effort investment. The following list describes beneficial factors that might be reasons for the decision to create an open system.

We separate three groups of business motivations for the creation of an open system: *an enhanced product, additional income* and *strategic goals*.

Enhanced Product

The following benefits may facilitate an enhanced product by utilizing openness properties:

- *New Features:* Opening a product may lead to a substantially increased feature set in a short period of time. While the system producing organization only provides the platform and an initial feature set, additional feature development is outsourced to third-party developers. As numerous different developers or development organizations are involved, feature development is highly parallelized.
- *Customizability:* Additional features can be installed by customers or users at will. As for an open system typically a high number of additional features are available, this leads to a highly customizable product that can possibly be

adapted to customer needs. Customizability is a strong selling point, often making customers choose such a product over a conventional system. The iPhone or Android phones are examples where the high number of available apps and thus high customizability makes them very successful products.
- *Product Integration:* Opening a system may lead to an improved integration with other products. Third party organizations might utilize the openness mechanisms to make their product interoperable. For example, in a setting were John Deere provides tractors as open systems, an implement manufacturer could decide to create extensions for the tractors to make their implements interoperate with the tractor. Creating interoperable products is beneficial for the producer of the open system as well as for the producer of the extension as it attracts customers to choose such products.

Additional Income

Additional income may be generated because of the openness property of a system. The following list provides an overview:

- *License Costs:* Openness allows additional income through license costs. Thereby, different licensing models are imaginable. One example is keeping a certain percentage of the selling price of system extension as license fees. Also registration costs for the permission to publish extension in an app store generate additional income. Apple applied similar concepts for their products.
- *Consulting Services:* Products that allow extensions offer the possibility to create a consulting business around it, which can generate additional income. For example, JBoss is a company that generates most of their income with consulting services around their open source application server product. Such business models are imaginable for open systems also. Consulting services for open systems can cover usage, operation or development of extensions for the open system. The more system aspects are opened for extension, the higher is the need and potential for consulting services around the product.

Strategic Goals

Openness might be an enabler to achieve strategic goals that contribute to the success of the product. The following aspects can be named:

- *Increased Visibility:* Creating an open system can potentially increase the visibility of the product on the market. With a large number of developers working with and communicating about the product can lead to an increased popularity. An example for this is the Android platform.
- *Lock-in Effect:* The features and integration potential of open systems create a lock-in effect that makes customers less willing to change from their existing product to a product of a different manufacturer. The effect is intensified, the longer the system is used and the more data is produced that

would have to be migrated. This is true in particular for open systems with specific system extensions that might not be available for another product. Preventing customers from changing to another manufacturer can be even expedited by ensuring downward compatibility of all products so that created data and existing extensions work also with new releases.

3.2.2 Business Motivation for Creation of System Extensions

Openness does not only offer additional business opportunities and benefits for the producers of open systems but also for the producer of system extensions. They can benefit from the existing infrastructure, the image and popularity of the system producer and its products as well as from an existing customer base. This can be beneficial in particular for start-up organization that lack large development resources and an established customer base. In the following, some aspects are described that might be considered as factors for organizations to develop extensions for open systems.

We group the potential benefits in *Technical Benefits* and *Strategic Goals*.

Technical Benefits

- *Infrastructure Usage:* Big parts of the infrastructure functionality are already present in open systems and must therefore not be implemented by an extension producer, but can simply be used. This reduces significantly the development effort for creating fully functional features and is in particular beneficial for organizations with limited development resources.

Strategic Goals

- *Image Benefit:* Extension producers can benefit from the image and popularity of the open system producer and its products. It is possible to gain a certain visibility without the need for marketing efforts.
- *Customer Base:* Related to the previous factor, extension producers can reach a much larger customer base, especially if a sales channel like an app store is available. This largely eliminates the need for marketing efforts which producers of conventional systems have to spend to raise the awareness for their product.
- *Low Prices:* Reusing the infrastructure functionality and reducing the marketing effort allow organizations to keep the prices for their products low. This can help to create an initial success as customers are typically willing to invest a small amount of money to try out a new functionality. This might help start-up organizations to achieve an early growth and further investments.

3.2.3 Motivation for Purchasing or Usage of Open Systems

Open systems provide also for customers and users benefits that often found the decision to choose an open system over a conventional alternative. We see

a clear trend towards this kind of system in specific domains and in many other domains the possibilities of transferring the concept are currently explored. Advantages of open systems for customers and users can be seen in the areas of the *product* itself but also in the *integration* with other systems.

Product

- *Features:* Open systems offer a large variety of features with which customers can extend the capabilities of their product. Additionally, new features are added after the system has been built and delivered. This means that customers constantly receive new features to extend and customize their system after the initial purchase.
- *Customization:* The rich feature set and constant evolution allow a high degree of customizability. Systems can be constantly adapted to new or changing requirements. And additionally, new feature might possibly allow more efficient and effective fulfillment of tasks.

Interoperation

- *Reuse of Existing Systems:* Open systems can allow smooth integration with other systems, devices and products. For customers it is beneficial to choose a product that allows interoperation with already present systems. A farmer will more likely choose a new tractor that allows controlling of his available implements, for example. Open systems facilitate such reuse as by utilizing the openness mechanisms, achieving interoperation with other systems can be simplified.
- *Availability of Interoperable Systems:* Opening a system for extensions also facilitates the creation of interoperable systems. This means that open systems will motivate other manufacturers to build systems that are capable of integration. The availability of interoperable systems can increase the possible alternatives for customers who plan to purchase additional systems in the future and thus the probability to find the most adequate solution.

3.3 Organizational Settings

Throughout the lifecycle of open systems, several organizations take different roles in the development, execution and operation of the open system. An overview of the different roles is given in Figure 3. We distinguish the roles of *User, Customer, Integrator, Open System Producer and Extension Producer.* It is important to note that the allocation of roles (who takes which role in a concrete setting) with concrete organizations is highly dependent on the system type. The user and customer might be the same person for a mobile device but in the case of commercial vehicles they are most likely different. The following sections describe each role with their tasks and responsibilities in detail.

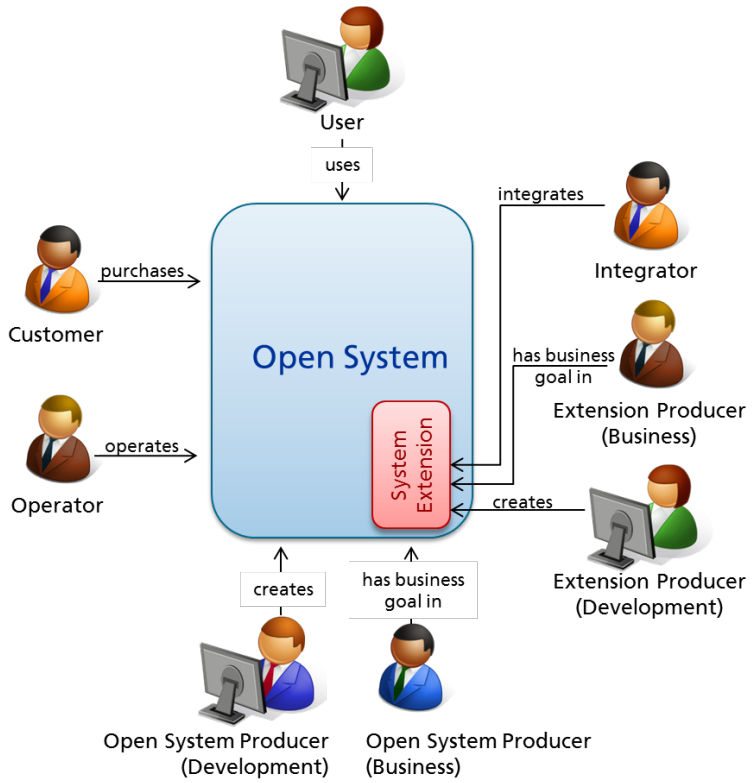

Figure 3. Organizational Setting Overview.

3.3.1 Open System Producer

Open System Producer is the role that is responsible for the development of the open system. The company that can fulfill this role is dependent on the type of open system to be produced. It can be a vehicle manufacturing company like John Deere, a car company like BMW or a plain software companies. Software companies usually cover a large variety of possible open systems. For example companies like Google (Android), Apple (iOS) or Microsoft (Windows Phone) produce mobile platforms. IBM developed the Eclipse platform as a desktop application. The Cloud service provider SalesForce developed Force.com as a platform in the cloud. So, in summary, any producer of software intensive systems can be an *Open System Producer*.

Two main roles inside the open system producer company perform the development of the open system. They are the *Open System Developer* and the *Open System Business Stakeholder*.

The *Open System Developer* is a a technical role, responsible for the development of the open system. To address the open system characteristics they need

to focus on some additional things. Their main responsibility is to create a high quality system, on which additional system components can be deployed. Acting as an architect, they need to provide the architectural guidelines for probable extensions as well. They need to perform transition planning and change management carefully as the system usually needs to be backward compatible to support all the existing extensions.

The *Open System Business Stakeholder* represents a class of stakeholders that fulfill business responsibilities around the open system, like sales, finding strategic cooperations or creating short, mid, and long-term business strategies. Additionally to conventional settings they need to focus on identifying ways of trading with open architecture and also explore new alliances and outsourcing opportunities in the context of the new ecosystem. The Business Stakeholders also need to make the business process (realized in the open system) clear to all future extension providers. Ill-defined and ad hoc business processes are not suitable for extension. They need to bridge the semantic differences in the interpretation of data among multiple business entities.

3.3.2 Extension Producer

Development of system extensions is in most cases done by other organizations than the one developing the open system. However, in particular cases the roles can be centralized to a single company and fulfilled by different internal groups. For example Google develops Android as well as the Gmail-App running on it. But in most cases it will be a software producing company focusing on producing system extensions for a specific or multiple open systems of the same type. For example Rovio Entertainment produces Angry Birds for all mobile platforms.

An Extension Producer can also be the customer company using the open system. For example in case of Force.com, the customer companies can extend the functionalities by themselves. In some cases, the task can also be outsourced to an external software company. It can be even an expert user who has good level of system knowledge. For example expert android mobile users can easily make custom applications to support their daily needs.

Extension producing can be distinguished in two main roles in the organizational setting. These are the *System Extension Developer* and *System Extension Business Stakeholder*. In general the developer and business stakeholder can be from the company that is developing the extension but alternatives may exist as well. For example if the customer company wants the open system to be extended, the business stakeholders of the customer company can make all the business related reasoning and can outsource the technical development task to some third party software companies. For example a company using the

Force.com platform can find out a new business plan and then outsource the realization of the extension to a software company.

The *System Extension Developer* role technically creates the extension for the open system. They need to master the development environment (if any) provided by the *Open System Producer* and follow the reference architecture for the extension. They need to be updated with the development of the open system. In the case some data format or API is changed or deprecated, they need to address those in their extensions for better compatibility in the future versions of the open system. To have a really extendable architecture they also need to provide possible extension points of the extension so that later it can be extended for some other business goals.

The *System Extension Business Stakeholder* finds out ideas of possible extensions of the open system. They need to understand customers' changing needs and explore new business opportunities in the context of the open system. They explore the possible extension opportunities provided by the open system and assess the impact of probable new extensions. They create processes that span across the default open system.

3.3.3 Operator

The *Operator* role is responsible for the day to day operation of the open system. Which organization is fulfilling that role is strongly dependent on the type of open system. In the case of vehicles, they are operated by the company that owns them, respectively the employees, like drivers, farm workers, etc. Desktop applications are operated on workstations, by the company using it or the end user. Mobile devices are usually operated by the users. Cloud platform services are operated by the company developing it or subcontractors, like internet service providers, hosting the application. So in summary the operator of an open system might either be the customer (organization) or a hired subcontractor.

3.3.4 Customer and User

The *Customer* is the organization or person purchasing the open system. A *User* can be an employee of that company or the Customer herself (in the case of a person). An example would be a farming company (customer) purchasing a John Deere tractor and a field worker (user) driving it. In case of the iPhone the customer and user is usually the same person.

Customers or users can identify their additional requirements and find out extensions to support their business goals. They can purchase those extensions, can produce extensions by themselves or even can outsource the development of the extensions to some third party software companies.

3.3.5 Integrator

The *Integrator* usually installs the extension and customizes the system if necessary. The integration role can be taken by the user, the customer company, the producer company or even by third party companies based on the type and complexity of the open system.

The integration of an extension into an open system is generally done by the user or operator without any dedicated integrator. The open system is intended to be integrated with the extensions, so by creation the open system has integration capabilities. For example, the user can install any app in the iPhone by herself.

But in case of complex open systems it might not be possible for the user or operator to perform integration. In this case integration might be offered as additional (paid) service. This may be offered by a third party.

If extension is produced by customer or some outsourced software company, the integration is also performed by them. But if a large number of devices need to be extended, integration can be taken as a paid service from third party. Moreover if customization or configuration is also required during integration, then dedicated integrator might be required.

3.4 Open Systems and Extensions

Open systems can come in very different forms and types. The form however has a substantial influence on the engineering and evaluation of open systems and their extensions. So, it is essential to be able to characterize both as comprehensively as possible. The following sections discuss the most important aspects of open systems and extensions, including e.g. *system types, system borders* or *opened system levels*.

3.4.1 Open Systems

To be able to characterize open systems, in the following sections the types of systems, system borders, the opened system levels, possible access rights, the initial capabilities of an open system and the provided support to build extensions are described.

3.4.1.1 System Types

The type of the open system has a major influence on its organizational setting and engineering. We distinguish the following types of open systems. An overview is given in Figure 4.

- *Vehicles and Machines:* Any kind of commercial or private vehicles like cars, trucks or tractors, as well as production machines.
- *Desktop Applications and Platforms:* Applications that are executed on a workstation, in contrast to applications being executed on a server or in the cloud. The Eclipse platform is an example for this.
- *Mobile Devices and Platforms:* Smartphones or tablets, together with corresponding mobile platforms like Android or iOS.
- *Distributed and Cloud Services:* Applications that are provided in a distributed manner or as cloud services and support extensions.
- *Computing Hardware:* Computing hardware that is opened for further extensions.

Figure 4. Open System Types.

3.4.1.2 System Boundaries

Any of the aforementioned system types can come in the form of an open system and in an isolated consideration the system boundaries are quite obvious. However, open system producers might also provide linked or compound systems as open systems to offer more features and extension possibilities to extension producers. An example for such a case would be a manufacturer of agriculture vehicles that produces tractors that support apps and additionally offers a cloud service to which the tractor uploads data and that can be extended with custom software. So, the question arises, where the boundaries of the open system are and what elements of such an integrated system can be regarded as the open system.

We can regard *every system and system part as element of the open system that either can be extended with third-party system extensions or that is provided to be used by system extensions to perform their functionalities*. This means, in the example given before, the vehicle as well as the cloud platform would be regarded as an integrated open system.

In consequence, system extensions can consist of multiple parts as well. We consider extensions as extension parts of the same extension if they are deployed in the same open system and conceptually contribute to the same functionality.

Figure 5 illustrates several different alternatives of open systems consisting of linked individual systems. The blue lines illustrate the connections between the

single systems. At the bottom, multiple system extensions are illustrated that are deployed onto the system parts. Accordingly, there are individual, as well as compound system extensions given. So, in summary it is important to note that both, open systems and system extensions can be individual or compound and different compositions of system parts can be regarded as the open system, depending on the openness mechanisms provided for extension.

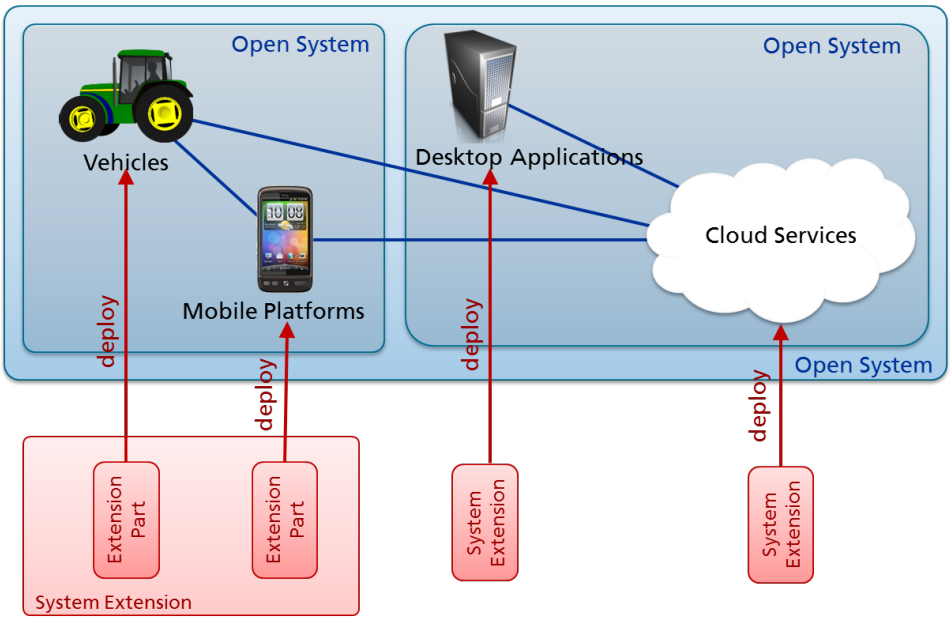

Figure 5. System Boundaries.

3.4.1.3 Opened System Levels

A central characteristic of open systems is the system levels that are opened for additions in the form of system extensions, or, from the view of the extensions, what types of system elements can be added to the open system by one extension. We distinguish the levels of *hardware, data, functionality, user interface* and *process*. It is important to note that an open system might open more system levels than are extended by a single extension. For example, an open system might open its user interface, functionalities and data for extension, a specific extension however only adds a new user interface. Accordingly it is also possible that a single extension provides enhancements for more than a single system level. An extension could for example provide additions for the user interface and functionality. In Figure 6 an exemplary situation is illustrated in which three system extensions provide additions on different system levels. In the following the levels are explained in more detail.

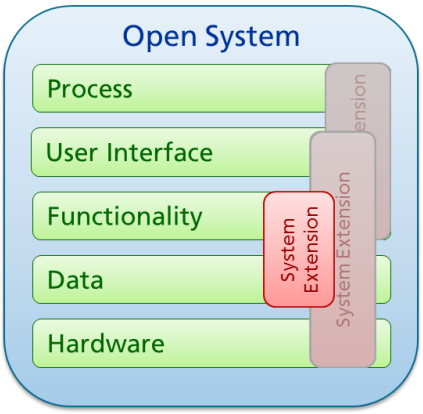

Figure 6. Opened System Levels.

Hardware

A very fundamental way of opening a system is to allow adding or replacing hardware of that system. It is imaginable that owner of such an open system replace hardware components with others that offer more resources or that new components are added that provide additional functionalities like sensors, for example. Standard personal computers are examples for this category as they allow adding and replacement of hardware components.

Data

Opening the data level may allow reading of existing system data or generation of new data. With data reading, extensions can access the data of the open system and process it. Concerning data generation it is imaginable that extensions provide additional sensors that allow measuring new kinds of data, or that extensions provide calculation, analytics or aggregation mechanisms that process existing data. The generated data would then be provided as additional data to the open system.

Functionality

Extensions may provide new functionalities to the system or alter existing ones. A great variety of additional functionalities is imaginable and can reach from basic features like a notepad app to advanced ones that provide complex calculations.

User Interface

Extensions may provide additional or alter existing user interfaces to allow better control, quicker capturing and understanding of data or overall an enhanced user experience. According techniques could include data aggregation, highlighting, notifications, etc.

Process

For systems that work with executable workflows it is possible to add or alter existing process steps.

3.4.1.4 Access Rights

Open systems may grant different access rights to extensions. In general extensions may read, write or modify elements of the system level to which they contribute. For example, it can be distinguished whether an extension shall be allowed to read the data of the open system or write additional or manipulate existing data.

However, it is also possible to manage access rights not only for the system levels, but more fine grained for specific data, functions or hardware parts. So, it would be imaginable that one type of data is accessible while the access to another type is restricted. Providing such information as documentation to extension developers is crucial for the success of the open system.

3.4.1.5 Initial State Capabilities

Another characteristic property of open systems is the capabilities that the system has in its initial form. This determines the degree to which the system is operational when it is delivered to the customer and consequently also the importance of the openness-attribute for a particular system. There exist open systems that are as such only *platforms* and offer almost no functionality in their initial state. The Eclipse RCP platform is a possible example. In such cases, system extensions have major importance for the product. On the other hand there are systems that are delivered to the customer in a fully functional form. For such systems, openness is more an addition that is nice to have.

The importance of the openness quality has a major effect on the engineering of the system and influences the tradeoff considerations between openness and other system qualities.

3.4.1.6 Extension Building Support

A characteristic that has a vital influence on how many third party developers are attracted to contribute to the product is the extension building support. This includes all artifacts and their quality that are provided as basic support to create system extensions.

A first artifact of this type is *documentation*. The quality of the documentation is mainly determined by how well it guides developers to create their extensions. Properties that influence this quality are among others the amount, the degree of detail, the type (requirements, architectural, low level design, code), or whether it is freely available or for purchase.

A second type of extension building support is *consulting*. Consulting can be offered in different forms. This can include electronic support, for example with chats or forums or on-site consultation. Consulting can also be provided for different tasks like for development, integration, or operation.

And yet another type of extension building support is the *source code* of the open system. The availability of source code can support the understanding of the functional principles of the open system and thus how to create an extension for it. Related to this, the programming language is also a factor that can be considered as supportive or as hindrance. Offering a variety of popular programming languages like Java, C# or Python to build system extensions from will be considered as supportive by third party extension producers. In contrast, only supporting extensions in a single programming language or just languages that are rarely used will be considered as hindrance.

3.4.2 Extensions

System extensions exhibit certain characteristics that distinguish them from other software systems. In the following such characteristic properties are explained in detail.

System extensions are always built for a *concrete open system*. Although it might be possible to create an extension as a product line, as it is often done for apps for different mobile platforms like Android and iOS, the final application is always built for a concrete open system and its technical specifics. This contains some implications:

System extensions are *executed* in the *context* of the corresponding open system. This includes that it is executed on the same hardware and run under control of the open system software. We explicitly exclude the case of extensions being executed on connected separate systems as we either regard this as parts of the same open system if they are provided by the same producer, or as interoperating systems if not.

Another implication is that a system extension is not or *not fully functional without its corresponding open system*. iOS apps for example cannot be executed without the iPhone or iPod and Eclipse extensions need the Eclipse platform to run. Being built on top of existing features and reusing infrastructure functionality makes extensions dependent on the open system.

And a third implication is that in contrast to interoperability, where a certain effort has to be invested to make to systems interoperate, the *integration* of open systems and extensions requires in most cases *few or almost no effort*. This is possible by openness mechanisms that have been implemented in the

open system during development time and that can be utilized by system extensions.

Details on the integration of open systems and extensions are given on below.

3.5 Integration of Open Systems and Extensions

Integration of open systems and extensions means installing the extensions on the deployment platform provided by the open system. It might require additional configuration or customization on both of the sides (the open system and the extension). Several aspects related to integration are described below.

3.5.1 Integration Effort

Open systems built for the integration of system extensions that are built exclusively for this concrete open system. Therefore the integration effort should be close to zero. During the development of the open system, integration mechanisms are established. These integration mechanisms are later utilized by the extensions when they are being integrated into the open system. For complex systems like vehicles or machines it might be required to put some effort for installation and customization. Even in the case of large number devices (for example in a company where hundreds of employees are working and one extension is required for their mobile devices), some integration effort is required. But in general it should not be the main consumer of effort in the spectrum of open system related activities.

3.5.2 Binding Time

The binding time describes the point in time when a system extension is integrated in the open system. Typical categories for the binding time include early and late binding, meaning integration during the system's development or at run-time. For open systems early binding typically does not apply, because open systems are built without targeting a concrete extension. Extensions are in most cases developed, when the open system is already available.

However, for open systems, different binding times are distinguished. A first distinction is whether an extension is integration while the system is executed (run-time binding) or while it is out of operation (configuration time binding). For configuration time binding, a system has to be stopped, the extension be integrated and the system restarted.

If extensions can be integrated while the system is executed, we can again distinguish between different forms. The binding of the extensions can be done in a plug and play manner, where the extension can be installed while the open system is in operation and the extension can be used immediately without re-

starting the system. For example in the case of mobile apps the user can access the features of the extension just after installing it. Alternatively it might be possible to install the extension during run-time, but necessary to restart the system for the features to become available and usable. An example for this is the Eclipse platform.

3.5.3 Integration Mechanisms

Integration mechanisms are means of integrating the extensions with the open system. It covers the installation, configuration, customization and binding of the extensions to the open system. Integration mechanism depends on the type of the open system and also to the policy of the company producing the open system. For example in case of mobile platforms, extensions can be installed directly or through some central extension repository maintained by the producer. Currently iOS supports to install apps only through the App Store whereas Android supports direct installation and also through Google Play. Restrictions on the integration mechanisms give the open system producing companies more control on their systems and can secure their system better. Cloud platform providers also follow the mobile platform mechanism. For example Salesforce provides AppExchange to host extensions for Force.com and those can be directly installed onto the platform.

For desktop applications the extension provider needs to provide some update or installation site so that customers or users of the extension can install the extension directly. Manual installation can also be possible. For example in case of Eclipse, the plug-in providers maintain their update sites and plug-ins can be installed manually or through the update sites.

Extensions may be integrated manually for vehicles or for complex devices by some expert who has in-depth knowledge about the open system. For example the integrator needs to take care whether the system needs to be shut down or some customization or configuration is required and so on.

3.6 Openness and other Quality Attributes

Openness is closely related to other software quality attributes, like interoperability, Extensibility or Flexibility, for instance. However, there are also differences between them. In the following, the similarities as well as the differences to other qualities are discussed in detail. Additionally, the different qualities are related to each other. Depending on the concrete system, openness might either benefit from or challenge other quality attributes and vice versa. In section 3.6.2 some qualities are discussed for which either a positive or negative influence is likely.

3.6.1 Delineation of Openness from Other Qualities

In the following sections, differences and similarities of openness to other common software qualities are discussed. This includes interoperability, flexibility, extensibility, and adaptability.

3.6.1.1 Interoperability

Interoperability is the "readiness" of two systems to interoperate (cf. [3]). As such, the major difference to openness is that interoperability is concerned with two independent and fully functional systems. Each of these systems is operated to fulfill its own designated purpose. The goal of making them interoperate is decreasing the usage and operation costs and has therefore a high importance, but it is typically not a fundamental necessity for their existence. System extensions in contrast cannot be operated without the corresponding open system.

A second difference is that interoperability is always targeted towards an anticipated system or system class and towards a concrete situation of interoperation. This means that when interoperability mechanisms are implemented in a system, there has to be a clear vision of which systems shall be integrated for which purpose at a later point in time. For openness, this is typically not the case. When openness mechanisms are created, it is mostly unknown what kind of extensions will be created and what enhancements they will contribute.

A major difference can also be found in the business models that underlie both qualities. For open systems, the openness quality is an essential, central and inherent quality. It is seen as unique selling point and competitive advantage. Interoperability on the other hand can also create a competitive advantage, but is typically not such a central point as for openness. For interoperable systems their own functionalities or qualities like user experience or performance are the main focus points.

However, there are also similarities. As well as for openness, interoperability has to be established at development time by introducing interoperability mechanisms. These mechanisms are then utilized during runtime or configuration time to realize the interoperation. Typically the more a system is prepared for interoperation during development time, the less effort has to be spent later to integrate two systems. Similar to interoperability, openness mechanisms have to be constructed into systems and are then utilized later by system extensions.

3.6.1.2 Flexibility

Flexibility is the property of a software system to allow conducting certain anticipated changes to the system (expressed in flexibility requirements) with acceptable effort for modifying the system's implementation artifacts. This means that the flexibility requirements are covered by the flexibility potential of the

system [4]. Although flexibility, in the sense of adapting a system to changing requirements, might also be achieved by utilizing openness mechanisms, flexibility as a quality attribute has some fundamental differences compared to openness.

A first difference can be seen in the organizational setting and business models on which openness is founded. Although flexibility might also generate a selling point, a business model in which an ecosystem evolves around a product as for openness is not present. Also, third party organizational units are not foreseen for flexibility. Although, there is no definite organization set in the definition, in most cases it will be the developing organization that adapts the system to changing requirements.

During the development of open systems, it is mostly unknown, what kinds of system extensions will be built in the future and what kinds of enhancements they are going to contribute. Flexibility in contrast always targets a concrete anticipated system or class of systems. So, systems are never flexible in general.

3.6.1.3 Extensibility

Extensibility is a special kind of flexibility. It relates to the ability of a system to allow change for adaptation to changing requirements without the need for large investments of effort. In contrast to flexibility, it focuses on addition of new elements or functionality, whereas flexibility focuses on change in any way.

As it is so closely related to flexibility, the differences to openness are similar. Extensibility is not related to business considerations in the way openness is. As well it does not necessarily include third party organizations that contribute to the system.

3.6.1.4 Adaptability

Adaptability refers to the ability of a system to allow substantial customization through tailoring by users, which means, after it has been delivered. As such, it has similarities to openness. High customizability is one of the main advantages of open systems.

In contrast to openness, adaptability does not demand any specific mechanism of how to achieve customizability. Specifically, customizability does not need to be realized by a system extension concept as it is for openness. Extensive configuration mechanisms hard coded in the system might fulfill adaptability requirements in the same way as can be achieved with extensions.

Another fundamental difference is that adaptability does not demand any third parties contributing to the adaptability of the system. It is more likely, that adaptability mechanisms are implemented by the organization that produces

the system. This implies that one of the main ideas of openness, a business eco system evolving around a product is not present as well for adaptability.

3.6.2 Potential Impacts on Other Quality Attributes

Introducing openness can have impacts on other quality attributes of the system. This includes potential positive impacts, where achieving other quality attributes can benefit from openness, as well as potential tradeoffs, where openness challenges the achievement or retention of other qualities. In the following, several of these potential influences are described. It is important to note that these influences must not necessarily exist as described. There only a tendency is given. In a concrete project context the real effects on other quality attributes have to be analyzed separately and in detail. Figure 7 shows an overview of the potential conflicts on other qualities that are explained in the following.

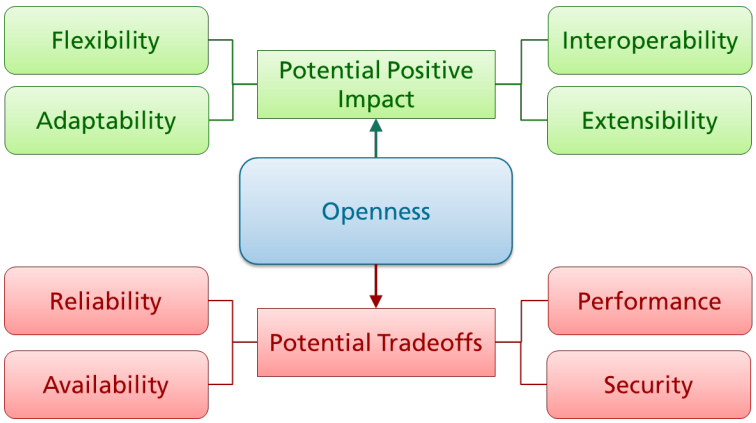

Figure 7. Potential Impacts on Other Qualities.

Potential Positive Impacts

Primarily quality attributes that are related to openness may potentially benefit from this property. In particular the ones described in section 3.6.1might be achievable with less effort by utilizing openness mechanisms.

To establish interoperability between an open system and a target system it is imaginable to use system extensions. The producer of the closed system could provide a system extension to create the integration of both systems. For example BluJet could create system extensions for open tractors created by John Deere, to make their implements controllable through the tractor.

Achieving flexibility and extensibility can be supported by openness. Flexibility mechanisms for anticipated change can be based on the openness mechanisms that might already be present in the system. So it might be efficient to create a

system extension to add a new functionality that satisfies a changed requirement instead of changing the open system internally.

Finally, adaptability to a certain degree is implicitly given for open systems. Normally the user or customer (they might be the same or different) are normally free to use and integrate any system extension they want to use. In this way the system can be adapted according to their needs. However, there might be cases in which the available system extensions do not provide the desired form or degree of adaptability, which will make the implementation of other adaptation mechanisms necessary.

Potential Tradeoffs

Openness can possibly challenge the achievement of other quality attributes. The following qualities might need special attention of architects when designing open systems.

Openness implies making internal system elements available to software contributed by third party organizations. This contains an inherent security risk. So it is vitally important to introduce adequate security concepts in open systems. This mainly includes tactics to prevent access of restricted system elements that have not been opened for usage or enhancement by system extensions. In this way, extensions shall be prevented from running malicious code that could for example, read protected data, access restricted functionality or write data to restricted regions.

Accordingly open systems need to ensure that system extensions do not prevent the system from fulfilling performance requirements. There might be risks of extensions that perform long running computations prolonging the response time of systems or allocating an inadequate amount of system resources. Open systems need to apply mechanisms that ensure the open system to fulfill their own performance requirements independently from the installed extensions.

Finally, availability and reliability might be challenged by openness. Installed extensions might possibly fail and affect the overall functioning of the system. Therefore it is important to implement mechanisms that prevent effects of extension failures to spread onto the complete system. If, for example, a system extension becomes unresponsive, the open system needs to detect and recover from the failure.

4 Classification of Open Systems

Based on the conceptual model, it is possible to characterize and classify each system with an open architecture, together with corresponding extensions, business and organizational settings and integration mechanisms. In the following sections a classification schema is introduced that summarizes the single aspects. The example scenarios that have been introduced in section 2 are then revisited and using the classification schema completely characterized.

4.1 Classification Schema

The classification schema captures in a concise and structured form all the factors and possible corresponding values described in the conceptual model. Using this schema it is possible to characterize any situation in the context of open architecture systems.

Table 1. Classification Schema.

Factor	*Values*	*Description*
	Organizational Setting	
Open System Producer	The name of the organization creating the open system.	The open system producer is described in section 3.3.1.
Extension Producer	The name of the extension producing organization in a concrete setting.	The extension producer is described in section 3.3.2.
Operator	The name of the organization that operates the open system.	The operator is described in section 3.3.3.
Integrator	The name of the organization integrating the open system with the extension	The integrator is described in section 3.3.5.
	Open System	
Business Motivations	• New Features • Customizability • Product Integration • License Costs • Consulting Services • Increased Visibility • Lock-in Effect	The business motivations that found the decision of the system producing organization to create an open system. Details are given in section 3.2.1.

System Type	• Vehicle / Machinery • Desktop Applications and Platforms • Mobile Devices and Platforms • Distributed and Cloud Services • Computing Hardware	The type of the open system as described in section 3.4.1.1.
System Boundaries	• Single Open System • Compound /Integrated Open System	Open system can consist of multiple linked single systems. What can be regarded as open system depends on what system parts are provided for extension. Details are explained in section 3.4.1.2.
Opened System Levels	• Hardware • Data • Functions • User Interface • Process	The system levels for which extensions are allowed. Details are given in section 3.4.1.3.
Access Rights	• Read • Change • Add	The access rights that are provided to system extensions for the single system levels. The rights are explained in section 3.4.1.4.
Initial State Capabilities	• Platform • Functional System	The capabilities that an open system provides upon delivery and thus, the importance of the openness concept in the corresponding system. Details are given in section 3.4.1.5.
Extension Building Support	• Documentation • Consulting • Source Code	Artifacts and activities that are offered as support for building extensions for an open system. The factor is described in detail in section 3.4.1.6.
Extensions		
Business Motivation	• Infrastructure Usage • Image Benefit • Customer Base • Low Prices	The business motivation of extension producers to create an extension for the corresponding open system. Details are given in section 3.2.2.

Contribution Levels	• Hardware • Data • Functions • User Interface • Process	The system levels for which the extensions provide contributions. Details on the system levels are described in section 3.4.1.3.
	Integration	
Needed Integration Effort	An effort value (PH, PD, PM,…)	The effort needed to integrate a concrete extension into the corresponding open system. Details are given in section 3.5.1.
Binding Time	• Runtime • Configuration Time • Build Time	The time at which an extension has to be integrated in an open system. Details are given in section 3.5.2.
Integration Mechanisms	An integration mechanism (expert engineer, app store, …)	The integration mechanism that is used to integrate the extension with the open system.

4.2 Revisiting the Example Scenarios

The three scenarios described in section 2 are revisited in this section following the classification schema presented above.

4.2.1 Agriculture Scenario

Table 2. Classification Agriculture Scenario.

Factor	*Values*
	Organizational Setting
Open System Producer	The tractor manufacturing company
Extension Producer	The software company that develops Cloud-Loader add-on for the tractor
Operator	The farmer
Integrator	The extension producer
	Open System
Business Motivations	• New Features • Customizability • Product Integration

	• License Costs • Consulting Services • Increased Visibility • Lock-in Effect
System Type	Vehicle / Machinery
System Boundaries	Single Open System
Opened System Levels	• Functionality • Data • User Interface
Access Rights	Read
Initial State Capabilities	Functional System
Extension Building Support	• Documentation • Consulting
Extensions	
Business Motivation	• Infrastructure Usage • Image Benefit • Customer Base
Contribution Levels	• Functions • User Interface • Process
Integration	
Needed Integration Effort	2 PH
Binding Time	Configuration Time
Integration Mechanisms	Through expert engineer

4.2.2 Eclipse Desktop Application Scenario

Table 3. Classification Eclipse Desktop Application Scenario

Factor	*Values*
Organizational Setting	
Open System Producer	The Eclipse Foundation
Extension Producer	The software company that built BPM plugin for Eclipse. And also the customer company who further extends the BPM plugin
Operator	The customer company of BPM plugin
Integrator	The operator.
Open System	
Business Motivations	• New Features • Customizability • Product Integration

System Type	Desktop Applications and Platforms
System Boundaries	Single Open System
Opened System Levels	• Functions • Data • User Interface
Access Rights	• Read • Change • Add
Initial State Capabilities	Platform
Extension Building Support	• Documentation • Source Code
Extensions	
Business Motivation	• Infrastructure Usage • Low Prices
Contribution Levels	• Functions • User Interface
Integration	
Needed Integration Effort	No integration effort required
Binding Time	Runtime
Integration Mechanisms	Directly through update site

4.2.3 Cloud Platform Scenario

Table 4. Classification Cloud Platform Scenario.

Factor	*Values*
Organizational Setting	
Open System Producer	SalesForce.com
Extension Producer	The software company that produced app for Outlook integration
Operator	The SalesForce's customer company
Integrator	In this case it is the extension producer
Open System	
Business Motivations	• New Features • Customizability • Product Integration • License Costs • Consulting Services • Increased Visibility • Lock-in Effect
System Type	Distributed and Cloud Services

System Boundaries	Single Open System
Opened System Levels	• Functions • User Interface • Process
Access Rights	• Read • Change • Add
Initial State Capabilities	Functional System
Extension Building Support	• Documentation • Consulting
Extensions	
Business Motivation	• Infrastructure Usage • Image Benefit • Customer Base • Low Prices
Contribution Levels	• Data • Functions
Integration	
Needed Integration Effort	2 PH
Binding Time	Runtime
Integration Mechanisms	Through AppExchange (app hosting site)

5 Summary

Openness of software systems constitutes a new way of collaboration between software development organizations – on the one hand the open system producer and on the other hand the extension producer (i.e., the consumer of the open system).

In this report we defined the term "open architecture" and present a conceptual model for openness within a software ecosystem. We further provided a classification schema, which helps organizations in characterizing the situation of their systems and gives guidance to development organizations on whether or not to invest into openness and what consequences to expect.

This report forms the basis for other reports in the Innovation Cluster "Digitale Nutzfahrzeugtechnologie (DNT)", which provides guidance on architecture-significant requirement for open systems and characterize engineering patterns for open systems and provide examples application and implementations in the context of the Living Lab "Smart Farming".

References

[1] Skud, "Defining openness: open source, open data, open APIs, open communities, and more," *infotropism*, 2009. [Online]. Available: http://infotrope.net/2009/11/16/defining-openness-open-source-open-data-open-apis-open-communities-and-more/.

[2] M. Zimmerman, "Open vs. open vs. open: a model for public collaboration," *We'll see | Matt Zimmerman*, 2009. [Online]. Available: http://mdzlog.alcor.net/2009/10/26/open-vs-open-vs-open-a-model-for-public-collaboration/.

[3] Olbrich, Steffen ; Weitzel, Balthasar ; Rost, Dominik ; Naab, Matthias ; Kutepov, Glib: "Decomposing Interoperability: A Quality Attribute in the Balance of System Usage, Operation and Development" Kaiserslautern, 2012. (IESE-Report; 104.12/E). - Reportnr. 104.12/E

[4] M. Naab, "Enhancing Architecture Design Methods for Improved Flexibility in Long-Living Information Systems", PhD Theses in Software Engineering, Fraunhofer IRB Verlag, 2012.

Document Information

Title:	Openness as an Architectural Quality
Date:	November 2013
Report:	IESE-075.12/E
Status:	Final
Distribution:	Public Unlimited