Modeling and Analyzing Openness Trade-Offs in Software Platforms

A Goal-Oriented Approach

Mahsa H. Sadi and **Eric Yu**

Department of Computer Science

University of Toronto

Feb 28th, REFSQ 2017, Germany

Background

Open Innovation

Software organizations open up their processes and platforms to external developers

- To use external ideas and paths to market

External developers

- A part of a software ecosystem offering applications

Chesbrough, H. W. (2006). Open innovation: The new imperative for creating and profiting from technology. Harvard Business Press.

Fitzgerald, B. (2006). The transformation of open source software. MIS Quarterly, 587-598.

Background

Open Platforms

A platform on top of which 3rd party applications can be built

- The source code is not necessarily available
- Extension mechanisms allow sufficient access

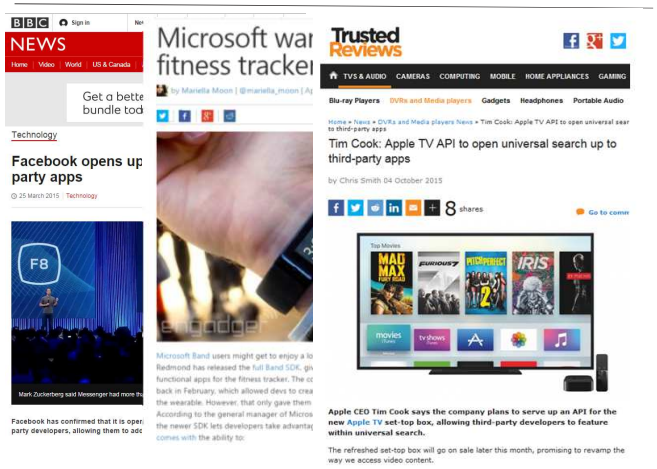
Boudreau, K. (2010). Open platform strategies and innovation: Granting access vs. devolving control. Management Science, 56(10), 1849-1872.

Fitzgerald, B. (2006). The transformation of open source software. MIS Quarterly, 587-598.

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.

Example – Mobile Platforms

Other Examples of Open Platforms



Problem Statement

Opening up platforms is a non-trivial problem

Openness requirements are in competition with security, performance, controllability

e.g. Distributing features versus maintainability

Boudreau, K. (2010). Open platform strategies and innovation: Granting access vs. devolving control. Management Science, 56(10), 1849-1872.

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.

West, J. (2003). How open is open enough?: Melding proprietary and open source plat-form strategies. Research policy, 32(7), 1259-1285.

Research Objective

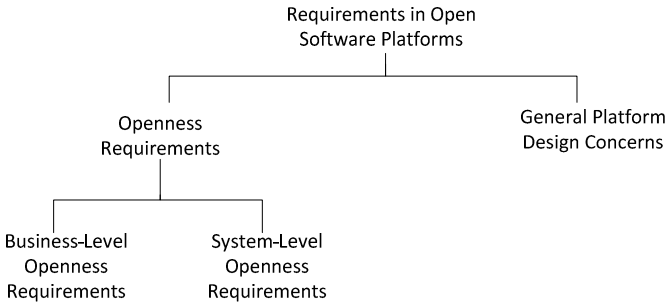
To help identify suitable design strategies for opening up a software platform

Contributions of this Study

1. Requirements and Concerns in Open Software Platforms

2. Modeling and Analysis of Requirements in Open Software Platforms

Requirements and Concerns in Open Software Platforms



Openness Requirements

The specific concerns and quality requirements that openness introduces on platform designs

Openness Requirements

Business-level openness requirements

The motivations for opening up the platform

Non-technical, related to the social, business, and organizational environment of the platform

System-level openness requirements

Technical, related to the quality of software design

Business-Level Openness Requirements – Example

Customer-Related Objectives

Stickiness of the Platform

Growing the network size of complementary applications hardens switching to a different platform

Popp, K. M. (2010). Goals of Software Vendors for Partner Ecosystems–A Practitioner´ s View. In Software Business (181-186). Springer Berlin Heidelberg.

Bosch, J. (2012). Software ecosystems: Taking software development beyond the boundaries of the organization. Journal of Systems and Software, 85(7), 1453-1454.

System-Level Openness Requirements – Example

Extensibility

The ease of adding a new application to a platform

Can be further refined into:

Composability, Deployability, Stability, Configurability

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.

Bosch, J. (2010). Architecture challenges for software ecosystems. In Proceedings of the Fourth European Conference on Software Architecture: Companion Volume (pp. 93-95).

General Concerns in Designing Software Platforms

The requirements possibly violated or at risk in open platforms

General Concerns in Designing Software Platforms – Example

Security

Possible defective or malicious code in external applications may disable the overall system

The integrity of platform services and data

The confidentiality and privacy of the end-users' data

Correct operation of features developed by multiple parties

Scacchi, W., & Alspaugh, T. A. (2013). Processes in securing open architecture soft-ware systems. In Proceedings of International Conference on Software and System Pro-cess.

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

Summary of the First Part

Two Characteristics of the Identified Requirements

Non-Functional, related to:

- Quality requirements (e.g. Accessibility, Extensibility)
- Business objectives (e.g. Stickiness, Ecosystem gravity)

Interacting with each other:

- Competition
- Synergy

Contributions of this Study

1. Requirements and Concerns in Open Software Platforms

2. Modeling and Analysis of Requirements in Open Software Platforms

The Proposed Approach

Non-Functional Requirements Analysis Method

Chung, L., Nixon, B. A., Yu, E., & Mylopoulos, J. (2012). Non-functional requirements in software engineering (Vol. 5). Springer Science & Business Media.

The Case Under Study - 1

AUTOSAR Platform

An Operating System for automotive platforms

Functionalities: Controlling the electronic units of a vehicle

(e.g. the speed, the brakes, the windows)

Opened up to a variety of 3rd party applications

(Certified, uncertified)

Eklund, U., & Bosch, J. (2014). Architecture for embedded open software ecosystems. Journal of Systems and Software, 92, 128-142.

The Case Under Study - 2

The Specific feature under study

The design of Data Provision Service

- How platform communicates data with 3rd party apps
- How 3rd party applications communicate data with each other

Objective: Revisit the Design of Data Provision Service

Step 1: Identifying and Prioritizing Requirements - 1

Domain-Specific Design Requirements

Dependability: Many embedded domains have stringent dependability requirements. These domains are probably not the first adopters of an ecosystem-based approach to software development. However, if that was the case the embedded platform would satisfy; 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.

Eklund, U., & Bosch, J. (2014). Architecture for embedded open software ecosystems. Journal of Systems and Software, 92, 128-142.

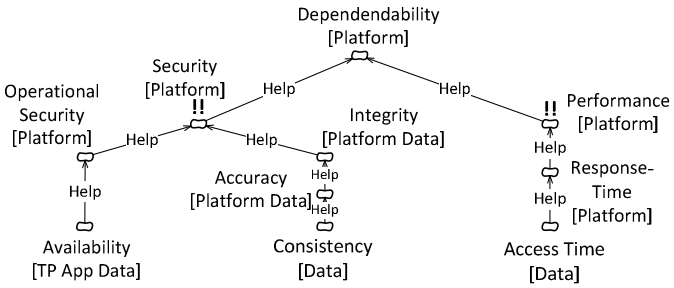
Step 1: Identifying and Prioritizing Requirements - 2

Domain-Specific Design Requirements

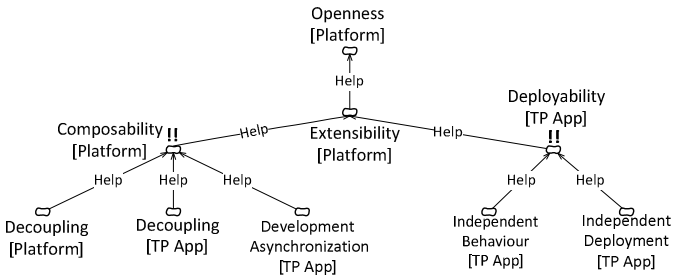
Dependability

Security [Platform] | High	**Performance | High**
Integrity	Response Time
Availability	Access-Time

Step 1: Identifying and Prioritizing Requirements - 3



Step 1: Identifying and Prioritizing Requirements - 6



Step 2: Identifying the Design Objective and Alternative Design Options

Design Objective: To provide data service to 3rd Party applications

5.4.2. *Pattern: Data and service provision*

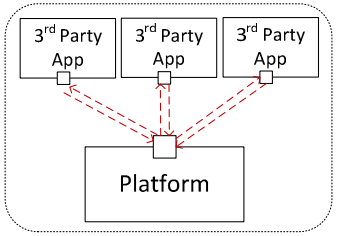
All data is stored and exchanged through the platform (but most data is isolated to a single application) through an API. There are no other mechanisms provided for information exchange between applications besides through the platform. Data and provided services from hardware abstractions are accessed through the API by either an explicit get/set and/or subscribe, both at run-time, i.e. data is pulled by those applications needing it and not pushed to them. There is also an API to determine the available data set at run-time on a particular embedded device.

The platform offers a predictable arbitration mechanism when several applications want to access the same actuator and use a particular service. The arbitration is implemented by the hard-

Eklund, U., & Bosch, J. (2014). Architecture for embedded open software ecosystems. Journal of Systems and Software, 92, 128-142.

Step 2: Identifying Alternative Design Options

Alternative Design 1: Centralized Data Provision



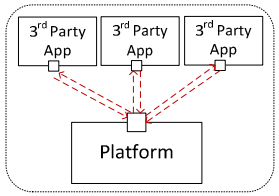
Step 3: Evaluating design options against the design requirements

Design Requirement: Response Time [Platform]: Access Time [Data]

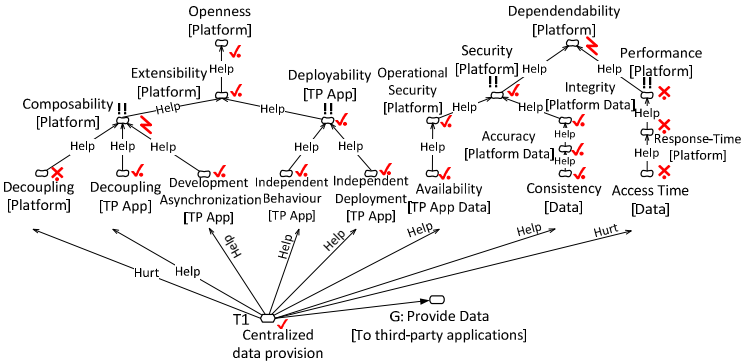
Evaluation: (-)

All data operation requests should pass through a central gateway and a queue controlled by the platform.

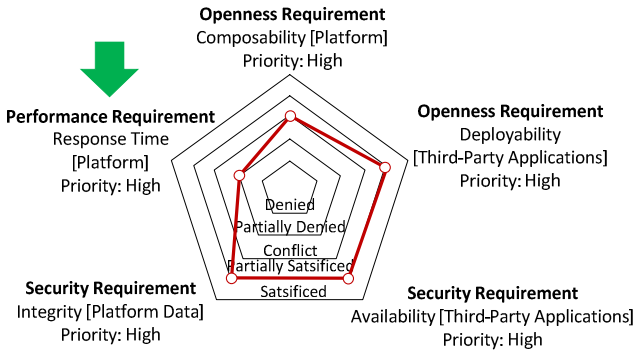
Centralized Data Provision



Analyzing the Fulfillment of Design Requirements in Centralized Data Provision



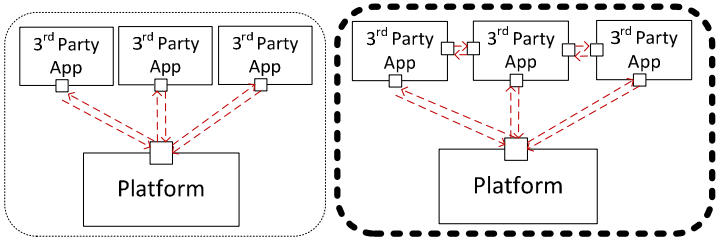
Analysis of Centralized Data Provision



Performance is sacrificed for Openness.

Step 2: Identifying the Alternative Design Options

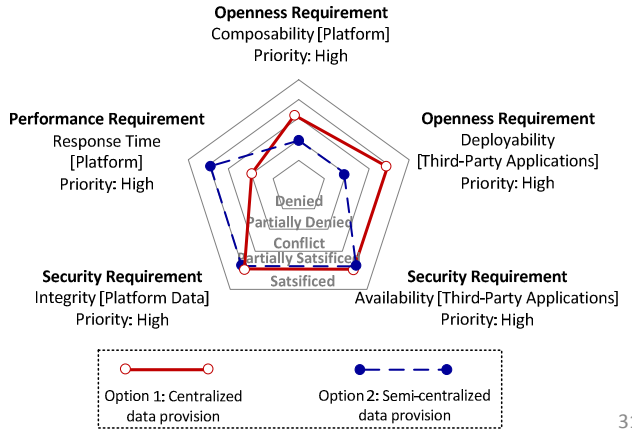
Design Objective: To provide data service to 3rd Party applications



Alternative 1:
Centralized Data Provision

Alternative 2:
Semi-Centralized Data Provision

Comparison of the Design Alternatives



Discussion

Performance: Crucial for real-time Operations of the platform

Composability and Deployability: Crucial for accommodating 3rd party applications overtime

A combination of centralized and semi-centralized data provision can be used.

Summary

Objective:

To identify suitable open platform design strategies

Proposed Solution:

1. Consider openness as a class of non-functional requirements
2. Refine it in parallel with other important concerns
   Use a goal-oriented requirements analysis approach
3. Use it as criteria for selecting a suitable design option
   Analyze potential trade-offs
   Balance the fulfillment of the requirements

Future Work

1. To assess the applicability of the proposed method in real-world open software platform projects
2. To provide knowledge support for refining openness requirements
3. To enrich the analytical capabilities of the proposed method for determining effective openness design strategies
4. To compare with peer approaches, such as ATAM

?

E-mail: mhsadi@cs.toronto.edu