REFSQ 2017 (pp 1 - 17) - Pre-Print

Modeling and Analyzing Openness Trade-Offs 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. [Context and motivation] 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 for extension and completion. **[Question / problem]** Opening up software platforms to third-party applications often involves difficult trade-offs between openness requirements and critical design concerns such as security, performance, privacy, and proprietary ownership. Deliberate assessment of these trade-offs is crucial to the ultimate quality and viability of an open software platform. **[Principal ideas / results]** We propose to treat openness as a distinct class of non-functional requirements, and to model and analyze openness requirements and related trade-offs using a goal-oriented approach. The proposed approach allows to refine and analyze openness requirements in parallel with other competing concerns in designing software platforms. The refined requirements are used as criteria for selecting appropriate design options. We demonstrate our approach using an example of designing an open embedded software platform for the automotive domain reported in the literature. **[Contributions]** The proposed approach allows to balance the fulfillment of interacting requirements in opening up platforms to third-party products, and to determine "good-enough" and "open-enough" platform design strategies.

Keywords: Requirements Engineering, Software Design, Decision Making, Open Software Platforms, Software Ecosystems, Open Innovation

1 Introduction

"How open is open enough?"– Joel West [1]

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 and third parties in order to use external ideas and paths to market (as well as the internal ones) to advance their technology [2]. External developers become part of a software ecosystem offering complementary applications and services for the open platforms [3, 4, 5]. Google Android, Apple iOS, and Windows Mobile are a few examples of open software platforms.

REFSQ, p. 1- 17, 2017.

However, developing open software platforms that are technically sound, socially sustainable and economically viable is a challenging problem in software development. First, critical decisions need to be taken in opening up software platforms to third-party products that raise serious concerns about proprietary ownership and confidentiality of a platform and its complementary applications. Examples of such decisions include: deciding between the core features and functionalities that build the core competencies of a platform, and those that can be opened up to third-party developers [6,7]; or identifying the appropriate *degree of openness* for engaging different third-party developers, some of whom are also competitors in the market place [7, 8]. Second, openness introduces a specific set of requirements on the design of a platform which are in competition with crucial requirements such as security, performance, maintainability, and controllability. For example, opening up a platform may urge the need for transparency and visibility of platform functionalities and data to third-party applications. These requirements pose serious risks to the security of the platform. Another example is that distributing platform features among applications from different parties threatens the controllability and maintainability of the overall platform [4], [9].

A successful example of an open software platform is Google Android. Lowering the entry barriers and providing easy access to extend the platform has significantly increased the adoption of Google Android among mobile manufacturers and application developers, introducing this operating system as a leader mobile software platform in the market [10]. However, Google Android and its complementary applications suffer from performance and security issues [11].

It is crucial to clearly understand and analyze the requirements that openness introduces on the design and evolution of a platform, and to carefully assess the related trade-offs before opening up a platform to third-party applications. To model and reason about openness requirements and related trade-offs, we propose a goal-oriented approach. The proposed approach reduces the problem of designing open software platforms to a decision making problem, treats openness requirements as a distinct class of non-functional requirements, refines them in parallel with other important design concerns, and uses the refined requirements as selection criteria to determine an appropriate design strategy from among alternative options for opening up a platform.

In Section 2, we identify some requirements and concerns that have been raised in the design of open software platforms. We briefly review the main steps of the Non-Functional Requirement (NFR) engineering approach in Section 3. We illustrate how to model and analyze openness requirements using NFR, in Section 4. We review the related research in Section 5 and conclude the paper in Section 6.

2 Requirements and Concerns in Open Software Platforms

An open software platform is a platform on top of which third-party applications can be built [3, 4, 5], [12]. Unlike in Free and Open-Source Software (FOSS) [13, 14], the source code of an open software platform is usually not made available to third parties. Instead, there are extension mechanisms, such as Application Programming Interfaces (APIs) or development environments that allow sufficient access to the services and

functionalities of the open platform. Moreover, in open software platforms, major players develop purposive strategies attempting to gain competitive advantage [3], [15].

The requirements that need to be considered in opening up software platforms to third-party applications can be categorized into two main groups: (1) *Openness design requirements*: The specific concerns and quality requirements that openness introduces on the design; and (2) *General concerns in designing software platforms*: The requirements that are possibly violated or at risk when opening up platforms to external applications. Often, these requirements cannot be fully fulfilled simultaneously in the design of a platform. A designer may need to compromise between these two types of requirements. In the following, we identify several of these requirements and concerns.

Table 1. Business-Level Openness Requirements

Market-Related Objectives – Market Reach, Market Presence, New Markets, Standardized Market, Adoptability, and Time to Market. A main reason for opening up software platforms is to expand market reach, open up new markets and communities for a platform, increase the adoption of a platform among various users and developers communities, increase the number and variety of innovative and complementary features, and reduce time to market of new and innovative features [15, 16, 17].
Customer-Related Objectives – Attracting New Customers and Developing New Customer Communities, Stickiness of the Platform, and Customer Retention. Growing the network size of complementary applications hardens switching to a different platform, thus increases the stickiness of a platform. Moreover, growing the variety of platform offerings increases attractiveness of the platform for new and potential users, and increases value of the core product to the existing users [15, 16].
Product-Related Objectives – Co-Innovation and Open-Innovation, and Variety of Software Vendor's Offerings. Innovative features play an important role in the success of a platform, specifically in knowledge intensive domains. Via growing the network size of developers, the platform owners can benefit from emerging external innovations [15].
Financial-Related Objectives – Revenue Stream, Sharing the Costs of Innovation, and Decreasing Total Costs of Ownership. Collaborating with partners in ecosystems shares the cost of innovation and decreases the total cost of ownership for commodity and innovative functionality [15, 16].
Network-Effect-Related Objectives – Customer and Partner Ecosystem Gravity, and Community Building. Third-party developers play an important role in the success of an open platform through their contributions and innovations. A larger pool of developers will provide more innovative output. Thus, platform developers aim to attract and engage a large number of developers to contribute and develop applications to their platforms. Factors, such as the degree of openness, low entry barriers of both monetary and technical nature, and the network size of a platform influence the choice of external developers to join a platform [10], [15, 16].

2.1 Openness Design Requirements

Openness introduces two types of requirements on the design of software platforms: (1) *Business-level openness requirements*: These requirements are the main motivations for opening up a software platform to third-party applications. Business-level openness requirements are non-technical, related to the social, business, and organizational environment of a software platform, and may indirectly influence the design of an open platform. These requirements often compete or interact with technical quality requirements in the design of open software platforms. Thus, specific attention should be given to this group in choosing effective design strategies for opening up a software platform. These requirements can be categorized into five main groups described in Table 1. (2) *System-level openness requirements*: These requirements are technical, related to the quality of software design, and directly influence the design decisions. The technical

quality requirements that need to be considered in opening up software platforms can be classified into seven groups described in Table 2.

Table 2. System-Level Openness Requirements

Accessibility. An open software platform needs to be accessible to third-party applications and have access to the features and services of third-party applications. The ease of access to and from a software platform is an important quality requirement for opening up a platform. The accessibility of a platform can be categorized into four levels: (1) accessibility of functionalities and services; (2) accessibility of data; (3) accessibility of platform structure (i.e., access to features and components); (4) accessibility of source code. [18]
Extensibility – Composability, Deployability, Stability, Configurability, and Evolvability. An open software platform needs to be extended and complemented by other software applications and components over time. Extensibility quality attribute identifies how easy a new application or feature can be added to a platform. Various quality criteria contribute to the extensibility of a platform, including: **(1)** *Composability*: Open and seamless integration of external modules is an important requirement for a platform. Factors such as *decoupling* third-party applications from each other, *eliminating the need for development synchronization*, and *independent development, integration, and validation* of third-party applications contribute to the composability of an open platform. [12], [19]. Carefully decoupled components with well-defined interfaces enable third-party developers to modify their applications without disrupting the overall correctness. Platform interfaces should decouple the platform organization from the third-party applications. Achieving this objective, allows the platform owner to release new version of the platform or new components without disabling the externally developed applications operating on top of the platform [12], [20]. **(2)** *Deployability*: Third-party applications must be possible to be deployed independently of each other, and the platform behavior must not depend on the order in which applications are deployed [19]. **(3)** *Stability*: Open software platforms and their APIs need to be sufficiently stable over time to provide a stable infrastructure for third-party applications [19]. *Backwards compatibility* is an important quality attribute contributing to the stability of the platform. **(4)** *Configurability*: Open software platforms must support variability in configuring the platform and third-party applications to enable customized products be developed [19]. **(5)** *Evolvability*. In open software platforms, new functionality are continuously added and the size of the platforms continuously grow. To deal with the growth, it is required to proactively refactor platform architecture and standardize platform interfaces. [20].
Decentralizability and Distributability. The functionalities of an open software platform need to be distributed among several applications, and platform components need to operate in a decentralized environment. Thus, the ease to operate in a decentralized environment is an important quality requirement for an open software platform. [13].
Interoperability. An open software platform requires to easily cooperate and interact with third-party applications. Mechanisms are required to coordinate and facilitate the interactions between the platform and third-party applications and to resolve conflicts that arise in coordination [4], [13], [20].
Reusability. An open software platform and its components need to be used and re-used in the development of other software features and applications. The ease to do so is an important design quality in an open platform.
Modifiability. To use the platform in the development of other applications and software features, the platform or some parts of its functionalities or structures may need to be modified and customzied. Thus, the platform should provide mechanisms that enables easy modification of some features.
Visibility or Transparency. To be complemented and extended by third-party applications, the platform structure, functionalities, and behavior need to be visible and transparent to external applications to various degrees [21].

2.2 General Concerns in Designing Software Platforms

Aside from openness requirements, there are other considerations applicable to the design of software platforms that are potentially impacted by openness requirements. Several instances of these requirements are identified in Table 3.

Table 3. General Design Concerns in Open Software Platforms

Security – Operational Security, Integrity, Confidentiality, and Privacy. The end-users use a composition of the core of platform and various external applications developed on top of it. Security concerns arise as possible defective or malicious code in external applications may disable the overall system [20], [22, 23]. Mechanisms are required: (1) to guarantee the integrity of platform services and data in the presence of access by third-party applications [19]; (2) to preserve the confidentiality and privacy of the end-users' information and platform data when opening up a platform to third-party developers. [7], [20]; and (3) to ensure safe and correct operation of features and services developed by multiple parties.
Controllability, Maintainability, and Centralizability. The development and maintenance of an open platform and its complementary applications is shared among various parties. In this setting, mechanisms are required to manage software enhancements, extensions, and architectural revisions in decentralized projects. Moreover, rules are required to govern and control the applications network. [4], [9], [13].
Reliability, Trust and Accountability. In open software platforms, parties providing and consuming a software service are easily exposed to cheaters. Therefore, mechanisms are required to guarantee trustworthiness and accountability of third-party services and functionalities [13], [23].
Proprietary Ownership. The ownership and intellectual property rights of the applications, components and data produced by external developers is a critical concern in open software platforms. Mechanisms are required to ensure responsibility and commitment to updating and supporting third-party modules. Moreover, the alignment of component licenses need to be checked in the usage and composition of open software components and modules at build time and deployment [7], [20], [23].

3 Non-Functional Requirements Analysis Method

To deal with interacting and competing requirements, we use the Non-Functional Requirements (NFR) engineering approach [24]. NFR reduces the problem of designing a software system into a decision making problem and a search for satisfactory design options. To identify an appropriate design option, four main steps are performed in NFR: (1) *Characterizing and Prioritizing Design Requirements*: In this step, the requirements and constraints important to a specific design context are identified and characterized in terms of a set of non-functional requirements; i.e., a set of technical and non-technical quality objectives that a design should meet. For this purpose, two main activities are performed: The design requirements are first identified and refined, then they are prioritized based on their importance in the specific design context. (2) *Identifying Alternative Design Options*: The second step is to identify the design objective (i.e., the specific functionality to be designed or implemented) and to explore alternative design options for achieving the specified objective. (3) *Evaluating Design Alternatives against Design Requirements*: To choose an appropriate design option, the design alternatives are evaluated based on the identified design requirements. (4) *Selecting Satisficing Design Options*. The final step is to select the most appropriate design options from among the available alternatives. To select an appropriate design option, it is required to formally describe and prioritize the identified design requirements, and to assess their fulfillment in each design option. For this purpose, NFR provides a goal-oriented modeling and analysis procedure [25]. The modeling procedure has two main steps of *describing a design decision* and *modeling the design decision* (explained in Table 4). To analyze the fulfilment of the identified design requirements in each design option, NFR provides a semi-automatic goal-oriented forward evaluation procedure. Using this approach, all the design alternatives are evaluated against the design requirements, and then the most satisfactory design option is selected. To analyze the

impact of each design alternative on the design requirements, a labeling system is used, which is explained in Table 4.

Table 4. Modeling and Analyzing Design Decisions Using i* Goal-Oriented Language

Modeling. Each design decision is described using three elements: **(a)** a *design objective*, **(b)** at least two atomic alternative *design options* (which are non-overlapping and exclusive), and **(c)** at least one *design requirement* which discriminates between the alternative design options. A design decision is modeled as follows: **(1)** The design objective is represented using "*Goal*" element. **(2)** Alternative design options are modeled using "*Task*" element. **(3)** The relationship between a design goal and design options are modeled using "*Means-Ends*" link. **(4)** If design requirements are atomic they are modeled using "*Soft Goal*" element. If design requirements are non-atomic, they need to be refined and modeled using "*Soft Goal Interdependency Graphs (SIG)*". In SIG graphs, refinement of a design requirement is modeled using "*Help*" contribution link. **(5)** Evaluation of design options against design requirements are modeled using "*Help*" and "*Hurt*" contribution links. **(6)** Priorities of design requirements are modeled using three types of priorities: *non-critical*, *critical*, and *very critical*. **Analysis. (1)** *Label Assignment*. The selection of a design option is described using a label assigned to the "*Task*" element representing the chosen option. **(2)** *Label Propagation*. The impact of an alternative on immediate design requirements are described using a predefined set of label propagation rules, which can be redefined in a specific evaluation. **(3)** *Label Resolution*. After each step of performing label propagation, a "*Soft Goal*" might receive a set of labels from the underneath "*Soft Goal*" or "*Task*" elements. A set of predefined label resolution rules determine the final label of the "*Soft Goal*" element, representing a design requirement. Label resolution step requires human input and is semi-automatic.

4 Example Modeling and Analysis

To demonstrate our proposed approach, we use the case study of designing the AUTOSAR platform for embedded automotive software reported in [19]. We have chosen this case study for two reasons. First, AUTOSAR is a real-world industrial open platform and its design process is explained in detail in [19]. The design process is explained in terms of the design requirements, the decisions taken in the design, and the final strategies adopted to design the platform. Thus, we add no hypothetical data or assumption to the requirements of this case. We only extract the explanations about platform functionalities (Section 4.1) and the related design requirements (Table 5), and then apply our proposed approach. Second, the designers have adopted a structured approach in identifying requirements and decisions, without using modeling for analysis. Using this study, we can show how the proposed modeling and analysis approach might be effective for designing a real-world industrial-scale open platform.

4.1 System Description: An Open Embedded Automotive Platform

The AUTOSAR platform manages the electronic units of a vehicle. Some electronic units control vehicle steering sensors and actuators, and some are responsible for accessory functions such as infotainment modules. Different electronic units communicate via data buses. The platform shares the control of the electronic units with third-party applications. The platform controls most of critical electronic units in charge of basic operations of a vehicle (such as the engine, brakes and forward sensing modules). Less critical functions (such as displaying vehicle speed in the cluster display, locking

the doors or infotainment modules) can be controlled either by certified third-party applications or by third-party applications developed by undirected developers. The core of the platform, including the set of software modules providing necessary services to use a vehicle, will be deployed on a car before delivery to the end-user. Less critical functions and accessories can be updated or deployed after delivery on an ongoing basis. The platform should be designed in a way that can accommodate these kinds of extensions and completion.

In the following, we focus on the scenario of designing data provision service to third-party applications from the platform. Third-party applications may require to access to and operate on platform data or data from other third-party applications. Examples of these data include: the speed and lateral acceleration of the vehicle or the speed of nearby cars. These data are aggregated from sensors in the wheels. Third-party applications may require access to platform data such as speed data to simply display it in the speed display or to automatically adjust the speed of a vehicle with respect to nearby cars. It is possible that several third-party applications require access to the same data at the same time. For example, auto-cruise system and direct brake control system may want to adjust the speed at the same time. Therefore, generic mechanisms should be designed in the platform to provide data service to present and future third-party applications. In the following, we demonstrate how to determine an optimal design strategy for opening up AUTOSAR platform data to third-party applications, treating openness requirements as a class of non-functional requirements.

4.2 Modeling and Analysis

Determining the most appropriate design strategy for providing data service to third-party applications consists of four main steps: (1) characterizing and prioritizing design requirements, including *domain-specific requirements* (general design concerns) and the *requirements that openness introduces on the design*; (2) identifying alternative design options for opening up platform data to third-party applications; (3) evaluating the design options against the identified design requirements; and (4) selecting an appropriate design option. To select an appropriate design option, the identified design options are modeled, prioritized and analyzed using NFR goal-oriented modeling and analysis as described in Table 4.

Characterizing domain-specific design requirements. The embedded platform is in charge of controlling automotive electronic units, many of which have safety-critical functionalities such as automatic control of the vehicle speed and brakes. Therefore, the design has to meet stringent *dependability requirements* with high priority. The dependability requirements are of two types: (1) *Performance requirements*: Platform and individual third-party applications must operate in real-time. Therefore, *the response-time* of the platform must be minimized and undesirable interactions between applications should be eliminated. (2) *Security requirements:* including *integrity* and *availability* of services to assure operational security of the platform. Relevant aspects of these requirements need to be fulfilled in the design of data provision service. The details of domain design requirements and their priorities are provided in Table 5.

Characterizing openness design requirements. The platform shares control of the electronic units with third-party applications. Opening up the platform imposes high-priority *extensibility requirements* on the platform including: (1) *Composability*: The automotive platform needs to accommodate and interact with third-party applications. Therefore, the open platform should enable open and seamless integration of external modules. (2) *Deployability*: Third-party applications must be deployed independently of each other. Openness requirements also need to be refined and considered in the design of data provision service. The details of openness design requirements and their priorities are described in Table 5.

Identifying alternative design options. Three alternative design options can be considered to provide data service to third-party developers, including: (1) *centralized data provision*, (2) *semi-centralized data provision* and (3) *decentralized data provision*. In centralized data provision, all data exchange operations between the platform and third-party applications are controlled by the platform. Third-party applications cannot communicate directly with each other. In semi-centralized data provision, third-party applications are allowed to exchange data directly. However, a supervisor (either the platform or the end user) mediate the data interactions between third-party applications. In decentralized data provision, the third-party applications can independently exchange data with each other. Further details about the design options is provided in Table 6.

Table 5. Design Requirements Important for Providing Data Service to Third-party Apps

Design Requirement	**Description**
Domain Design Requirements : Security | Priority: High	
Integrity [Platform Data]	Many of platform data are safety critical (such as speed data). The platform must implement necessary mechanisms to ensure the integrity, accuracy and consistency of all the operations performed on safety-critical data.
Availability [Platform] and [Third-Party Applications]	The platform services should correctly operate at any time. Mechanism are required to guarantee high-availability and fast failure recovery of platform operations.
Domain Design Requirements : Performance | Priority: High	
*** Response Time [Platform]	Access-Time [Data]: Platform and third-party applications should operate in real-time. Thus, response-time of the platform and access-time of third-party applications to the required data should be minimized and platform should respond to the data access requests in real-time.
Openness Design Requirements: Composability [Platform] | Priority: High	
Decoupling	(1) [Third-Party Applications]: Third-party applications must be decoupled from each other and work independently. (2) [Platform]: Platform and third-party applications development and evolution should be decoupled.
Development Asynchronization [Third-Party Applications]	The design must eliminate the need for development synchronization and enable third-party applications to be developed, integrated and validated independently of other applications. Since non-technical users cannot integrate and validate the composition themselves, this requirement must be supported by the platform.
Openness Design Requirements: Deployability [Platform] | Priority: High	
Independent Deployment [Third-Party Applications]	Third-party applications must be deployed independently from each other.
Independent Behaviour [Third-Party Applications]	Platform behaviour must not depend on the order in which the applications are installed and deployed.

*** Response time and access time design requirements were not explicitly mentioned in [19]. We inferred these requirements from the real-time operations that the automotive platform must perform.

Table 6. Providing Data Service To Third-Party Apps: Three Alternative Design Options

Design Objective: To provide data service to third-party applications
Design Option 1: Centralized Data Provision The platform controls all data interactions between third-party applications and the platform, and between one third-party application and another. In this design alternative, all data is stored and exchanged through the platform, but most data is isolated to a single application through a single API. Data and provided services are accessed through the platform API by either and explicit get/set and/or subscribe, both at run-time. There is also and API to determine the available data set at runtime.
***** Design Option 2: Semi-Centralized Data Provision** Third-party applications can communicate directly in some cases. Any data access request is initially submitted to a mediator (end-user or the platform). After checking and allowing the request, third-party applications can communicate directly. For this purpose, applications declare what data and information they need at install-time. The platform decides to control data write operations, data read operations or both.
***** Design Option 3: Decentralized Data Provision** Third-party applications can directly exchange data and information with each other. Data and information exchange between one third-party application and another is controlled and supervised by the third-party application that provides the requested data. In this design, data access requests are declared at run-time and third-party applications are responsible for managing the data access requests from other third-party applications. Data provider application is in charge of controlling the consistency of data write operations.
******* Design options 2 and 3 did not exist in the original study. They are generated as alternative options for the design strategy that the original designers have adopted (as a part of the proposed analysis approach).

Evaluating design options against the design requirements. The fulfilment of each domain-specific and openness design requirement (Table 5) should be evaluated in each data provision design option. The details of this evaluation is presented in Table 7. In Table 7, the contribution of design options to the refined design requirements are represented by a (+) or (-) label. A (+) indicates that a design option has a positive impact on the fulfilment of a design requirement. A (-) indicates that the design option violates or is negatively co-related with a design requirement. Each evaluation is accompanied with reasons explaining why a positive or negative label has been assigned.

Selecting an appropriate design option. As shown in Table 7, each design option has received a set of (+) and (-) labels in the evaluation against the requirements. This means that in choosing each design option, trade-offs should be made between a set of competing requirements. For example, choosing centralized data provision helps achieve "*Decoupling[TP App]"* (an important design requirement for opening up the platform) but as a result "*Access Time[Data]"* is violated. However, access time is also an important requirement for real-time operations of the automotive platform. To take a final decision between the design options, all the trade-offs between the requirements need to be carefully examined. For this purpose, the identified requirements, their priorities and their trade-offs need to be formally modeled and analyzed. We have modeled the information presented in Table 5, 6, and 7, and analyzed the impact of each design option on the design requirements using goal-oriented modeling and analysis (explained in Table 4). The results are presented in Fig 1.

In Fig 1, the design requirements and their refinements are shown in the upper part, the design options and their evaluation against the immediate refined requirements are shown at the bottom, and the degree of fulfillment of each design requirement in each design option is shown by the colored labels besides the requirements. Moreover, trade-offs points can be recognized in two ways: (1) directly from the "*conflict*" label beside a design requirement. For example, "*Composability [Platform]*" has received a "*conflict*" label from two options. The reason for the conflict can be traced back to the

fulfillment of its refinement; i.e., "*Decoupling[Platform]*" and "*Decoupling[TP App]*". "*Centralized data provision*" design option helps decouple third-party applications from each other, but in return, it increases the coupling between the platform and third-party applications. Similarly, "*Decentralized data provision*" design option has this conflict in reverse order; (2) indirectly by comparing the labels of the same color between different design requirements. For example, "*Extensibility [Platform]*" has received a "*partially satisficed*" red label, and "*Performance[Platform]*" has received a "*partially denied*" red label. This difference indicates that by choosing "*Centralized data provision*" option, a designer has to sacrifice some degree of performance in order to gain some degree of extensibility for openness. The fulfillment of the design requirements in each design option is summarized and compared in Fig 2. In Fig 2, the nested pentagons represent different degrees of design requirements satisfaction (from denied to satisficed). The nodes of the pentagon depict the main design requirements for the automotive platform. As Fig 2 shows, "*Centralized data provision*" outperforms other options except in fulfilling "*Performance [Platform]*".

4.3 Discussion

Our modeling and analysis (Fig 1) detects two important trade-off points between the requirements: One trade-off is between two openness requirements of "*Decoupling[Platform]*" and "*Decoupling[TP App]*". This means that in choosing each of the design options, a designer has to comprise between independence of the platform from third-party applications and independence of third-party applications from each other. Both of the requirements are important for the platform and their dissatisfaction may have irreversible impacts. Another trade-off is between the openness requirement of "*Extensibility[Platform]*" and "*Performance[Platform]*". Performance requirement is of particular importance for the real-time operations of the automotive platform (e.g. the real-time adjustment of speed or the real-time activation of brakes). On the other hand, extensibility is also crucial to accommodate third-party applications. The impact of this trade-off must be carefully assessed before making any final design decision.

In [19], the original designers have implemented "*centralized data provision*" strategy for designing the automotive platform, without acknowledging the above trade-offs. This decision may have two reasons: (1) The designers use an informal and descriptive method for designing the open platform; i.e., they identify the requirements and then explain a set of generic design patterns that fulfill the requirements. Since the design process is comprised of numerous decisions (typical in an industrial-scale design project), it is possible that the designers have lost the track of some requirements in the design. (2) It is also possible that the designers have noticed the above trade-offs, and have decided to sacrifice some degrees of performance to gain higher degrees of extensibility (i.e. deployability and composability) for openness. According to our analysis, to alleviate the performance issue, a combination of centralized and semi-centralized data provision strategies could be considered for providing data to different types of third-party applications in different layers of the platform.

Table 7. Evaluating Design Options against Identified Design Requirements

Decoupling
Option 1(1) [Platform] **(-)**: Centralized data provision increases the interactions between third-party applications and the platform since all data access operations should pass through the platform. (2) [Third-Party Applications] **(+)**: Central control by the platform eliminates any one-to-one interaction between third-party applications.
Option 2(1) [Platform] **(-)**: Platform is involved in data write operations between third-party applications. This increases the interactions between the platform and third-party application. (2) [Third-Party Applications] **(-)**: Since the applications can interact with each other directly, the interactions between third-party applications increase.
Option 3(1) [Platform] **(+)**.Third-party applications can exchange data without platform control. Application interactions are thus decoupled from the platform. (2) [Third-Party Applications] **(-)**: Applications can interact with each other directly. Thus the interactions between third-party applications increase.
Development Asynchronization [Third-Party Applications]
Option 1**(+)**: Prohibiting direct communications between third-party applications separates the integration and validation of third-party application from each other.
Option 2**(-)**: The correctness of the behavior of third-party applications should be validated in combination with the related third-party applications.
Option 3**(-)**: This design is similar to Option 2.
Independent Behavior [Third-Party Applications]
Option 1**(+)**: Since all the data interactions are controlled by the platform, the behavior of the applications are completely separated and independent from each other.
Option 2**(-)**: The applications installed later can access the data of the applications that are installed earlier.
Option 3**(-)**: This design is similar to option 2.
Independent Deployment [Third-Party Applications]
Option 1**(+)**: The platform prohibits direct communications between third-party applications. Therefore, third-party applications can be deployed independently of each other.
Option 2**(-)**: Third-party applications can request access to the data of other third-party applications at install-time. This kind of requests violates the independent deployment of applications.
Option 3**(-)**: Third-party applications can send data requests to other applications at any time (either at install-time or after that).
Availability [Third-Party Application Data]: Failure Recovery
Option 1**(+)**: The platform is informed if a third-party application becomes unavailable. Therefore, data requests for unavailable data can be mitigated proactively.
Option 2**(+):** The supervisor (either platform or end-user) is informed of possible unavailability of third-party applications. Therefore, a data request for an unavailable third-party application can be mitigated proactively.
Option 3**(-):** In decentralized communications, the unavailability of applications is not known beforehand. Therefore, a data request for an unavailable third-party application lead to an unmitigated failure.
Integrity [Data] : Consistency [Data]
Option 1**(+)**: Platform controls every data access and modifications between third-party applications. This centralized access control reduces the chance of inconsistency in data read and write operations.
Option 2**(+)**: Data write operation can be supervised by the platform. This supervised access control reduces the chance of inconsistency in data read and write operations.
Option 3**(-)**: Third-party applications can interact with each other without informing central control. This increases the possibility of data inconsistencies in several data read and write operations by different third-party applications.
Response Time [Platform]: Access Time [Data]
Option 1**(-)**: All data operation requests should pass through a central gateway and queue controlled by the platform. Central checking increases the waiting time of third-party applications that require to access data around the same time, even if the requests are for different data from different third-party applications.
Option 2**(+)**: Many of unwanted waiting time for data requests, specifically data read operations, can be eliminated, because third-party applications can directly request data read from other third-party applications.
Option 3**(+)**: Data read operations are handled similar to option 2. Moreover, there will be no central queue for data write operations since the third-party application that provides data is responsible for consistency checking.

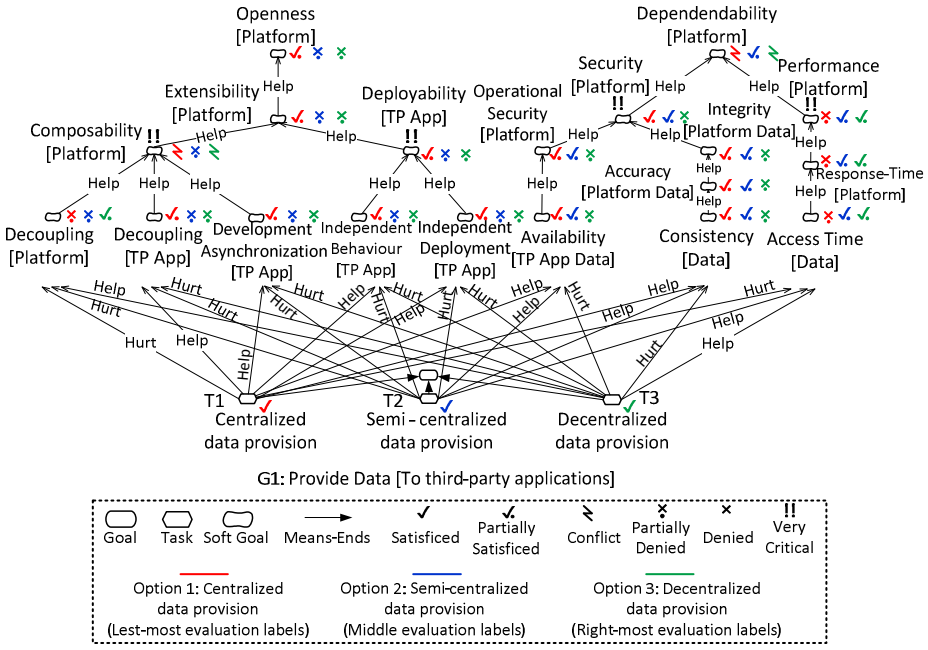

Fig. 1. Modeling and Analyzing Trade-Offs between Openness and Other Requirements in Alternative Data Provision Design Options Using i* Goal-Oriented Language

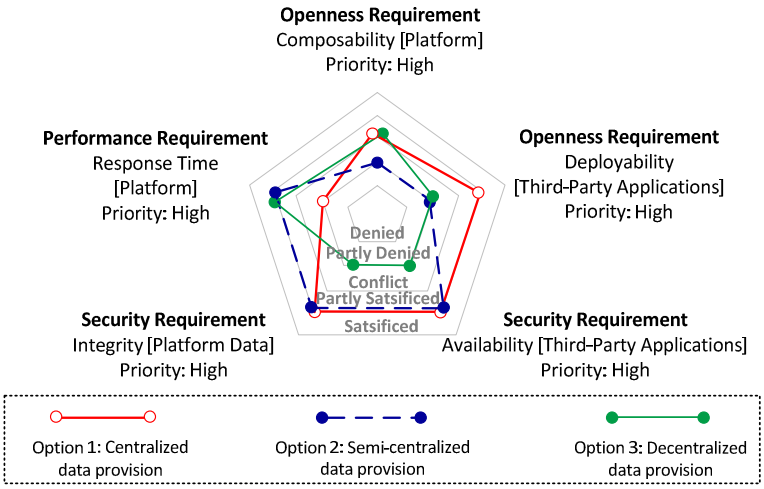

Fig. 2. Comparing Design Strategies for Opening up Platform Data to Third-Party Applications Based on Important Design Requirements for the Automotive Platform

The presented modeling and analysis is only one design scenario among several others that we have investigated in the design of AUTOSAR platform. We aim to confirm our findings with the original designers in a future case study.

5 Related Research

Three groups of research efforts relate to this paper: (1) *Designing open software platforms*: Various efforts have been dedicated to the design and development of software platforms that can smoothly accommodate third-party applications. Most of these efforts focus on providing best practices and techniques for developing APIs that enable seamless and secure communications with third-party applications. (e.g. [21]). Little attention has been given to model-based approaches for designing open software platforms. In a few research works, the need for systematic modeling and analysis in open platforms has been discussed (e.g. [26, 27, 28, 29]) However, no validated modeling method has been proposed for this purpose yet. (2) *Requirements engineering in open software platforms:* Many recent research efforts have investigated the practice of requirements engineering in open software platforms (e.g. [7, 8], [30, 31, 32]). These efforts either focus on identifying the challenges of requirements engineering practices in the presence of multiple development parties or characterising the multi-faceted nature of requirements in open software platforms. A few research works also emphasize the need for rigorous modeling and checking of the requirements in open platforms (e.g. [22]). To support requirements modeling and analysis in open software platforms, several attempts have been made (e.g. [29], [33, 34]) which are in the early stages of development. (3) *Decision Support for open software platforms*. Another group of research works discuss the need to support systematic decision making of open platforms owners and designers (e.g. [33]). However, these efforts mainly focus on adopting open-source components rather than design reasoning support for opening up platforms.

6 Conclusions

We presented a goal-oriented method to model and analyze openness requirements and related trade-offs in designing software platforms. Modeling and analysis of openness trade-offs allows to formally compare alternative design strategies for opening up a platform to third-party applications. This systematic comparison helps determine “good-enough” and “open-enough” design strategies for opening up a platform to third-party applications. Adopting such balanced design strategies is essential to developing open software platforms that are technically, socially and economically balanced, thus having a higher chance of sustainability.

The proposed approach allows to model the relation between business-level and system-level openness requirements, and provides semi-automated support to assess alternative design options and to spot trade-off points.

To improve the applicability of the presented method for modeling and analyzing openness trade-offs, two issues need to be addressed: (1) *Scalability of modeling*: In

this work, we illustrated trade-off modeling in a single design decision in an open software platform. Applying trade-off modeling and analysis at the scale of a design process comprised of numerous interrelated decisions need to be further addressed. (2) *Scalability of analysis*: To find appropriate design options, we used the goal-oriented forward evaluation method, which exhaustively evaluates all the possible options to reach to the best alternative. To improve the efficiency of analysis, algorithms are required which eliminate this exhaustive search.

Moreover, future work is required to compare the proposed method with existing methods for analyzing trade-offs, including Architecture Trade-off Analysis Method (ATAM) [35], and to assess the applicability of the proposed method in real-world open software projects.

This research work is the first step to support design and decision making in open software platforms. The next steps towards this ultimate objective aim to enrich the proposed method in three ways: (1) to provide knowledge support via developing modules for refining openness design requirements as a class of non-functional requirements and developing catalogues of options for designing open platforms; (2) to enrich the analytical and reasoning capabilities of the presented method for incorporating the priorities and preferences of multiple parties in selecting optimal design options in open software platforms; and (3) to provide semi-automated tool support for modeling and analyzing requirements in open software platforms and finding optimal design options.

7 References

1. West, J. (2003). How open is open enough?: Melding proprietary and open source platform strategies. Research policy, 32(7), 1259-1285.
2. Chesbrough, H. W. (2006). Open innovation: The new imperative for creating and profiting from technology. Harvard Business Press.
3. Fitzgerald, B. (2006). The transformation of open source software. MIS Quarterly, 587-598.
4. Boudreau, K. (2010). Open platform strategies and innovation: Granting access vs. devolving control. Management Science, 56(10), 1849-1872.
5. 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.
6. Munir, H., Wnuk, K., & Runeson, P. (2015). Open innovation in software engineering: a systematic mapping study. Empirical Software Engineering, 1-40.
7. Knauss, E., Yussuf, A., Blincoe, K., Damian, D., & Knauss, A. (2016). Continuous clarification and emergent requirements flows in open-commercial software ecosystems. Requirements Engineering, 1-21.
8. Valenca, G., Alves, C. M., Heimann, V., Jansen, S., & Brinkkemper, S. (2014). Competition and collaboration in requirements engineering: A case study of an emerging software ecosystem. In IEEE 22nd International Requirements Engineering Conference (384-393).
9. 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.
10. Koch, S., & Kerschbaum, M. (2014). Joining a smartphone ecosystem: Application developers' motivations and decision criteria. Information and Software Technology, 56(11).

11. 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.
12. 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.
13. Scacchi, W. (2007). Free/open source software development: Recent research results and methods. Advances in Computers, 69, 243-295.
14. Feller, J., & Fitzgerald, B. (2000). A framework analysis of the open source software development paradigm. In Proceedings of the twenty first international conference on Information systems (58-69).
15. Popp, K. M. (2010). Goals of Software Vendors for Partner Ecosystems–A Practitioner´ s View. In Software Business (181-186). Springer Berlin Heidelberg.
16. Bosch, J. (2012). Software ecosystems: Taking software development beyond the boundaries of the organization. Journal of Systems and Software, 85(7), 1453-1454.
17. Jarke, M., Loucopoulos, P., Lyytinen, K., Mylopoulos, J., & Robinson, W. (2011). The brave new world of design requirements. Information Systems, 36(7), 992-1008.
18. 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).
19. Eklund, U., & Bosch, J. (2014). Architecture for embedded open software ecosystems. Journal of Systems and Software, 92, 128-142.
20. Bosch, J. (2010). Architecture challenges for software ecosystems. In Proceedings of the Fourth European Conference on Software Architecture: Companion Volume (pp. 93-95).
21. 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).
22. Scacchi, W., & Alspaugh, T. A. (2013). Processes in securing open architecture software systems. In Proceedings of International Conference on Software and System Process.
23. Baresi, L., Di Nitto, E., & Ghezzi, C. (2006). Toward open-world software: Issue and challenges. Computer, 39(10), 36-43.
24. Chung, L., Nixon, B. A., Yu, E., & Mylopoulos, J. (2012). Non-functional requirements in software engineering (Vol. 5). Springer Science & Business Media.
25. Horkoff, J., & Yu, E. (2013). Comparison and evaluation of goal-oriented satisfaction analysis techniques. Requirements Engineering, 18(3), 199-222.
26. 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.
27. Boucharas, V., Jansen, S., & Brinkkemper, S. (2009). Formalizing software ecosystem modeling. In Proceedings of the 1st international workshop on Open component ecosystems (41-50).
28. Sadi, M. H., & Yu, E. (2015). Designing Software Ecosystems: How Can Modeling Techniques Help? In Enterprise, Business-Process and Information Systems Modeling (360-375).
29. Sadi, M. H., Dai, J., & Yu, E. (2015). Designing Software Ecosystems: How to Develop Sustainable Collaborations?. In Advanced Information Systems Engineering Workshops (161-173).
30. Wnuk, K., & Runeson, P. (2013). Engineering open innovation–towards a framework for fostering open innovation. In International Conference of Software Business (48-59).
31. Linåker, J., Rempel, P., Regnell, B., & Mäder, P. (2016). How Firms Adapt and Interact in Open Source Ecosystems: Analyzing Stakeholder Influence and Collaboration Patterns. In

International Conference on Requirements Engineering: Foundation for Software Quality (63-81).
32. Linåker, J., Regnell, B., & Munir, H. (2015). Requirements engineering in open innovation: a research agenda. In Proceedings of the 2015 International Conference on Software and System Process (208-212).
33. Franch, X., & Susi, A. (2016). Risk assessment in open source systems. In Proceedings of the 38th International Conference on Software Engineering Companion (896-897).
34. Sadi, M. H., & Yu, E. (2014). Analyzing the evolution of software development: from creative chaos to software ecosystems. In Research Challenges in Information Science (RCIS), 2014 IEEE Eighth International Conference on (1-11).
35. Kazman, R., Klein, M., Barbacci, M., Longstaff, T., Lipson, H., & Carriere, J. (1998). The architecture tradeoff analysis method. In Engineering of Complex Computer Systems, 1998. ICECCS'98. Proceedings. Fourth IEEE International Conference on (68-78). IEEE.