Opening the Ecosystem Flood Gates: Architecture Challenges of Opening Interfaces within a Product Portfolio

Slinger Jansen

Information and Computing Sciences, Utrecht University, The Netherlands
slinger.jansen@uu.nl

Abstract. Technology firms are increasingly opening up their products to develop an active ecosystem of developing partners around it. Both opening up products and organizing a developer ecosystem around an organization are non-trivial. In this paper we provide a case study of a leading communications technology firm that opened up and platformized 11 product lines. First, we identify and describe four architecture patterns that are applied multiple times across these product lines. Also, the software ecosystems initiative is centralized in one central department, which has created a central knowledge hub for the creation of a software ecosystem. We highlight the guidelines collected by the central department, to assist technology firms in the platformization process and support them in their own software ecosystem creation efforts.

Keywords: Software platforms, Software Ecosystems, APIs, Case Study, Extendible Product Lines, Extension Patterns

1 Introduction

The creation of partner and developer ecosystems around IT companies is gaining interest rapidly. IT companies observe the successes that can be achieved with app stores, hackathons, open source developer communities, and other initiatives that drive software ecosystems. The creation of an ecosystem around a traditional IT product, however, is far from trivial and IT companies are looking for approaches to open up their products and have them adopted by communities of active developers who wish to co-innovate and share in the wealth created by the products and its auxiliary materials.

IT companies aim to create active developer communities and ecosystems around their products. We define developer ecosystems as a set of software developers functioning as a unit and interacting with a shared market for software artefacts. We ask the reader to observe the parallels between the definition on developer ecosystems and the definition on software ecosystems: a software ecosystem is a set of actors functioning as a unit and interacting with a shared market for software and services, together with the relationships among them [7]. We also state that the term (open source) developer ecosystem is a synonym for (open source) developer community [2].

The research domain of software ecosystems is still in its infancy [10]. Researchers and IT companies are curious about new theories, methods, and techniques for the initiation, development, and grooming of software ecosystems. As such, there is an urgent need for examples and case studies that exemplify excellent practices, for theory formation and for teaching practitioners lessons.

Large IT companies are currently launching and running developer ecosystems. The challenges for a large IT company compared to a web start-up, however, are much larger, as large IT firms typically have many products organized in product lines, whereas a small web start-up will only have one or two domain APIs that need to be opened up. From this crucial difference, many new challenges arise. First, a small web start-up will start with a blank slate, whereas a large IT company has different product lines that may already be involved in supporting a software ecosystem of its own, with varying success and stages of maturity. Furthermore, due to the large range of different technologies that are adopted by a large IT firm over time, different entry points are required for each product, different types of participants are active in the ecosystem, and different business models need to be applied over those different products.

In this paper a case study is presented of an IT firm with a large product portfolio that with one initiative is hoping to open up ecosystems around a large set of its products. The lessons provided stem mostly from the software architecture domain: we illustrate how a product portfolio can be opened up using a generic platformization approach. Enabling an ecosystem requires a level of openness for a platform: without any extension mechanisms for third parties it is practically impossible for a software ecosystem to exist [1, 6]. The openness level of a software platform is also a powerful tool for platform owners, as their choices determine the flexibility and extendibility of the platform, and subsequently of the ecosystem. A platform that is too open runs the risk of giving away its competitive uniqueness for free, whereas a platform that is too closed risks not being interesting enough for platform extenders in the ecosystem. The platformization approach is explained using four architectural extension patterns that were applied 21 times across 21 products.

We continue this paper with a description of the case study and the use of grounded theory for explorative research in Section 2. In Section 3 the case of NetComp is described. A detailed description is provided of the implementation of the software ecosystems initiative at NetComp: the managerial approach, the technical approach, and the developer ecosystem approach are discussed. In Section 4, four product extension patterns are described and details are provided of the historical background of the technologies provided at NetComp and on how the patterns were influenced by technological and strategical advancement of NetComp. To illustrate the observed patterns, NetComp's Telepresence product line is used to illustrate the observed patterns in Section 5. In Sections 6 and 7 we analyze the efforts at NetComp and identify the challenges of undertaking such an initiative: both from the architecture and managerial perspective. Finally, we summarize our findings in Section 8 and hint towards a catalog of

extensibility patterns that describes the different methods that can be used to extend a software product into a platform.

2 Case Study Method

Context: The context of the case study is NetComp (company and department names anonymized), a relatively young international firm that produces hardware and software products for enterprise and carrier communications. The firm has around 200,000 employees and is growing rapidly. The company has a worldwide presence.

Case study type: The case study can be seen as a participant case: the first author has worked alongside the FloodGate Department, a department that is burdened with the ecosystem initiative of the company. The main responsibility of the FloodGate Department is to expose current products to the developer ecosystem of NetComp and to build out the developer ecosystem. Please note that alongside the FloodGate Department are more commercial departments oriented around partnering, business models, etc. The FloodGate Department mostly deals with technical issues, development documentation, and the developer ecosystem. Please also note that the researcher was not involved in any of the decisions described in this paper, as his work focused more on the growth and grooming of the developer ecosystem.

Unit of Analysis: The units of analysis for this study are the architectures and extension points for each of the products in the product lines. Furthermore, the FloodGate Department and its responsibilities have been a unit of analysis as well.

Method: The data about the products and architectures has been collected through interactions with the FloodGate Department and in some cases through direct interaction with the product units. Also, several interviews with extending partners have taken place. The case study has been exploratory: multiple topics for study have been extracted. The current report (i.e., this paper) is the first in a set of reports. The methods followed were document study, interviews, interactions through the in-company chat system, and frequent e-mail interaction. The communication through digital channels helped solve translation issues. In all of the interviews different translators were present. The interviews have been recorded. Through inductive reasoning, topics have been extracted, highlighted, and grouped, using a digital folder system. For the study at hand, document study has been the main source of the material that is presented.

3 Case Report: Studying 11 Product Lines in NetComp

The starting point for this research has been the *FloodGate Department*. The FloodGate Department is a horizontal department in NetComp that is responsible for the ecosystem initiative of NetComp. The FloodGate Department was founded to create one unit within the company that is dedicated towards enabling partners to extend the most successful products of NetComp. As NetComp has

a huge product portfolio, the unification of these efforts results in a knowledge hub on the creation of extendible interfaces for software and hardware products. The end goal of the FloodGate Department is to create welcoming and open software ecosystems for partners to participate in.

The FloodGate Department has been built upon several loose initiatives in the product lines to make their products extendible. The product lines for unified communications, for instance, had been creating extendible products and platforms for several years already. Currently, the FloodGate Department opens up capabilities in 11 product lines, in more than 20 products. These product lines have amassed between tens and hundreds of ISV and implementation partners (i.e., extension builders) that are dependent on NetComp products for their revenues.

As the FloodGate Department has executive backing, many product units have found a strategic partner in the FloodGate department: their resources are not influenced by one product's success and they have the knowledge on how to open up any kind of product. Some product lines were reluctant to support the FloodGate initiative at first, but as time progresses, they too see that the FloodGate Department plays an important and strategic role in enabling product units to build their own partner network.

The *FloodGate Components* consist of the different software components that are managed by the FloodGate Department. These components are FloodGate servers, which are typically installed alongside NetComp products, extension libraries (JARs and other SDKs), and controls (like OCX controls). Another important concept in the case study are the *FloodGate Labs*, where partners can remotely test their software against NetComp hardware. These labs are located in one office building and contain hardware test set-ups that can be worked with in a timesharing manner. The FloodGate Labs are described in Section 3.2.

3.1 Joining the FloodGate Initiative

The FloodGate Department supports eight programming platforms (.Net, C(++), Java(script), Ruby, Delphi, JSP, PHP, and Python) and six operating systems (iPhone, Android, Linux, OpenSuse, Windows, and Mac OS X). This variation has mostly been evolutionary: as new product lines join the FloodGate initiative they are bringing in new domain specific technologies. It is important to note that in many cases the FloodGate Department does little more than provide documentation and an open interface for the products. The responsibility for the product and its interfaces remains with the product departments, although in quite some cases the FloodGate Department has inherited the software development of complex extension servers (more on this in Section 4).

NetComp supplies funding to both the product team and the FloodGate Department for each new product that starts creating FloodGate Components. There is no formal procedure for joining the FloodGate initiative, but the process is based on common practices. The process consists of the following steps: (1) Assess suitability of the product, (2) design new architecture for the product to enable an open extendible platform, (3) publish the products' FloodGate

Components (typically an SDK). In the case of some strategic products, the FloodGate Department has initiated the collaboration themselves. The organization does currently not evaluate the financial results of opening up parts of products. The ecosystem initiative is strategic and is assumed to be useful for the whole firm. As it is hard to predict whether some of the products are going to be successful as platforms, NetComp mostly works on feedback from partners and channel managers.

In a typical case, the FloodGate Department is approached by a product unit first. They will explain their needs for extension, the (potential) size of the partner network, and the efforts they think are required to open up their products and platforms. The FloodGate Department assigns a project leader to the product unit, who from then on is responsible for all contact with the product unit. The project is started and an inventory is made of the efforts required to open up the products. Ideally, the FloodGate Department deploys their FloodGate server software next to the products and platforms and uses this server as an abstraction layer between the product unit's products and the extensions built by partners. The FloodGate Department and the product unit develop the capabilities in lock step: first the product is opened up further, and then the FloodGate components (more on these later) are evolved. When the software is considered ready for publication documentation is created on the FloodGate ecosystem hub, a web site where partners are gathered and supported.

As the platform is adopted by partners new responsibilities are introduced. The FloodGate Department remains responsible for the maintenance and development of the extension software, its co-evolution with the products, and partner support. The product units remain responsible for the development and maintenance of the products and the support of partners from a commercial perspective.

The commercial departments of specific products are responsible for managing partners. The FloodGate Department is responsible for solving technical problems that partners face. Unfortunately, there is little to no sharing contact data between the business departments and the FloodGate Department, which leads to two separate databases with partners, i.e., those that collect incentives from the business departments versus those that ask questions to the FloodGate Department. Both the business departments and FloodGate Department are calling for a unified partner management system. The responsibilities are mapped out in Table 1.

The FloodGate Department is responsible for receiving extensibility and product requests from partners. The project leaders in the FloodGate Department forward the product related questions to the product units and implement the extensibility requirements where possible. Product management is in the hands of the product units. The FloodGate Department is responsible for opening up the architecture, but the interfaces have typically been prepared by the product departments. Interestingly, some of the product units are no longer independently planning new features for their products, but involve a mixed team with members from the FloodGate Department.

Table 1. Responsibilities divided between product units versus the FloodGate Dept.

	FloodGate Department	**Product Departments**
Product management		
Release planning	One week after each product release.	The product departments release their versions independently.
Requirements engineering	From the product departments and partners.	From partners and end-customers.
Extendibility requirements	From the product departments.	From partners.
Software delivery	Independently delivers components.	Independently delivers products.
Development and Support		
Architecture development	Gives guidelines in regards to interfaces required.	Develop their own product based architecture.
Error messages	From the FloodGate Components.	From the internal components.
Documentation	About FloodGate Components.	About the product.
Support	To the product departments and to partners.	To the partners (product related) and customers.
FloodGate Labs	Completely responsible.	Helps FloodGate Dept. setting up the products in the labs.
Business aspects		
Partner management	Developer community.	Partner community.
Financial responsibility	Centrally coordinated.	Revenue based.

When a new version of a product is released, the FloodGate Department typically responds within one week with an update to the extensible components as well. As the FloodGate Department is kept up to date of a product's progress and release schedule, they are developing the extensibility components parallel to the product development. In many cases they FloodGate Department can release on the same day as the product unit does. The FloodGate Department is somewhat hard to manage because of this: as the product release schedules are not coordinated, the FloodGate Department has different work loads at different times. The FloodGate Department and the product department organize meetings between once and twice per month to coordinate new releases, new development efforts, and ecosystem challenges.

3.2 FloodGate Labs

The ecosystem enablers that may be beneficial for one product are not beneficial for another. For example, some of the products in NetComp, such as IP Cameras and routing equipment, require access to test hardware. NetComp prides itself for providing access to a large laboratory in the cloud, that can be approached by partners at specific times (effectively timesharing the lab). Partners positively evaluate this practice, as they do not need to procure expensive test setups for their own development. NetComp leverages its own IP camera system to show that the hardware controls being called from the lab actually have the desired effect by pointing IP cameras at the hardware, such as servers, switches, and ironically, IP cameras. Please see figure 1 for an example of a movable IP camera that is used to monitor servers in the FloodGate Labs.

For some of the software intensive systems this is not beneficial, however, as it can be less cumbersome to just buy the required hardware or create a virtualized test setup. Partners complain that many of the platforms in the

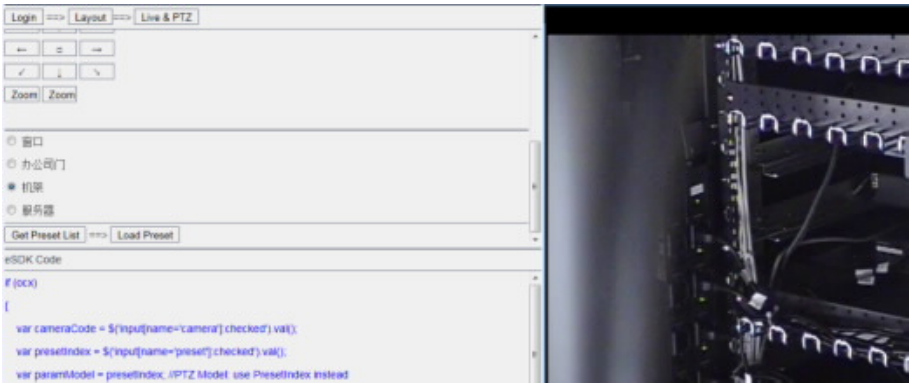

Fig. 1. An IP camera is pointed at a piece of hardware to show API users that the device status changed according to their calls. The IP camera can be moved to look at other adjacent devices as well with simple controls.

organization can be tested in a virtualized environment or through simulators as well. NetComp is currently building such simulators, to make partners less dependent on FloodGate Labs.

4 Product Extension Patterns

NetComp has always specialized in manufacturing hardware devices, such as routers, switches, and IP cameras. As time progressed and the organization became more mature, multiple abstraction layers have been required over the devices. This can be found in, for instance, the telepresence system that has an advanced management interface that can plan telepresence meetings across a network of telepresence devices, or the abstract datacenter management layer that can control several storage servers simultaneously. In NetComp four levels of abstraction have been observed. The levels of abstraction are found in Figure 2.

- **Server Level -** NetComp has traditionally manufactured servers. The software interfaces to those servers were always of concern, but as in the early years many of the implementation projects were in fact done by NetComp, these interfaces were usually in poor shape in terms of software quality. As the market for software has become more commoditized, however, these interfaces to servers have become better managed, higher quality, and accessible by third parties through APIs.
- **Server Management Level -** As NetComp grew, there was an increasing need for management infrastructures that controlled large numbers of device servers, such as IP cameras, routers, and storage servers. These management infrastructures typically use existing protocols for controlling devices, such as the Simple Network Management Protocol, and can interface with hardware from other suppliers as well.

– **Federated Servers Level -** As NetComp started orienting towards more advanced markets and specifically targeting larger enterprises, new requirements were introduced for telepresence, single sign-on, and device management. Whereas before it could focus on building the best IP camera, it now has to focus on providing "the best" federated infrastructures for heterogeneous hardware, both from NetComp and third party hardware providers. NetComp currently provides several such federated infrastructures, for instance for unified communications, equipment dedicated to communication in a specific communication bandwidth, and IP cameras. These federated infrastructures typically consist of several software solutions on different servers, but are conceptually unified into one coordinating server.
– **Advanced UIs Level -** At the highest level of abstraction, NetComp enables mobile and other advanced interfaces to the federated infrastructures. This, for instance, enables the creation of NetComp applications for Smart Cities, such as a mobile app that can control a set of federated IP cameras or desktop apps that can control alarms across server federations.

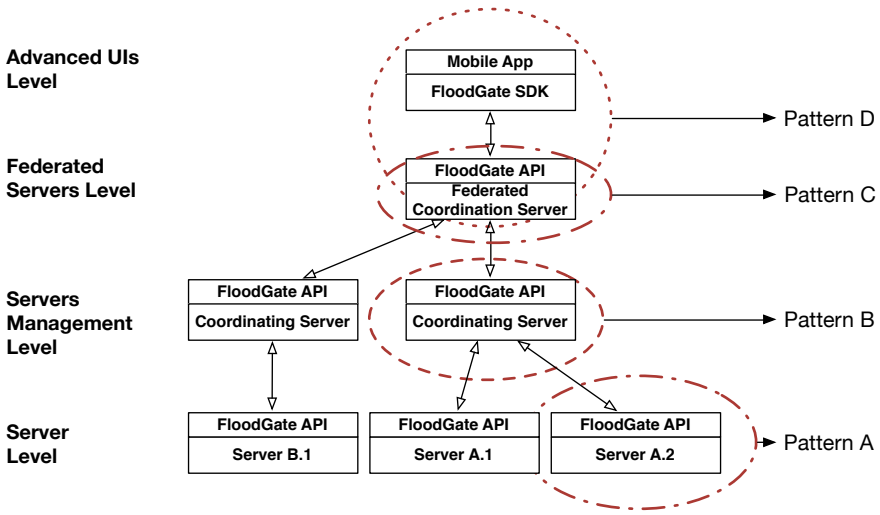

Fig. 2. Three extension patterns are found multiple times in the products of NetComp. The patterns are applied multiple times across different products and product lines.

Figure 2 shows more than just the levels of product abstraction that NetComp offers. It also models the four different extension patterns that are employed by the FloodGate Department to open up NetComp's products. In the following overview, each of the extension patterns is described. The overview is summarized in Table 2.

- Extension Pattern A: Simple Server Extension - To provide third parties with opportunities for controlling and interacting with NetComp hardware, many of the hardware products can be extended with a simple server. Several different mechanisms are applied to open up the servers. In some cases a software switch can be flipped, but more frequently, a separate JAR needs to be deployed on the hardware to open up its capabilities.
- Extension Pattern B: Coordinating Server Extension - In the case of coordinating servers, the patterns and technologies used are similar to that of Pattern A. In many cases, however, this feature is provided and switched on automatically. It is interesting to see that for Pattern A the technologies used are technologically close to the device (i.e., JARs for Java servers, DLLs for Windows based servers, etc.) whereas for Pattern B more abstract technologies are used, such as SOAP and REST.
- Extension Pattern C: Federated Server Extension - Federated servers are powerful mechanisms that can abstractly control heterogeneous devices in a network. These federated servers are even more frequently used for extension by customers and partners, as these are able to control all the customer's devices. These "servers" also are closer to the end-user. An example is the Bring Your Own Device solution, which provides access to different features in that domain, such as a single-sign on server, an asset management solution, and a software repository for mobile devices, also known as enterprise app stores.
- Extension Pattern D: Advanced User Interface Extension - To provide access to the different infrastructures in the enterprise, NetComp supplies different SDKs and even reference implementations for customers and partners. Examples of such SDKs are the telepresence control SDK for Android and iOS, enabling the development of apps that allow for initiation, planning, execution, and termination of telepresence calls. Many of the apps built by partners are geared towards end-users, even though the SDKs are typically open and allow for much more.

One extra way of extending NetComp products is through third party platforms and products, such as Outlook, VMWare, OpenStack, Microsoft Lync, etc. NetComp supplies software that already extends these platforms. It is interesting to see that as we move up to higher levels of abstraction, more abstract protocols and technology-agnostic extension mechanisms are found. Whereas at the lowest level controls are typically built in highly technologically native environments (Java SDKs, linux libraries, etc.), at the higher levels mechanisms become increasingly abstract (REST, SOAP). This is in part caused by the time at which these higher level abstractions were built, but also because more partners require different types of technologies for integration at higher levels.

5 Illustrative Case: the Telepresence Product Line

One of NetComp's most successful product lines is the telepresence. NetComp manufactures many different systems for this domain: from HD television mounted

Table 2. The observed extension patterns, their occurrences (in a total of 21 extendible components), and the observed technologies at NetComp. Please note that for some product lines multiple extension patterns are observed. In Section 5 all four are observed in one product line.

	Occur.	**Extension mechanism**	**Extension technology**
Pattern A: Simple Server Extension	6	1. Turn on software switch 2. Deploy server on device 3. Deploy server on another device	Standalone executable, JARs, .so libraries, powershell libs, DLLs
Pattern B: Coordinating Server Extension	3	1. Turn on software switch 2. Deploy server on device 3. Deploy server on another device	Standalone executable, JARs, SOAP, REST
Pattern C: Federated Server Extension	4	1. Deploy server on another device 2. Buy secondary dedicated device	Standalone executable, JARs, SOAP, REST, SNMP, Python
Pattern D: Adv. User Interface Extension	8	1. SDKs for (mobile) apps 2. Provide client controls	Android, iOS, OCX

cameras to smart microphone and speaker interfaces. These systems are most effective when they integrate well with the infrastructure of a customer organization. Meetings must be planned through office applications such as Outlook, for instance.

Traditionally, managers of telepresence systems performed their operation and maintenance tasks through the NetComp Service Management Center (SMC), a server that is dedicated to the detection and management of telepresence devices. The SMC can directly access features of the telepresence devices and execute device-specific commands, such as start, stop, and record commands.

As customers started developing their systems, however, they also attempt to integrate other NetComp (generic IP cameras) and third party hardware (HD videoconferencing). NetComp added a component to their SMC called the Converged Gateway: a product that enables interaction through a unified interface with other (sometimes non-telepresence) devices.

In the scope of the FloodGate initiative, the capabilities of the devices, the SMC, and the converged gateway are opened up for extension by third parties. This is done through the FloodGate server, which is independently deployed in the IP network. The FloodGate server can be approached directly with SOAP calls. Furthermore, there are JAR libraries available to quick start third parties with the development of the advanced choreographies that are necessary to negotiate advanced telepresence scenarios. The JARs available concern mostly the SMC, but also contain more high-level abstractions, with the most interesting one being the eHealth JAR, containing specific capabilities for remote health care.

The network deployment of the telepresence components is modeled in Figure 3. Extenders have the option to approach the FloodGate server through the SDK or through its direct API, using SOAP. There exists a link between the FloodGate server and endpoint telepresence devices in the network, but this is no longer documented, as extenders are advised to use the SMC interface. It is interesting to observe that the extension patterns identified in Section 4 are all

found in the telepresence product line. First, the SIMPLE SERVER EXTENSION PATTERN is found in the fact that it used to be possible to directly address endpoint devices in the telepresence deployment. Secondly, the COORDINATING SERVER EXTENSION PATTERN is found in the SMC extension possibilities. The SMC controls the whole deployment of telepresence devices and through its FloodGate interface can be controlled by a third party application. Thirdly, the FEDERATED SERVER EXTENSION PATTERN is found in the converged gateway, enabling hardware from others and non-telepresence devices, such as simple IP cameras, to also be controlled by third parties. It must be mentioned that the converged gateway and SMC are no longer deployed separately and are always deployed together. At the highest level we observe the ADVANCED USER INTERFACE EXTENSION PATTERN, as the JARs provided give third parties the opportunity to quick start the development process and develop domain specific solutions, like the telemedicine library offered.

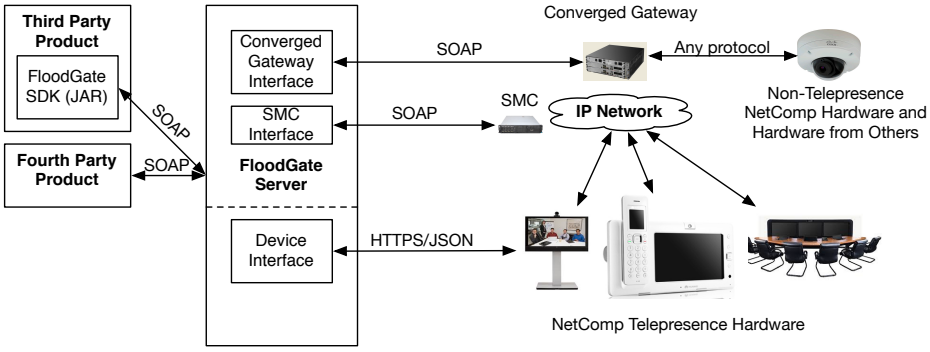

Fig. 3. A typical telepresence network deployment and its extension possibilities. The dotted line indicates that it is still possible to address telepresence devices directly, but it is no longer supported or documented. Lines without a label indicate proprietary protocols that are not visible to third parties.

6 Openness and Architecture Challenges

The FloodGate Department initiative is generally experienced positively by the product departments. Product units get to focus on their product innovations, while extensibility questions and partner support are delegated to the FloodGate Department. Furthermore, as the FloodGate Department has strategic support in the organization, much needed resources in the product units can be used for "regular" product innovation. The centralized approach results into (architectural) challenges that are experienced across different product units.

How open is open enough? There is a constant discussion between partners and NetComp about how open products are. For example, there is a rich tool suite for unified communications, that provides features such as instant messaging, document sharing, voice chat, video chat, and screen sharing. The tool suite is packaged into an extendible client. Partners are calling for modularization, as they do not wish to use the client, but embed smaller features in their own tooling, such as mobile applications. NetComp needs to strategically evaluate such requests: is the call for such modularization going to add value for customers? Will security be compromised? And can profit still be made when partners can replace components so easily? These decisions are typically made by the product lines, with support from the FloodGate department.

How must documentation be standardized for partners? One of the biggest challenges for NetComp has been to standardize across the product units. When looking at the documentation for 3rd parties, for instance, some of the documents are supplied in .chm format (a documentation format that is specific to Microsoft) whereas other documents are supplied through online web content management systems, and as Word documents. Furthermore, the look and feel of the documentation is different across different products and sometimes even for different documents (Java documentation versus C++ documentation, in one instance) about the same product. An improvement initiative has been undertaken to bring all documentation to the web.

How must error messages be handled, communicated, and supported for partners? One of the more interesting discussions at NetComp is about error messages. As we were discussing the quality, findability, and reproducibility of the error messages, it was quickly uncovered that there are actually two different classes of error message: those that come from the FloodGate Components and those that are generated by the products. The FloodGate Department is responsible for the error messages generated by the FloodGate Components, whereas the product units are responsible for the error messages that are generated by these lower layers. The FloodGate Department is running into problems with these: partners call with questions about product error messages, whereas they are only capable of answering questions in regards to the FloodGate Components and their error messages. The FloodGate Department needs a mandate to force product units to regularly update error message documentation and improve them where necessary. Simultaneously, the FloodGate Department is responsible for providing the product units with an infrastructure in which they can publish their error messages and documentation.

How must crashes propagate through the systems? When one of the products crashes, the FloodGate Components, typically a separate server, keeps running. The product units have not been instructed on how to inform the FloodGate Components about crashes and the like. Partners are expected to solve this problem themselves. Should a product crash, it simply becomes unavailable to the FloodGate Components.

How must extensions be secured? In the communications industry security is a major concern. The FloodGate Department architects are responsible for

executing and checking security guidelines. These guidelines are well documented and well managed in NetComp. The architects have three levels of security check in place, which we cannot share for reasons of confidentiality. However, we are allowed to illustrate some of the guidelines that are used by the architects. At the first level, the architects look at data leaks, unlawful interception, and privacy protection. At the second level, the architects have more advanced steps, like data encryption, attack and integrity protection, and log auditing. At the third level, the architects apply tools like virus protection, security hardening, protected installations, database hardening, and some guidelines for partners on security. An interesting observation is that NetComp presently shares little of this knowledge with partners, whereas partners can greatly benefit from security audits. There are many ecosystem opportunities here: partners can be audited, certified, and trained in the domain of security. NetComp is evaluating these different options presently.

How must partners be convinced to deploy newer versions? As the hardware running for customers is generally deployed and then left alone, so are the FloodGate Components. This results in situations where the FloodGate Components running on extendible hardware is running far behind the most recent version, making it harder to develop against. It is, however, a challenge to convince partners to update the software running on the hardware and its accompanying FloodGate servers without any business incentive. Simultaneously, however, when a customer wishes to acquire extended features through a NetComp partner, all hardware drivers must first be brought up to date. The FloodGate Department is working on a policy to incentivize partners to upgrade software, even when there is no direct need for the partner to do so.

7 Analysis, Discussion, and Related Work

The FloodGate Department is relatively new: many product lines developed similar interfaces before the FloodGate Department was implemented in full. It is impossible to say whether the extension patterns were implemented independently by the product units, although we have good reason to believe this to be the case. It is even more interesting then, that such similar patterns evolved. Parallels must be drawn to other systems for further research, but for now we observe a common theme in software architecture: with the growth and expansion of systems and offerings, so do the abstractions on top of them.

As the challenges are unfolded in this paper, one could even wonder what the advantages are of having one large ecosystem initiative for all different product lines. After all, there are so many challenges, that it may feel like trying to trap all the different animals on earth unwillingly onto an ark. However, the participants in the initiative indicate that their expertise at this point is unparalleled in the company and that none of the product units would have the resources available to undertake the initiative at its current speed. Another trend that keeps surfacing is the "one organization, one ecosystem": if a large partner extends different

products from different product lines, NetComp wants to be aware of this, as that partner is playing a strategic role in the ecosystem.

In earlier work, we have conducted similar studies. In the work on the extensibility of mobile operating systems [1] we observed that mobile operating systems are open and extendible, but that restrictions, rules, and abstraction layers protect the inner cores of mobile operating systems. This is true for NetComp to a lesser extent: partners are expected to be 'more responsible' than mobile app developers. Also, as there are simply fewer extending partners than there are mobile app developers, NetComp does not have the resources to test and harden every interface, albeit with an exception for security aspects.

In the work on pragmatic reuse [4,5] in start-up companies, we observed eight different pragmatic extension patterns. The pragmatism is found, for instance, in the fact that these start-ups would sometimes simply hack the database of another product and read and write to it directly to extend it. None of this pragmatism is found in the extension mechanisms provided by NetComp: the extension mechanisms used most are traditional SDKs that communicate with independent "service providers", typically running on the hardware itself. As NetComp is active in the communications industry, this is not surprising: hardware deployments need to be easy to extend, loosely coupled, quick to deploy, easy to manage, and above all secure.

In the work of Kabbedijk [8] he presents a multitude of patterns that enable variability in multi-tenant environments. The pattern catalog created there is an inspiration for the current work on extensible software platforms. In the future we hope to create a similar overview to provide insight into the most common patterns used to enable and support software ecosystems.

Wnuk et al. present several case reports about Axis [12, 13], a company that is equally dependent on hardware as NetComp, but where the ecosystem initiative is currently less mature. Parallels that can be drawn are the need for standardization from partners, the need for partners to be informed regularly about platform developments, and the actual response to change requests from partners. Finally, Axis too is having difficulty opening up the platform for several different products, although this is not further specified in the case reports.

A large body of work is available on software product lines. Although seemingly this work focuses on product lines, the real contribution lies in the view on a coordinated effort in opening up products in several product lines. In that sense this work is close to product lines, but perhaps even more about organizational boundaries surrounding product lines, as for instance illustrated by Hanssen [3]. Toft also highlights the challenges of central collaboration between departments in a software product line [11]. Contrary to this work, they propose a decentralized mode of working, that forces departments to collaboratively share architecture and components.

8 Conclusion

The paper provides four contributions. First, four patterns are provided that illustrate typical scenarios for opening up a portfolio of hardware-based software products, with the goal of creating extendible software platforms. The four patterns are Simple Server Extension, Coordinating Server Extension, Federated Server Extension, and Advanced UI Extension. We provide a background on the history of the creation of the four patterns to illustrate their history and use. Secondly, the (architecture) challenges of doing so in a large company like NetComp, in a centralized fashion, are highlighted and provide interesting insights and challenges. The insights presented illustrate the advantages of centrally coordinating platformization and ecosystem efforts and the division of responsibilities in an organization that has a large product portfolio. Thirdly, several challenges of launching a platform around a hardware and software product portfolio are presented: how to open up different systems, how to document their extendible interfaces, how product and extension error messages must be propagated through the systems and organization, how crashes must be handled, and how extendible interfaces must be secured without becoming useless.

The FloodGate Department still has a large amount of work in front of it. Although the architectures are now ready for extension, the management of the ecosystem and the coordination practices of partners are still immature and varying across product departments. Secondly, the FloodGate Department would like to unify the code bases as much as possible, which is introducing an interesting architectural challenge of supporting different technologies, while keeping all in one code base and collection of software artifacts.

On the academic side, there are challenges as well. First, we plan to create a collection of platform extensibility patterns, i.e., patterns that aim to enable the creation of an ecosystem around a product, similar to our work in multi-tenant patterns [9]. Secondly, we are working on a software ecosystem management maturity matrix (SEM^3) that enables companies to evaluate their ecosystem management practices and advance them based on a set of strategic requirements, based on our earlier work [7].

References

1. Mohsen Anvaari and Slinger Jansen. Evaluating architectural openness in mobile software platforms. In *Proceedings of the Fourth European Conference on Software Architecture: Companion Volume*, pages 85–92. ACM, 2010.
2. M. Goeminne and T. Mens. A framework for analysing and visualising open source software ecosystems. *Proceedings of IWPSE-EVOL*, pages 42–47, 2010.
3. Geir Kjetil Hanssen. Opening up software product line engineering. In *Proceedings of the 2010 ICSE Workshop on Product Line Approaches in Software Engineering*, pages 1–7. ACM, 2010.
4. Slinger Jansen, Sjaak Brinkkemper, and Anthony Finkelstein. Component assembly mechanisms and relationship intimacy in a software supply network. In *15th*

International Annual EurOMA Conference, Special Interest Session on Software Supply Chains, 2008.
5. Slinger Jansen, Sjaak Brinkkemper, Ivo Hunink, and Cetin Demir. Pragmatic and opportunistic reuse in innovative start-up companies. *Software, IEEE*, 25(6):42–49, 2008.
6. Slinger Jansen, Sjaak Brinkkemper, Jurriaan Souer, and Lutzen Luinenburg. Shades of gray: Opening up a software producing organization with the open software enterprise model. *Journal of Systems and Software*, 85(7):1495–1510, 2012.
7. Slinger Jansen, Michael A Cusumano, and Sjaak Brinkkemper. *Software Ecosystems: Analyzing and Managing Business Networks in the Software Industry*. Edward Elgar Publishing, 2013.
8. Jaap Kabbedijk. *Variability in Multi-Tenant Enterprise Software*. Utrecht University, Department of Information and Computing Sciences, 2014.
9. Jaap Kabbedijk, Tomas Salfischberger, and Slinger Jansen. Comparing two architectural patterns for dynamically adapting functionality in online software products. In *PATTERNS 2013, The Fifth International Conferences on Pervasive Patterns and Applications*, pages 20–25, 2013.
10. Konstantinos Manikas and Klaus Marius Hansen. Software ecosystems–a systematic literature review. *Journal of Systems and Software*, 86(5):1294–1306, 2013.
11. Peter Toft, Derek Coleman, and Joni Ohta. A cooperative model for cross-divisional product development for a software product line. In *Software Product Lines*, pages 111–132. Springer, 2000.
12. Krzysztof Wnuk, Konstantinos Manikas, Per Runeson, Matilda Lantz, Oskar Weijden, and Hussan Munir. Evaluating the governance model of hardware-dependent software ecosystems–a case study of the axis ecosystem. In *Software Business. Towards Continuous Value Delivery*, pages 212–226. Springer, 2014.
13. Krzysztof Wnuk, Per Runeson, Matilda Lantz, and Oskar Weijden. Bridges and barriers to hardware-dependent software ecosystem participation–a case study. *Information and Software Technology*, 56(11):1493–1507, 2014.