To be Presented in CAiSE 2017 – Pre-print

Accommodating Openness Requirements in Software Platforms: A Goal-Oriented Approach

Mahsa H. Sadi[1], Eric Yu[1,2]

[1] Department of Computer Science, University of Toronto
[2] Faculty of Information, University of Toronto
{mhsadi, eric} @ cs.toronto.edu

Abstract. Open innovation is becoming an important strategy in software development. Following this strategy, software companies are increasingly opening up their platforms to third-party products. However, opening up software platforms to third-party applications raises serious concerns about critical quality requirements, such as security, performance, privacy and proprietary ownership. Adopting appropriate openness design strategies, which fulfill open-innovation objectives while maintaining quality requirements, calls for deliberate analysis of openness requirements from early on in opening up software platforms. We propose to treat openness as a distinct class of non-functional requirements, and to refine and analyze it in parallel with other design concerns using a goal-oriented approach. We extend the Non-Functional Requirements (NFR) analysis method with a new set of catalogues for specifying and refining openness requirements in software platforms. We apply our approach to revisit the design of data provision service in two real-world open software platforms and discuss the results.

Keywords: Ecosystems; Open Platforms; Software Design; Requirements

1 Introduction

Open innovation is becoming an increasingly important strategy in software development. Following this strategy, software development organizations open up their processes and software platforms to external developers in order to use external ideas, knowledge and paths to markets (as well as the internal ones) to advance their technology [1]. External developers become part of a software ecosystem offering complementary products and services for the open platforms [2 - 5].

However, opening up software platforms to third-party products is recognized as one of the most difficult transitions in software product development. While openness has the potential to create momentum for the widespread adoption and support of the platform in the market, it may lead to losing overall control of the platform [2]. Moreover, opening up platforms to third-party applications raises serious concerns about critical quality requirements, such as security, performance, proprietary ownership of the platform and its complementary applications. Yet, there is no systematic method to address these concerns in opening up platforms.

adfa, p. 1, 2011.

A successful transition to an open platform relies on adopting openness design strategies that can fulfill open innovation objectives while preserving the quality of the platform and complementary applications and services. Adopting such balanced design strategies calls for deliberate analysis of the requirements that openness introduces on the design of software platforms from early on in the transition process. Nevertheless, openness is only one design concern among many that should be accommodated in software platforms. Effective openness design strategies should optimally fulfill all of these concerns.

Example. Consider a common design scenario in opening up software platforms: *providing data service to third-party applications*. The design includes decisions about how a platform communicates data with third-party applications and how third-party applications communicate data with each other. Three design alternatives can be considered for opening up platform data to third-party applications; namely: (1) *Centralized data provision (CDP)*: Platform centrally checks every data communications between third-party applications; (2) *Semi-centralized data provision (SDP)*: A mediator (either the platform or the end-user) decides whether and under what conditions third-party applications can communicate directly; and (3) *Decentralized data provision (DDP)*: Third-party applications communicate data directly without any central control.

To choose an appropriate design strategy to open up platform data, *performance* can be a critical concern for a specific platform. Considering this, *centralized data provision* is not an appropriate design since central data control imposes additional load on the platform and increases data access time for third-party applications. *Data integrity* can be another requirement for the platform. In this regard, *centralized data provision* performs well since every data operation is performed under direct control of the platform, helping eliminate inconsistencies in simultaneous data read and write operations. Comparably, *semi-centralized data provision* also works well enough if platform is the mediator and if the platform decides to control critical data operations itself. *Decoupling third-party applications* is also important for the *open* platform since with the increase of third-party applications, it will be difficult to maintain the platform and prevent potential erroneous and malicious data communications. Considering this, *centralized data provision* is the most effective design since it minimizes the coupling of third-party applications. *Increasing adoptability of the platform* among external developers can be one main reason for opening up the platform. However, *centralized data provision* creates "*accessibility*" barriers for the platform since third-party applications should be checked and permitted by the platform to be installed and access their required run-time data. This difficulty negatively impacts the platform adoptability.

To choose the most appropriate openness design strategy, systematic methods are required that help decide between these competing and interacting requirements.

Contributions. We propose to treat openness as a distinct class of non-functional requirements, and to refine and analyze it in parallel with other concerns in designing software platforms using a goal-oriented requirements modeling language [6]. The proposed approach allows to specify and refine the business requirements behind openness,

the technical quality requirements that openness imposes on the design of software platforms, and the concerns that openness introduces on other quality requirements. The refined requirements are used as criteria for selecting optimal design alternatives. To facilitate specification and analysis of openness requirements, we propose three types of catalogues: (1) *Openness requirements specification and refinement catalogues*; (2) *Openness operationalization catalogues*; and (3) *Openness correlation catalogues*. The catalogues encode alternative paths for refining and operationalizing openness requirements, which can be customized for a particular design context. We apply our proposed approach to revisit the design of data provision service in two real-world open software platforms and discuss the results.

2 The Proposed Approach

We consider openness as a concern that should be met in the design of platforms functionalities [7]. We describe openness as a soft goal (i.e. an objective that can be fulfilled to various degrees) and refine it using contribution links. We assess the fulfillment degree of openness requirements in alternative design mechanisms using the goal-oriented forward evaluation procedure [6].

To deal with openness requirements, we customize the Non-Functional Requirements (NFR) analysis method [6]. The customized approach is comprised of seven main steps, which can be performed iteratively: **(1)** Specifying and refining openness requirements; **(2)** Specifying and refining other design concerns; **(3)** Prioritizing the requirements; **(4)** Identifying possible alternative operationalizations; **(5)** Evaluating fulfillment degree of the identified requirements in each operationalization; **(6)** Analyzing potential trade-offs; and **(7)** Selecting an appropriate design mechanism.

To facilitate specification and analysis of openness as a class of non-functional requirements, we extend NFR with a new set of catalogues, namely *openness catalogues*. Openness catalogues are of three main types: **(1)** *Openness requirements specification and refinement catalogues*; **(2)** *Openness requirements operationalization catalogues*; and **(3)** *Openness correlation catalogues*. These catalogues are used in the related steps described above, and provide extensible and customizable patterns for specifying, refining and operationalizing openness requirements in the design of software platforms.

In the following, we present instances from each type of the openness catalogues. To save space, we omit the details about the complete definition and refinement of the items in the presented catalogues, and the sources from which the items are extracted.

Openness Requirements Specification and Refinement Catalogues. These catalogues help characterize and refine the specific requirements and concerns that openness introduces on the design of software platforms. Openness requirements catalogues are of three types: **(1)** *Business-level openness requirements catalogues*. **(2)** *System-level openness requirements catalogues*; and **(3)** *General design concerns catalogues*.

System-Level Openness Requirements Catalogues. These catalogues characterize gen-

eral technical and quality requirements that should be met in the design of open platforms. Three instances of system-level openness requirements catalogues are shown in Fig 1. For example, the first catalogue (Fig 1-a) identifies that openness introduces seven types of requirements on the design of software platforms, including "*accessibility*" and "*extensibility*". From this catalogue, requirements specification paths can be generated, such as: "*To open up a platform, the platform needs to be accessible to third-party applications*", or "*To open up a platform, the platform design needs to be extensible*". The second catalogue (Fig 1-b) identifies that "*accessibility*" requirement can be refined in four ways, including "*accessibility [functionality or service]*" and "*accessibility [data]*". From this catalogue, more detailed requirements specifications can be generated, such as "*To open up a platform, platform data need to be accessible to third-party applications*". The third catalogue (Fig 1-c) identifies that "*extensibility*" requirement introduces six types of requirements on a platform design, including "*composability [Platform]*" and "*deployability [Third-party applications]*", each of which needs to be further refined into more fine-grained requirements. From this catalogues, refinement paths can be generated such as "*To make a platform design extensible, the platform needs to be composable*", and subsequently "*To make a platform composable, third-party applications should be decoupled from the platform and from each other*".

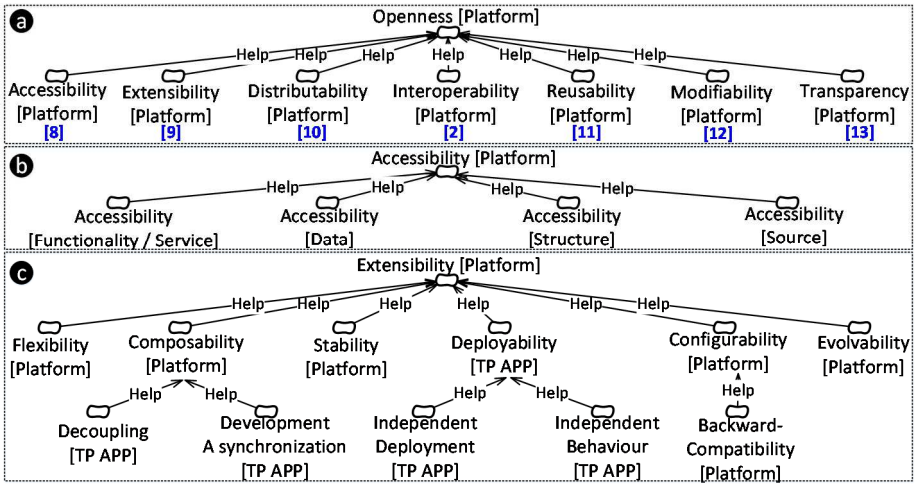

Fig. 1. Three Instances of the System-Level Openness Requirements Catalogues

To develop system-level openness requirements catalogues, two steps are performed: (1) The content of the catalogues is extracted from the Software Engineering literature discussing technical requirements in open software platforms. (2) The requirements are classified, related, and refined using two types of non-functional requirement refinement [6]: topic refinement (e.g. "*Accessibility*" catalogue) and type refinement (e.g. "*Openness*" and "*Extensibility*" catalogues). To structure the content, related elements of the goal-oriented requirements modeling language are used. "*Soft goal*" element is used to represent non-functional requirements, and "*Help*" contribution link is used to relate and refine the requirements.

Business-Level Openness Requirements Catalogues. These catalogues characterize general non-technical requirements in open software platforms and relate them to system-level openness requirements. Non-technical requirements include the business and organizational incentives that drive the need for openness as well as the social requirements that should be met in open software platforms. Each business-level openness requirements catalogue has two parts: a set of non-technical requirements and the related technical requirements. Two instances of these catalogues are depicted in Fig 2. For example, the first catalogue (Fig 2-a) identifies that "*Stickiness*" and "*Market Presence*" are two non-technical requirements in open software platforms. Stickiness refers to the degree that a software platform supports its continued use by a user instead of switching to a competitor platform [14]. "*Stickiness*" can be further related to more fine-grained business requirements such as "*Network size*". Network size refers to the number of complementary application and services that support a platform [15]. From this catalogue, specifications and refinement paths can be generated, such as "*One objective in opening up a software platform is to increase the stickiness of the platform.*", and then "*To increase the stickiness of a platform, the network size of the platform should grow.*" "*Network size*" requirement can then be related and refined to system-level openness requirements, such as "*accessibility*". One refinement is as follows: "*To increase the network size of a platform, the platform needs to be made accessible to third-party applications.*"

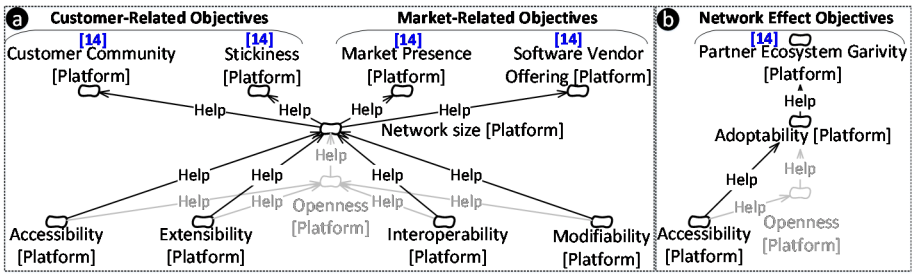

Fig. 2. Two Instances of the Business-Level Openness Requirements Catalogues

To develop business-level openness catalogues, three steps are taken: (1) The content of the catalogues is extracted from a set of Business and Software Engineering literature discussing open innovation, and the business, organizational, and social needs that it introduces on the development of software platforms. (2) The requirements are described using soft goals, and categorized, related and refined using "*help*" contribution links. Since business-level openness requirements are often described as openness business objectives, the notion of soft goal is conceptually close for describing these requirements. (3) The last row of refinement in each business-level openness catalogue is related to a set of first-row refinements in the system-level openness requirements (i.e. Fig 1-a) using "*help*" contribution links. Contribution links allow to smoothly refine and relate the business-level requirements into the system-level requirements.

General Design Concerns Catalogues. These catalogues characterize general concerns and requirements raised in opening up software platforms. These concerns may have

synergistic or conflicting relationships with openness requirements, and need to be refined and operationalized in parallel with openness requirements in designing software platforms. Two instances of this group are shown in Fig 3. For example, the first catalogue (Fig 3-a) identifies "*security*" as a general concern in opening up software platforms and also characterizes the specific types of security requirements (such as "*integrity*" and "*availability*") that are potentially impacted by openness requirements. From this catalogue, specifications can be generated, such as "*Security needs to be assured in opening up a platform*". Then this requirement can be further refined as follows: "*To assure platform security, integrity of the platform data should be preserved*."

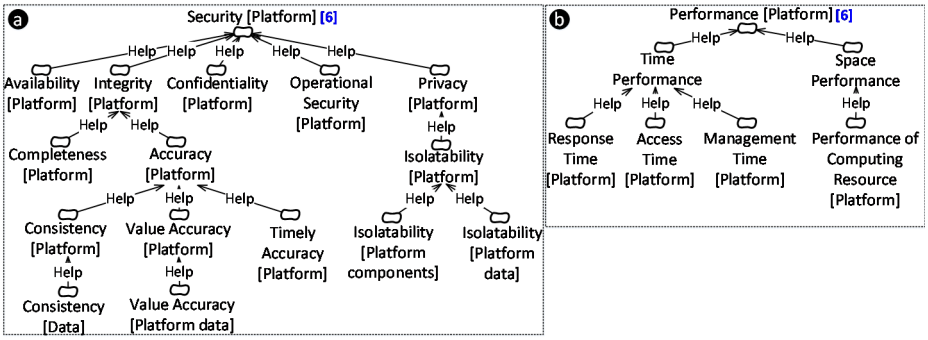

Fig. 3. Two Instances of the General Design Concerns Catalogues

The content of this group of catalogues is extracted from a set of Software Engineering and Business literature discussing problems, concerns, and requirements in opening up software platforms. The content is then structured similar to the previous catalogues. Some requirements in this group, such as security and performance overlap with existing NFR catalogues [6]. The existing catalogues have been reused and customized according to the specific context of open software platforms.

Openness Requirements Operationalization Catalogues. Operationalization catalogues identify the system functionalities that should be specifically designed to open up platforms to third-party products. They also enumerate alternative mechanisms and patterns for designing these functional requirements. Each openness operationalization catalogue has two parts: (a) *Design objectives*: the specific functionality that need to designed or implemented; and (b) *Design alternatives*: Alternative mechanisms to realize the design objective. An instance of the openness operationalization catalogues is illustrated in Table 1. The catalogue is related to the design of "*Data provision and communications service*". The catalogue elaborates on three generic alternative mechanisms for designing this functionality, namely: (1) *Centralized data provision*; (2) *Semi-centralized data provision*; (3) *Decentralized data provision*.

The content of these catalogues is extracted from a set of Software Engineering research resources discussing technical design of open software platforms.

Table 1. One Instance of the Openness Requirements Operationalization Catalogues

Design Objective: To provide data service to third-party applications

Design Mechanism 1: Centralized Data Provision (CDP) [16]
The platform controls every data and information interactions between third-party applications and the platform, and between one third-party application and another. In this design, all data is stored and exchanged through a single API in the platform. Data is accessed through the platform API either by explicit get/set operations or publish/subscribe at run-time. An API identifies available data at run-time.

Design Mechanism 2: Semi-Centralized Data Provision (SDP) [17]
Third-party applications can communicate data directly in some cases. Third-party applications declare what data they need at install-time. The requests are initially submitted to a mediator (i.e. end-user or platform). The mediator decides to allow data communications directly or not. If yes, third-party applications can communicate directly. If no, the mediator decides to control data read operations, data write operations or both.

Design Mechanism 3: Decentralized Data Provision (DDP) [10]
Third-party applications can directly exchange data and information with each other. Data interactions between two third-party applications are controlled and supervised by the third-party application that provides the requested data. Data access requests are declared at run-time and the data provider application is responsible for managing the requests and controlling the consistency of data read and write operations.

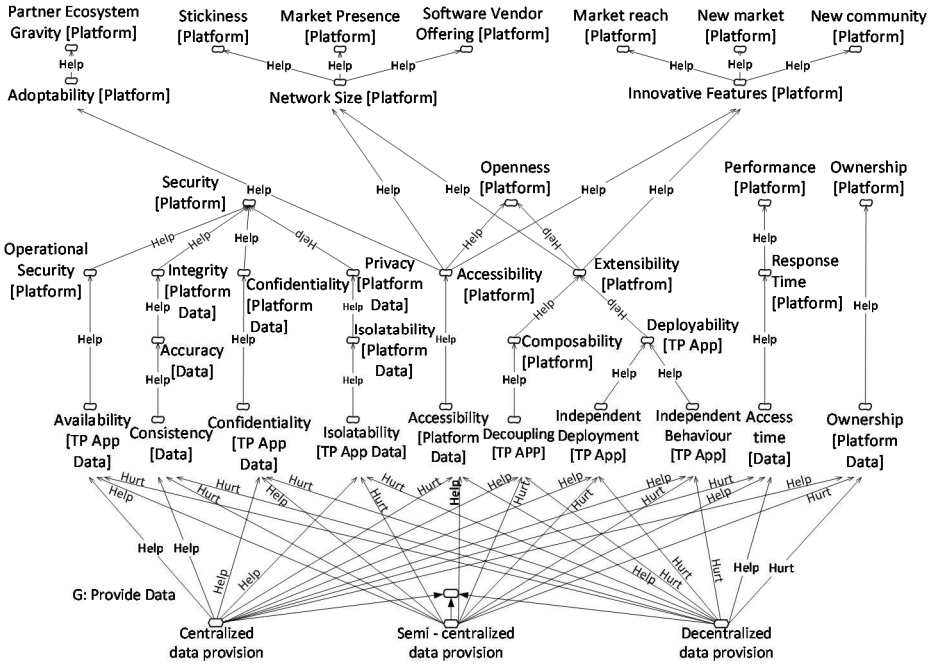

Fig. 4. One Instance of the Openness Correlation Catalogues

Openness Correlation Catalogues. Openness correlation catalogues identify the impact of each openness design alternative (in the operationalization catalogues) on the fulfillment of the related openness requirements (in the specification and refinement catalogues). An instance of a correlation catalogue is shown in Fig 4. For example, one *security* concern in designing data provision service can be data integrity ("*Integrity [platform data]*"). This requirement can be further decomposed into "*accuracy [data]*" and then "*consistency [data]*". The presented catalogue identifies that "*centralized data*

provision" design alternative meets the requirement of data consistency. In contrast, the other two alternatives of "*semi-centralized data provision*" and "*decentralized data provision*" violate this requirement. Another requirement that may be important in opening up a platform is "*accessibility [platform]*", which can be further refined into "*accessibility [data]*". The catalogue identifies that "*centralized data provision*" has a negative impact on the accessibility of platform data. In contrast, the other two alternatives have a positive impact on this requirement.

To develop correlation catalogues, two steps are performed: (1) The related requirements that are affected by each alternative operationalization are selected from the requirement refinement catalogues. (2) The positive or negative impact of the alternative on fulfilling the related alternatives is assessed. The assessment is done based on expert knowledge from the design alternatives and must be accompanied by a sound reasoning or evidence. The alternative mechanisms are assessed against the last row of refinement for each related requirement, and are described using "*help*" or "*hurt*" contribution links. A detailed example of an assessment is provided in [7].

3 Application of the Proposed Approach

We use the proposed approach to revisit the high-level architectural design of data provision service in two real-world open software platforms. Both platforms are embedded operating systems. The first platform is an operating system controlling the electronic units of a vehicle and the second one is an operating system for smartphone devices.

To apply the proposed approach on each design case, two preparatory steps have been taken: (1) The documents containing information about the design of each platform have been collected from the literature. (2) The information required for applying the proposed approach has been extracted from the collected documents. The extracted information is of two types: (a) the important design requirements for each case; i.e. the requirements that openness introduces and other general concerns that should be considered in opening up each platform; and (b) the priority of each design requirement. Where the required information was absent or not explicitly mentioned, we have augmented the information based on our own understanding from the case. Augmented information is distinguished from the extracted information using "*".

To use the catalogues, two preparatory steps need to be performed. (1) The domain requirements are matched with the requirements items available in the catalogues. If the wording of a requirement is different, the most similar requirement item in the catalogues is selected. If no similar item is found, the correct placement of the requirement is found and the related catalogue is augmented with new the content. Adding new content may also need modifying the structure of the catalogue. (2) The evaluation of design mechanisms in the correlation catalogues may also be revised in each context.

To re-design the data provision service in each case, the seven steps described in the beginning of Section 2 are performed. To refine the requirements in each design context, the related refinement paths in the catalogue presented in Fig 4 are used. Refinement is done up to the level that there is evaluation data for the refined requirement and

the three alternative designs in the correlation catalogue. The fulfillment of the requirements is then evaluated using the goal-oriented forward evaluation procedure. The evaluation results identify the degree of requirements fulfillment in each design alternative. Requirements fulfillment is described in five degrees: *Satificed (Sat), Partially Satisficed (PSat), Conflict (Conf), Partially Denied (PDen),* and *Denied (Den).* The evaluation results are used to compare alternative designs and identify the potential trade-offs that should be made between identified requirements by choosing each option. Based on the comparison results, the most appropriate design for the data provision service is selected. The selected option is then compared to the original design.

An Open Embedded Automotive Software Platform. The information related to this platform is extracted from [16]. In [16], the process of designing the platform is explained in detail. The document explains the requirements of the platform, their priorities, the decisions that were made to design the platform, and the rationale for those decisions. However, no modeling and analysis has been done in the design process. All the information required for our analysis was available in the document.

The platform is an operating system sitting on top of the electronic hardware of a vehicle to control the vehicle electronic units. The platform has to deal with safety critical functionalities and data. Thus it should be highly dependable. The platform has been opened to different types of third-party applications, such as applications developed by certified developers and applications developed by undirected developers. Third-party applications sit on top of the platform and add functionality to it. Examples of these additional functionalities include: automatic control of the speed of the vehicle or displaying the speed of the vehicle in the display. To perform such operations, third-party applications may need read or write access to data (such as speed and lateral acceleration data), controlled by the platform or other third-party applications.

The important design requirements of the platform and their priorities are described in Table 2. The related paths in the catalogue of Fig 4 that help specify and refine the requirements as well as their fulfillment in each alternative design are shown in Fig 5.

Table 2. Design Requirements for the Open Embbedded Automotive Platform

Design Requirements	**Text Description**
Openness Requirements	
Type: "**Composability**" Priority**: "High**"	"The software platform must fulfil a set of properties *to allow the* ***decoupling of applications*** and ***eliminate*** *the need for* ***development synchronization***. The architecture should allow development, integration and validation of applications independent of other applications. Non-technical users cannot do this themselves, it must be provided for by application and/or platform developers."
Type: "**Deployability**" Priority: **"High**"	"The applications must be possible *to* ***be deployed independently*** *of each other*, and the ***product behavior must not depend on the order in which applications are installed***. There must also be a deployment infrastructure in place which fulfils necessary integrity requirements."
General Design Concerns	
Type: "**Dependability**" Priority: **"High**"	"Many embedded domains have *stringent dependability requirements*; i.e. ***real-time requirements*** *for the execution of individual applications*, ***integrity requirements***, ***high availability***, and *mechanisms to* ***eliminate*** *undesired* ***feature interaction*** if several applications interact with the same actuators."

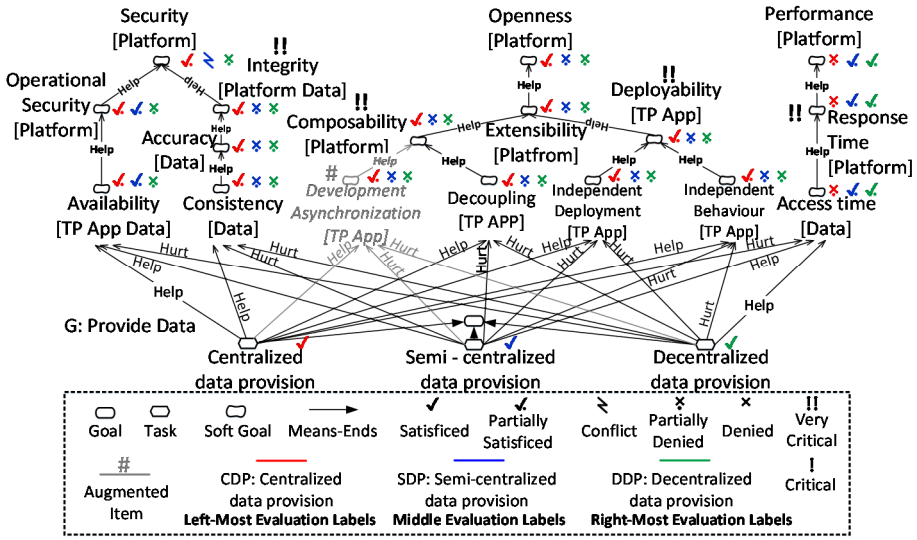

Fig. 5. Specification, Refinement, and Evaluation of the Important Design Requirements

Table 3 summarizes the fulfillment of key requirements in each design alternative. As shown, "*centralized data provision*" outperforms the other two alternatives in fulfilling all the requirements except performance. In contrast, the other two alternatives partially satisfice performance. However, "*semi-centralized data provision*" violates two openness requirements of "*composability*" and "*deployability*", and "*decentralized data provision*" underperforms in the fulfillment of all the other requirements.

Table 3. Fullfillment of the Important Requirements in Design Alternatives for Data Provision

Requirements	**Security**		**Openness**		**Performance**
	OS	**IY**	**CP**	**DP**	**RT**
Priority	**H**	**H**	**H**	**H**	**H**
CDP	PSat	PSat	PSat	PSat	PDen
SDP	PSat	PDen	PDen	PDen	PSat
DDP	PDen	PDen	PDen	PDen	PSat

OS: Operational Security [Platform]; **IY**: Integrity [Platform Data]; **CP**: Composability [TP Application]; **DP**: Deployability [TP Application]; **RT**: Response Time [Platform]; **H**: High (Very Critical)

Although "*centralized data provision*" fulfills four of the five important design requirements and achieves the highest rank among the three alternatives, it has negative impact on the performance of the platform. Centralized control over all data interactions creates a bottleneck in the platform. In case of several simultaneous data read and write requests, this design creates a queue of requests that should be checked by the platform and increases the waiting time of data operations. However, the automotive platform is in charge of safety-critical and real-time operations. Considering this, performance is not a negligible requirement.

In comparison, "*semi-centralized data provision*", though violating two openness requirements of "*composability*" and "*deployability*", alleviates the load of platform by delegating the control of *some* data interactions to the related third-party applications.

Since critical third-party applications are developed by certified developers, the platform can easily decide to control which data operations, delegating the control of less critical data interactions to the related third-party applications. Considering this, semi-centralized control does not negatively impact the integrity and security of the platform data. Accordingly, we assess the final impact of "*semi-centralized data provision*" on "*Security [Platform]*" as positive. Thus, it would be reasonable to sacrifice some degrees of "*composability*" and "*deployablity*" to achieve higher degrees of performance for real-time operations of the automotive platform.

In [16], "*centralized data provision*" alternative has been adopted to open up the automotive platform data to *all types* of third-party applications. The problem of performance (real-time data access) is alleviated via attaching different priorities to different types of third-party applications waiting in the data request queue. However, according to our analysis, for the third-party applications with less safety-critical operations "*semi-centralized data provision*" is also appropriate. Thus, using both options of centralized and semi-centralized data provision to open platform data to different types of third-party applications improves performance, while minimizing negative impacts on the openness requirements of composability and deployability.

This difference might have several reasons: (1) Performance has been sacrificed to gain higher degrees of composability and deployability, and probably security. (2) It is also possible that the track of performance requirements has been lost in designing data provision service. This is plausible due to the large number of decisions made during the design and the lack of support for requirements tracking. (3) Alternatively, due to some domain characteristics not mentioned explicitly in the design document, such as the hardware infrastructure, performance is not significantly impacted by the bottleneck of centralized data provision.

An Open Embedded Mobile Operating System Platform. Different pieces of information related to the design of the mobile platform have been collected from [2, 15, 17]. Some requirements and priorities have been added based on our understanding from the context, which are distinguished by "*".

The platform is an operating system sitting on top of the hardware device of a smartphone to control its functionalities. The platform hosts native and non-native applications. Third-party applications add a wide range of functionalities that could be of potential interest to various mobile users. Development of mobile applications is highly knowledge-intensive. Thus, mobile application development is usually open to a wide range of third-party developers. Third-party applications may need read or write access to platform data or the data generated by other third-party applications.

The requirements of the mobile platform and their priorities are described in Table 4. The related specification and refinement paths from the catalogue of Fig 4 and the fulfillment degree of the requirements in each design alternative are shown in Fig 6.

Table 5 summarizes the fulfillment of the identified requirements in each design alternative. As shown, "*centralized data provision*" underperforms in fulfilling all the *high-priority* requirements, namely "*accessibility*", "*adoptability*", "*partner ecosystem gravity*", "*innovative features*", and "*performance*". Interestingly, this alternative out-

performs in fulfilling *medium-priority* requirements, such as "*composability*", "*deployability*" and "*ownership*". In contrast, the other two design alternatives equally satisfice *high-priority* design requirements. However, "*semi-centralized data provision*" performs better in fulfilling "*privacy [data]*" requirement.

Table 4. Design Requirements for the Open Mobile Platform

Design Requirements	**Text Description**
Openness Requirements	
Type: "**Innovative Products**" Priority: **"High"**	"In many knowledge intensive domains, users and external parties play an ***important*** role in ***developing innovative products***. The mobile operating system providers benefit from emerging external innovations because ***having a high number of applications*** increases *the* ***attractiveness*** *of the platform for* ***potential customers***. Having large number of customers lead to a bigger market share in the mobile application market." [15]
Type: **"Partner Ecosystem Gravity"** Priority: "**High**"	"Third-party developers have to be considered as ***important*** players in the mobile ecosystems. While not every application can be considered innovative, ***a larger pool of developers will provide more innovative output***. The ***network size of developers and end users*** *(i.e. network effects)* will be a significant factor for application developers in selecting which mobile ecosystem to join." [15]
Type: **"Low Entry Barriers"** Priority: "**High**"	"***Entry barriers of*** *both monetary and* ***technical nature****, including entry barriers for application market, development resource needs and programing languages,* will be ***a significant factor*** for developers in selecting which mobile platform to join. Openness and entry barriers include aspects of hardware, software and market in open platforms." [15]

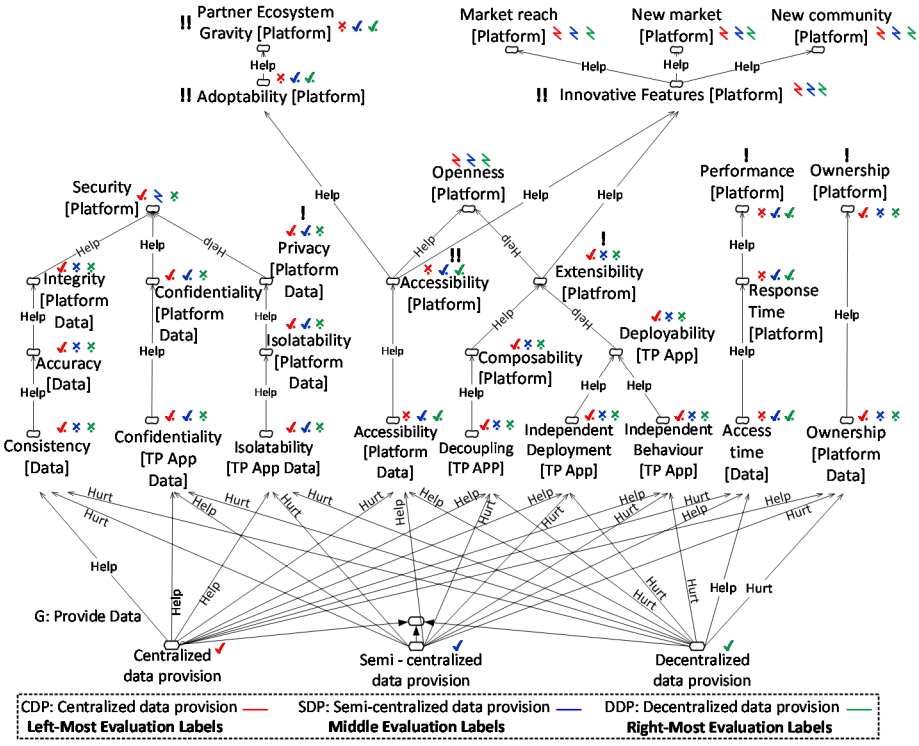

Fig. 6. Specification, Refinement, and Evaluation of the Important Design Requirements

Table 5. Fullfillment of the Important Requirements in Design Alternatives for Data Provision

Require-ments	Security	Openness: System-Level			Openness: Business-Level		Performance	Ownership
	*PV	AC	*CP	*DP	AP	ICF	*PR	* OW
Priority	*M	H	*M	*M	H	H	*H	*M
CDP	PSat	PDen	PSat	PSat	PDen	^ Conf → PDen	PDen	PSat
SDP	PSat	PSat	PDen	PDen	PSat	^ Conf → PSat	PSat	PDen
DDP	PDen	PSat	PDen	PDen	PSat	^ Conf → PSat	PSat	PDen

PV: Privacy [Platform Data]; **AC**: Accessibility [Platform]; **CP**: Composability [Plat]; **DP**: Deployability [TP App]; **AP**: Adoptability; **NS**: Network Size [Platform]; **ICF**: Innovative and Complementary Features; **PR**: Performance; **OW**: Ownership; **H**: High (Very Critical) ; **M**: Medium (Critical); ^: Conflict is resolved to partially denied or partially satisficed.

Although "*semi-centralized data provision*" satisfices all the high-priority requirements and achieves the highest score from among the three design alternatives, its implementation has negative impact on two *openness* requirements of "*composability*" and "*deployability*". It also violates "*data ownership*" requirement. Nevertheless, composability and deployability are two important technical quality attributes for an open platform. Decoupling third-party applications from each other and reducing their dependencies plays an important role in the maintainability and controllability of the platform. Specifically when the size of a platform and its complementary applications and services grow, which is usually the case for an open mobile platform. Moreover, the ownership of platform data is not a negligible requirement for a platform owner.

However, "*accessibility*" and the impact it has on the "*adoptability*" and '*innovative features*" is strategically critical to the success of a mobile platform in the market, specifically in a fierce competition with other platforms. Thus, it would be reasonable to sacrifice some degrees of the *system-level openness requirements* to gain more support from innovative and complementary applications (the *business-level openness requirements*), specifically in a knowledge-intensive domain as mobile applications.

The result of our analysis indicates that "*semi-centralized data provision*" is the best option from among the three alternatives to open up mobile platform data to third-party applications. This result is consistent with real-world implementation of open mobile platforms such as Android [17]. In Android, third-party applications declare the data they require from the platform and other third-party applications at install time. The access is permitted by the end user (i.e. end user is the mediator).

4 Discussion

Our goal was to provide a method to determine appropriate design strategies for opening up software platforms to third-party applications. We proposed to treat openness as a non-functional requirement and to use a goal-oriented approach to refine and analyze openness in parallel with other requirements. The refined requirements are used to select optimal design options. We have developed a set of catalogues that facilitate reasoning about openness requirements.

We applied the proposed approach to revisit the design of data provision service in two real-world open software platforms: an automotive platform and a mobile platform. Our goal was to determine the most appropriate openness design strategy for each case.

In the first case, our analysis identifies that a combination of centralized and semi-centralized data provision can be used to open up the platform data to different types of third-party applications. This result is slightly different from the original design of the platform, which is only centralized data provision. We aim to discuss the results of our analysis with the original designers in a future interview. In the second case, our results are consistent with the design of open mobile platforms, such as Android. The analysis justifies the accessibility of mobile platforms to external applications. Moreover, the analysis shows that system-level openness requirements can be sacrificed to fulfill business-level openness requirements. Finally, in both cases there is no design option that can fulfill all the identified requirements. In each case, trade-offs should be made. Therefore, it is crucial to detect and analyze the trade-off points.

The proposed approach allows to reason about openness as a distinct requirement. This approach complements recent research efforts on the development of open software platforms, which either focus on the technical design of the platforms, including API development (e.g. [9, 13, 18])) or on the business aspect (e.g. [2, 19]).

This paper presents only one instance of a complete openness correlation catalogue that we have developed. The complete definition and refinement of the requirements and operationalizations in the presented catalogues in addition to other catalogues will be published in a future work.

To improve the applicability of the proposed approach, three issues need to be further addressed: (1) The catalogues and the models developed for a specific domain become complex too quickly. To handle this complexity, automated support is required. (2) The evaluation procedure to select optimal design strategies needs to be made efficient via omitting exhaustive evaluations of all the options. (3) The evaluation procedure should allow to assess the final impact of selecting multiple operationalizations on the fulfillment of the identified requirements in a design process.

Further research is needed to extend and validate the content of the proposed catalogues and to compare the proposed approach with peer requirements analysis methods for software systems, such as Architecture Trade-Off Analysis Method (ATAM) [20].

5 Conclusion

We proposed a goal-oriented approach for analyzing openness requirement in software platforms. The proposed approach is supported by a set of catalogues that facilitate specification and refinement of openness requirements. We presented instances of these catalogues herein. Specification and analysis of requirements is essential for adopting effective openness design strategies that are "*open enough*" to benefit from the contributions of third-party applications and at the same time possess the quality of "*closed*" systems. Adopting such balanced strategies is crucial for the viability and sustainability of open platforms. Further research is needed to assess the effectiveness of the proposed approach and catalogues in case studies of open platform projects.

References

1. Chesbrough, H. W. (2006). Open innovation: The new imperative for creating and profiting from technology. Harvard Business Press.
2. Boudreau, K. (2010). Open platform strategies and innovation: Granting access vs. devolving control. Management Science, 56(10), 1849-1872.
3. West, J. (2003). How open is open enough?: Melding proprietary and open source platform strategies. Research policy, 32(7), 1259-1285.
4. Jansen, S., Brinkkemper, S., Souer, J., & Luinenburg, L. (2012). Shades of Gray: Opening up a software producing organization with the open software enterprise model. Journal of Systems and Software, 85(7), 1495-1510.
5. Sadi, M. H., & Yu, E. (2014). Analyzing the evolution of software development: from creative chaos to software ecosystems. In IEEE Eighth International Conference on Research Challenges in Information Science (RCIS), 2014, (1-11).
6. Chung, L., Nixon, B. A., Yu, E., & Mylopoulos, J. (2012). Non-functional requirements in software engineering (Vol. 5). Springer Science & Business Media.
7. Sadi, M. H., & Yu, E. (2017). Modeling and Analyzing Openness Trade-Offs in Software Platforms: A Goal-Oriented Approach. In International Working Conference on Requirements Engineering: Foundation for Software Quality (pp. 33-49).
8. Anvaari, M., & Jansen, S. (2010). Evaluating architectural openness in mobile software platforms. In Proceedings of the Fourth European Conference on Software Architecture: Companion Volume (85-92).
9. Bosch, J., & Bosch-Sijtsema, P. (2010). From integration to composition: On the impact of software product lines, global development and ecosystems. Journal of Systems and Software, 83(1), 67-76.
10. Scacchi, W. (2007). Free/open source software development: Recent research results and methods. Advances in Computers, 69, 243-295.
11. Bosch, J. (2010). Architecture challenges for software ecosystems. In Proceedings of the Fourth European Conference on Software Architecture: Companion Volume (pp. 93-95).
12. Baresi, L., Di Nitto, E., & Ghezzi, C. (2006). Toward open-world software: Issue and challenges. Computer, 39(10), 36-43.
13. Cataldo, M., & Herbsleb, J. D. (2010). Architecting in software ecosystems: interface translucence as an enabler for scalable collaboration. In Proceedings of the Fourth European Conference on Software Architecture: Companion Volume (65-72).
14. Popp, K. M. (2010). Goals of Software Vendors for Partner Ecosystems–A Practitioner´ s View. In Software Business (181-186).
15. Koch, S., & Kerschbaum, M. (2014). Joining a smartphone ecosystem: Application developers' motivations and decision criteria. Information and Software Technology, 56(11).
16. Eklund, U., & Bosch, J. (2014). Architecture for embedded open software ecosystems. Journal of Systems and Software, 92, 128-142.
17. Shabtai, A., Fledel, Y., Kanonov, U., Elovici, Y., Dolev, S., & Glezer, C. (2010). Google android: A comprehensive security assessment. IEEE Security & Privacy, (2), 35-44.
18. Christensen, H. B., Hansen, K. M., Kyng, M., & Manikas, K. (2014). Analysis and design of software ecosystem architectures–Towards the 4S telemedicine ecosystem. Information and Software Technology, 56(11), 1476-1492.
19. Ghazawneh, A., & Henfridsson, O. (2013). Balancing platform control and external contribution in third-party development: the boundary resources model. Information Systems Journal, 23(2), 173-192.
20. Kazman, R., Klein, M., Barbacci, M., Longstaff, T., Lipson, H., & Carriere, J. (1998). The architecture tradeoff analysis method. In Fourth IEEE International Conference on Engineering of Complex Computer Systems, 1998. ICECCS'98. Proceedings. (68-78).