Modeling and Analyzing Architectural Diversity of Open Platforms

Bahar Jazayeri[1(✉)], Simon Schwichtenberg[1], Jochen Küster[2], Olaf Zimmermann[3], and Gregor Engels[1]

[1] Paderborn University, Paderborn, Germany
{bahar.jazayeri,simon.schwichtenberg,gregor.engels}@upb.de
[2] Bielefeld University of Applied Sciences, Bielefeld, Germany
jochen.kuester@fh-bielefeld.de
[3] University of Applied Sciences of Eastern Switzerland, St. Gallen, Switzerland
ozimmerm@hsr.ch

Abstract. Nowadays, successful software companies attain enhanced business objectives by opening their platforms to thousands of third-party providers. When developing an open platform many architectural design decisions have to be made, which are driven from the companies' business objectives. The set of decisions results in an overwhelming design space of architectural variabilities. Until now, there are no architectural guidelines and tools that explicitly capture design variabilities of open platforms and their relation to business objectives. As a result, systematic knowledge is missing; platform providers have to fall back to ad-hoc decision-making; this bears consequences such as risks of failure and extra costs. In this paper, we present a pattern-driven approach called *SecoArc* to model diverse design decisions of open platforms and to analyze and compare alternative architectures with respect to business objectives. SecoArc consists of a *design process*, a *modeling language*, and an automated *architectural analysis technique*. It is implemented and ready-to-use in a tool. We evaluate the approach by means of a real-world case study. Results show that the approach improves the decision-making. Future platform providers can reduce risks by making informed decisions at design time.

1 Introduction

In recent years, prominent software companies succeed in growing by transforming their software products to platforms with open Application Programming Interfaces (APIs), so that third-party providers can develop software on top of them. Usually, online marketplaces are used to distribute the third-party developments of such *open platforms*. Success of open platforms highly depends on suitable design and governance of the ecosystem surrounding them, i.e., the arrangement of human actors and their interaction with the platforms [1]. Such an ecosystem is called *software ecosystem* and is the result of a variable and complex range of architectural design decisions whereas the decisions span across

S. Dustdar et al. (Eds.): CAiSE 2020, LNCS 12127, pp. 36–53, 2020.
https://doi.org/10.1007/978-3-030-49435-3_3

business, application, and infrastructure architectures [2]. Inevitably, different business decisions may result in completely different application and infrastructure architectures [3]. For instance, in ecosystems around mobile platforms (e.g., Apple iOS), the platform providers decide about technical standards like programming languages, which give them control over thousands of third-party providers. In contrary, open source software platforms (e.g., Mozilla Firefox) support innovation whereas developers are free to choose the development environment as well as to publish on the marketplaces. Another group is highly commercialized industrial platforms (e.g., belonging to Citrix and SAP) that are rigorously tested and verified and have networks of strategic partners around them.

Although a diverse range of open platforms has already been created in practice, there is still a lack of systematic architectural guidance to design the ecosystems surrounding them [4]. There is a multitude of Architecture Description Languages (ADLs) that focus on specific operational domains [5], while only supporting needs of those domains, e.g., for mobile applications [6]. In addition, general-purpose languages like UML and Enterprise Architecture Modeling (EAM) like ArchiMate do not allow architects to directly capture diverse design decisions of open platforms and more importantly to analyze suitability of ecosystem architecture with respect to business objectives and quality attributes [7]. The generality and abundance of notations in these languages lead in practice to laborious and time-consuming work [8]. More importantly, some crucial characteristics like *platform openness* are not addressed by these languages [9]. Therefore, the companies that also wish to open their own platforms have to only rely on ad-hoc decision-making and face the consequences such as extra costs of developing useless features, irreversible business risks of unwillingly exposing platform's intellectual property, or technical debt due to sub-optimal decisions [10]. A language that provides the suitable abstraction and captures the architectural diversity of open platforms can facilitate systematic creation of these systems in the future.

In this paper, we present a pattern-driven approach based on a study of 111 open platforms, called *SecoArc*, to model ecosystem architectures and analyze their suitability with respect to business objectives. The contribution of this paper is threefold. SecoArc provides: A) a systematic *design process* for stepwise development of ecosystem architecture. B) a *modeling language* to explicitly express design decisions of open platforms. The language is grounded on a rich *domain model* that embeds the architectural diversity of 111 platforms and a systematic literature review as our preliminary work [11–13]. C) a fully automated *architectural analysis technique* based on three patterns for suitability assessment and in-depth comparison of alternative architectures. SecoArc [14] has been implemented as a ready-to-use tool.

We evaluate the approach by means of a real-world case study called *On-The-Fly Computing Proof-of-Concept (PoC)*. The PoC platform is going to be open. However, there are several architectural variabilities and the possibility of achieving different architectures. The team would like to ensure the suitabil-

ity of future ecosystem architecture with respect to crucial business objectives. Two alternative architectures are designed, analyzed, and compared. Using a semi-structured interview [15] aligned with ISO/IEC 25010 [16], functional suitability of SecoArc is evaluated. Results show that *i*) SecoArc contributes to familiarizing the architect with the domain knowledge of open platforms. *ii*) the automatic architectural analysis technique improves decision-making by disclosing enhanced and degraded business objectives and quality attributes. *iii*) availability of the tool and guide material directly impacts the uptake. In the future, platform providers can reduce risks by making informed decision-making and accordingly take actions at design time. In the following, Sect. 2 introduces three architectural patterns and the case study, PoC. In Sect. 3, we extract architectural variabilities of the PoC. Section 4 describes SecoArc and its elements. There, the applicability of SecoArc is demonstrated by means of the case study. Section 5 elaborates on the evaluation. After discussing related work in Sect. 6, Sect. 7 concludes and discusses future research directions.

2 Background

In this section, first, we present three architectural patterns that are the basis of architectural analysis in SecoArc. Afterwards, we introduce the case study.

2.1 Architectural Patterns of Software Ecosystems

In practice, patterns are used as powerful design instruments to communicate well-established knowledge [17]. In our previous work [11], we identify three architectural patterns that are derived from an examination of 111 open platforms from diverse application domains. Each pattern is characterized by an *organizational context* and a set of *design decisions*, and it helps to achieve certain *quality attributes* and *business objectives* as depicted in Fig. 1. The quality attributes are the attributes of business ecosystems from the quality model introduced by Ben Hadj Salem Mhamdia [18]. In the following, we present the architectural patterns.

Open Source Software (OSS)-Based Ecosystem: Innovation. This pattern aids to attract providers of open source software services. The providers are non-commercially motivated, and the platform is not safety-critical. The high degree of openness, e.g., open and free platforms, enhances creativity by opening the ecosystem for innovative services. Here, a platform is considered open when the third-party providers can directly contribute to the platform's code whereas a platform being free concerns the dimension of cost. Examples of ecosystems are Mozilla, Eclipse, and Apache Cordova.

Partner-Based Ecosystem: Strategic Growth. The pattern helps to grow complex and industrial software ecosystem while enhancing profitability. Platform provider strategically opens the platform by establishing partnerships with

professional providers. The intellectual property is protected by applying monetization and openness policies, e.g., by defining entrance requirements and closing the source code. Exemplary ecosystems are Citrix, Symantec, and SAP.

Resale Software Ecosystem: Business Scalability. The pattern helps to gain business scalability and get control over a large market of third-party services. Platform provider is a large company. Providing testing frameworks and discovery features, e.g., rating and ranking, establish a sustainable development and marketing environment. Additionally, providing execution resources for services and Integrated Development Environment (IDE) enhance interoperability. Apple, Adobe, and Salesforce are the exemplary ecosystems.

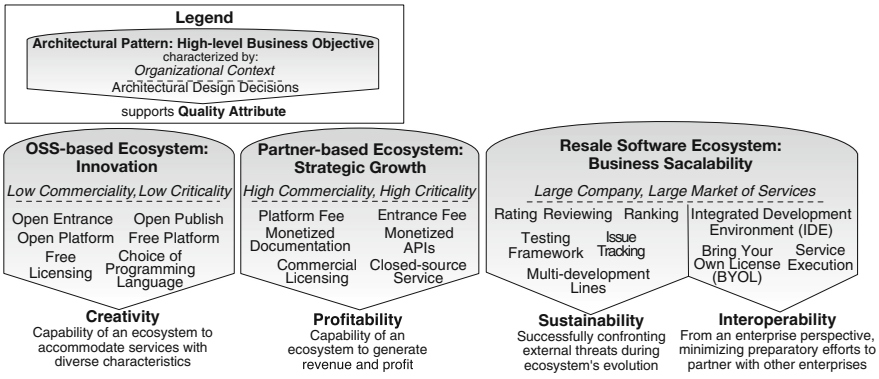

Fig. 1. Three architectural patterns for software ecosystems [11]

2.2 Case Study

On-the-fly Computing (OTF)[1] is a paradigm for the provision of tailor-made IT services that stems from a research project. Service providers publish basic services on a marketplace. Upon user's request, customized IT services are configured on-the-fly from a set of basic services. *The Proof-of-Concept (PoC)*[2] is a realization of the OTF paradigm with *60k+* lines of code. The basic services come from the Machine Learning (ML) libraries Weka[3], in Java, and Scikit-learn[4], in Python. With the help of PoC, users automatically create tailor-made ML services for typical classification problems without having any prior knowledge of ML. E.g., from labeled pictures of cats and dogs, the PoC learns a classification model that predicts if an unlabeled picture shows a dog or cat.

In dialogue with a *chatbot*, the user submits her (non-)functional requirements and the training data. The chatbot broadcasts the requirements to *configurators*.

[1] sfb901.uni-paderborn.de, Last Access: March 20, 2020.
[2] sfb901.uni-paderborn.de/poc, Last Access: March 20, 2020.
[3] cs.waikato.ac.nz/ml/weka, Last Access: March 20, 2020.
[4] scikit-learn.org, Last Access: March 20, 2020.

Configurators are computer programs to find and compose basic services. They try out different combinations of basic services. Once suitable combinations are found, the services with different non-functional properties are offered to the user. Upon the user's acceptance, her personal service is deployed for execution in a *compute center*.

3 Architectural Variabilities of the PoC

Goal of the PoC team is to open the platform to external service providers, so that their third-party ML services will be composed on the basis of the PoC platform. The team consists of managers responsible to take strategic decisions, e.g., defining business visions, ten domain experts for conceptualizing single components like the chatbot, a chief architect responsible for designing and integrating the system components, and ten programmers developing the components. The domain experts have an interdisciplinary background in business and economics, software engineering, and networking and infrastructure areas. Correspondingly, they hold different requirements.

The team is confronted with several variation points that can result in different architectures. In general, the variation points are the design decisions that can be changed at the design time, without violating core functional requirements. To extract the variation points, we conduct a workshop with the PoC team and two external service providers. During the workshop, the participants express important quality attributes in form of elaborated and prioritized scenarios. From these scenarios, the variation points and their variants are identified in three categories of business, application, and infrastructure. Table 1 shows the variabilities. The variants are not mutually exclusive, i.e., several variants of the same variation point can be applied at the same time. More details on the workshop can be found in [14].

While different ecosystem architectures can be built upon the variabilities, different architectures support different business objectives. The PoC team wants to decide on the most suitable architecture among two alternative choices:

> ***Architecture #1: Open Ecosystem*** *The platform remains an independent open-source project, so that the ecosystem grows by the direct contributions of service providers. All source codes should be free-to-use.*
> ***Architecture #2: Semi-Open Controlled Ecosystem*** *If the number of service providers drastically increases, the ecosystem openness might cause degradation in the quality of services. In this case, the PoC team wants to prevent unleashed growth of the ecosystem by establishing a controlled software development and marketing environment.*

Moreover, the following business vision should always be fulfilled:

> ***Business Vision:*** *The ecosystem should support innovation, while being sustainable in terms of confronting external threats that could have a long-term impact on the platform's success. Furthermore, the platform ownership should be managed by using the GNU General Public License (GPL).*

Table 1. Architectural variabilities of the PoC ecosystem

Business	**Entrance Quality Check (VP1)**	(**v1.1**) Entrance to the ecosystem is subject to no quality check. (**v1.2**) If necessary, using static code analysis, quality of third-party services should be checked. (**v1.3**) Third-party providers' qualification may need to be evaluated.
	Entrance Fee (VP2)	(**v2.1**) Entering the ecosystem is free. (**v2.2**) However, in some cases, the third-party providers need to pay an entrance fee.
	Service Fee (VP3)	The ecosystem supports (**v3.1**) free and (**v3.2**) paid third-party services.
	Access to Platform (VP4)	(**v4.1**) The platform's source code is open so far the platform quality is not threatened. (**v4.2**) The team might want to limit the access if necessary.
	Access to Third-party Services (VP5)	(**v5.1**) Basic services need to be open source. (**v5.2**) Protecting intellectual property of services becomes service providers' right.
Application	**Marketplace (VP6)**	(**v6.1**) While the default is to use available service repositories, (**v6.2**) the team is ready to establish own marketplace.
	Programming Language (VP7)	The ecosystem leverages two programming languages for third-party services, i.e., (**v7.1**) Java and (**v7.2**) Python.
Infrastructure	**Hardware Allocation (VP8)**	(**v8.1**) Compute centers dedicate fixed hardware resources to basic services. (**v8.2**) Compute centers dynamically allocate the resources at run-time based on resource availability and performance metrics.
	Distribution of Execution (VP9)	(**v9.1**) Execution of basic services is distributed on several compute centers. (**v9.2**) Each basic service is executed centralized by a compute center.

Variation Point (VP_i) / Variant ($v_{i,j}$)

4 Modeling and Analyzing Architectural Diversity by Using SecoArc

SecoArc (Architecting Software Ecosystems) facilitates designing ecosystems around software platforms by means of three main elements, i.e., a **design process**, a **modeling language including a domain model and modeling workbench**, and an **architectural analysis technique**. These elements are already implemented by using Eclipse Modeling Framework[5] and Eclipse Sirius Framework[6] in a tool [14]. Figure 2 shows the SecoArc design process using Business Process Model and Notation (BPMN). The goal of the design process is to assist architects in a step-wise realization of suitable ecosystem architecture. It comprises the following activities: *specify organizational context of platform provider*, *model business, application, and infrastructure design decisions of ecosystem architecture using the SecoArc modeling language*, and *assess fulfillment of business objectives and quality attributes using the SecoArc architectural analysis technique*. If the architecture does not fulfill the objectives and attributes, further ecosystems can be modeled and then compared. After extracting the variabilities of PoC (cf. Table 1), the chief architect models, analyzes, and compares *Architectures #1* and *#2* using SecoArc. The architect familiarizes himself with the SecoArc using the user guide provided by the SecoArc specification [14].

[5] eclipse.org/emf, Last Access: March 20, 2020.
[6] eclipse.org/sirius, Last Access: March 20, 2020.

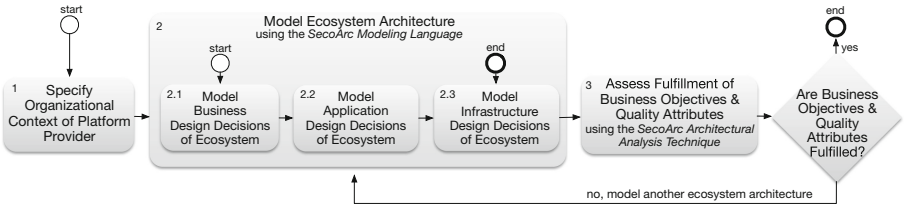

Fig. 2. SecoArc design process (Notation: BPMN)

4.1 Specify Organizational Context of Platform Provider

Designing an ecosystem begins with specifying the context of an organization. This is because a great part of architectural design decisions is derived from the organizational context. In addition, these contextual factors are considered during the SecoArc architectural analysis. Significance of four contextual factors are frequently supported by the literature on software ecosystems, e.g., [3]: *company size*, *market size*, *domain criticality*, and *commerciality*. Company size refers to the number of employees in an organization. Market size is the number of services on a marketplace. Domain criticality determines whether software failure is dangerous to human lives. Commerciality is the degree of protecting intellectual property.

4.1.1 Organizational Context of the PoC

Since the contextual factors refer to the organizational context of PoC, they remain the same for both *Architectures #1* and *#2*. The platform is open, free-to-use, and not safety-critical. With *<100* employees and *<100* third-party services, the PoC is considered a small size organization with a medium size market. The relations between the range of values and the company and market size are specified in [11].

4.2 Model Ecosystem Architecture

Architects can use the modeling language to design an ecosystem architecture starting with the business decisions, and then the application and infrastructure decisions. The language aims at 1) facilitating explicit expression of design decisions of software ecosystems, 2) capturing architectural diversity, and 3) providing a modular integration of business, application, and infrastructure.

The modeling language is grounded on a rich *domain model* for software ecosystems. The domain model is a metamodel (a.k.a abstract syntax and its semantics) that covers key design decisions of software ecosystems. It is based on the examination of literature [13] and existing ecosystems [11,12]. In addition, a *modeling workbench* provides the visual notations (a.k.a. concrete syntax and its semantics) to design architecture. Both the domain model and modeling workbench span across business, application, and infrastructure architectures. Due to space limitations, the paper does not intend to describe complete lists of the language constructs. Complete lists are provided by the SecoArc specification [14].

4.2.1 Model Business Design Decisions of Ecosystem

Figure 3 presents the domain model of SecoArc. At the top, the `BusinessArchitecture` is shown. `SoftwareEcosystem` consists of `HumanActor`, `BasicSoftwareElement`, and `BusinessAction`. A `HumanActor` can be a `User`, `EcosystemProvider`, `ServiceProvider`, and `Partner`. A `BasicSoftwareElement` can be a `SoftwarePlatform`, `Marketplace`, and `Service`. The choice of `BasicSoftwareElement` is a fundamental decision that is taken early in the design phase with other business decisions. Using `BusinessActions`, business decisions can be defined, modified, or reused across the actors.

The domain model captures three significant groups of business decisions in software ecosystems, i.e., `Fee`, `OpennessPolicy`, and `QualityCheck`. `Fee` can be `PlatformFee` to use the platform, `EntranceFee` to enter the ecosystem, or `ServiceFee` to use a `Service`. A `ServiceFee` is defined by a `ServiceProvider` or `Partner`. Furthermore, using `OpennessPolicy`, the architect specifies which actors, and to which extend, can access or own a `SoftwarePlatform` or `Service`. Here, `PlatformOpenness` determines whether one can only contribute to the platform development or he/she has equity ownership of the platform. One way to regulate `PlatformOpenness` is where the third-party developers of the Mozilla Firefox web browser develop new functionality for the browser without having its ownership. Another `OpennessPolicy` is `ServiceOpenness` to grant a license to publish or to share intellectual property of services. For instance, in the Apple ecosystem, the developers receive license to publish on the Apple App Store whereas, in the Cloud Foundry ecosystem, third-party services belong to their providers and can be traded outside of the ecosystem [11]. Finally, using `QualityCheck`, it is defined whether a `Service` needs to pass `StaticCodeAnalysis` before being published or whether any of `HumanActors` should fulfill certain entrance requirements that are specified as an `EntranceCertificate`.

4.2.1.1 ***Business Design Decisions of Architecture #1***

To keep the ecosystem fully open, entering the ecosystem is free for the users and service providers (Table 1: **v2.1**). The third-party services are free (**v3.1**) and released under the GPL license (**v5.1**). To enable third-party providers' contribution to a full extent, the team will provide open code repositories (**v4.1**). The only entrance barrier for the service providers is to pass a static code analysis (**v1.2**).

4.2.1.2 ***Business Design Decisions of Architecture #2***

To tackle an uncontrolled growth, several business decisions are to be taken into account. Figure 4 shows the business decisions designed using the SecoArc notations. Human actors, like `Providers of ML Services`, interact with the `BasicSoftwareElements` on the basis of `BusinessActions`. The symbols on the actions, e.g., $E_{€}$ and $S_{€}$, represent the business decisions that impact those actors.

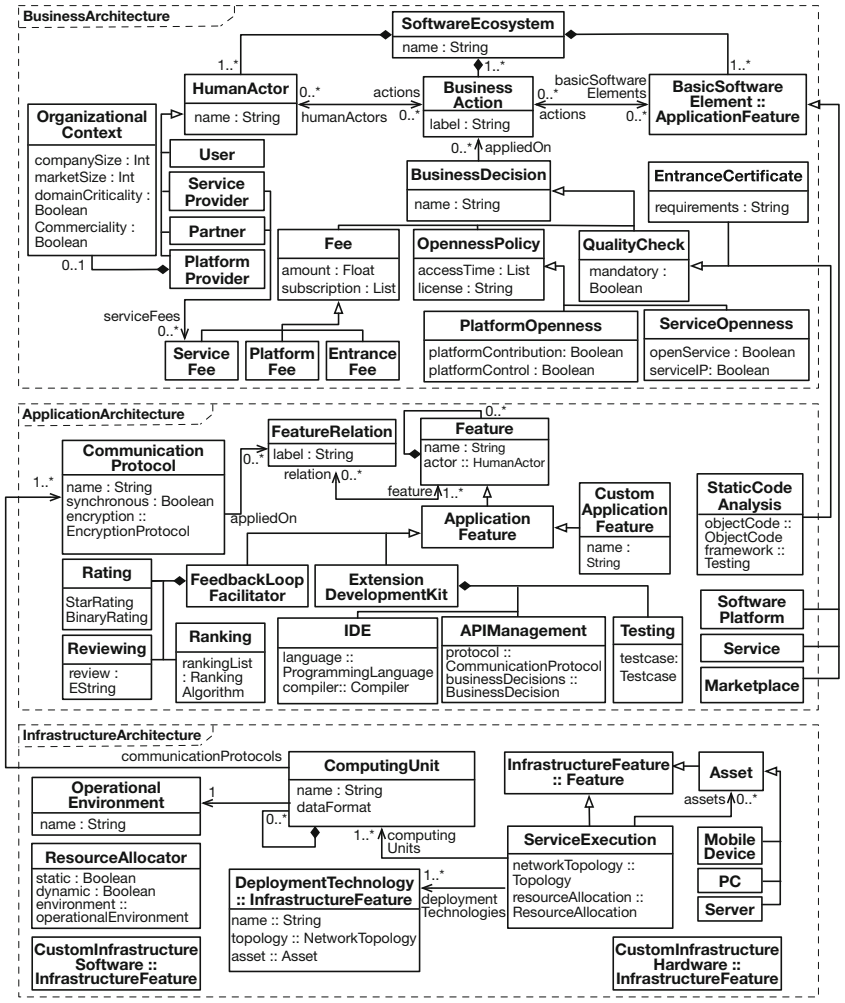

Fig. 3. Domain model for software ecosystems (Notation: UML Class Diagram)

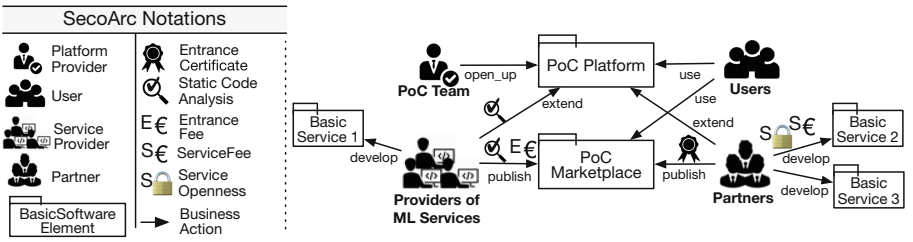

Fig. 4. *Architecture #2*: Business decisions modeled using SecoArc (For readability purposes, adopted from the original screenshot [14])

Firstly, to create a controlled marketing environment, the team considers providing an own marketplace (**v6.2**). This way, the providers are forced to continuously improve their services in order to beat the increased market competition. In addition, membership controls in terms of *1€* annual `EntranceFee` for the providers are to be set (**v2.2**) (shown by $E_{€}$ in Fig. 4). Note that the notations in Fig. 4 do not show attributes of the business decisions. However, the SecoArc modeling workbench has a *properties view* that allows to define them. According to the business vision in Sect. 3, the PoC platform remains free and under the GPL license. Thus, `PlatformContribution` is `true` (**v4.1**).

Secondly, to ensure providing high quality services, the team considers partnering with other companies. In this case, `Partners` need to *"possess certain amount of annual revenue"*, which is specified using an `EntranceCertificate` in Fig. 4. Moreover, the partners are allowed to monetize their services and the belonging documentation by defining price models (**v3.2**), closing the source code, or use their own licenses (BYOL) (**v5.2**). Thereby, two types of services are created for `Partners`, i.e., `Basic Service 2` and `3`, whereas `Basic Service 2` has `ServiceFee` and `ServiceOpenness` policies defined. The rest of business decisions remain the same as *Architecture #1*.

4.2.2 Model Application and Infrastructure Design Decisions of Ecosystem

The application architecture supports business decisions by its application merits whereas the infrastructure architecture provides the application architecture with computing and deployment resources. Such resources are the software and hardware for the purpose of service execution. The domain model in Fig. 3 shows the `Application-` and `InfrastructureArchitecture` that respectively include `Application-` and `InfrastructureFeature`. In general, a `Feature` can be accessed by `HumanActor`, be a part of another `Feature`, or be in relation with another `Feature` in the same or another architecture.

1) Modeling of Built-in Features: Part of the domain model captures *built-in features*, i.e., the features specific to software ecosystems and open platforms. The built-in features originate from our study on variabilities of ecosystem architectures [12,13]. As depicted in Fig. 3, the application-specific features are

captured by means of ExtensionDevelopmentKit and FeedbackLoopFacilitator. ExtensionDevelopmentKit represents the software features that enable development on top of open platforms. This mainly includes IDE, APIManagement, and Testing. Android Studio is an example of IDEs used to develop Apps for Google Android. Android SDK is the APIManagement that facilitates accessing the platform APIs. Android Emulator is a Testing feature that is used to simulate variety of hardware for testing purposes. Moreover, FeedbackLoopFacilitator makes user feedback operational in the ecosystem using Rating, Reviewing, and Ranking. The feedback returned to the ecosystem through Rating and Reviewing is used to generate ranking lists and to improve services by service providers.

Furthermore, infrastructure-specific built-in features are grouped as DeploymentTechnology, ServiceExecution, and Asset (cf. Fig. 3). As the names suggest, they represent computing resources to deploy Services on a DeploymentTechnology, to execute them using ComputingUnits, and to deliver the execution results to an *Asset*. A ComputingUnit has an OperationalEnvironment and supports at least one CommunicationProtocol. For instance, Amazon.com is the provider of cloud computing services called Amazon Web Services (AWS). Figure 5 shows that AmazonEC2 is the service to execute the AWS1. It uses virtualMachine1 to execute the AWS1 on an AmazonLinux. HTTPS is the CommunicationProtocol. The service execution is performed on AWSSever1 and AWSSever2. The result of execution is sent to userPC [12].

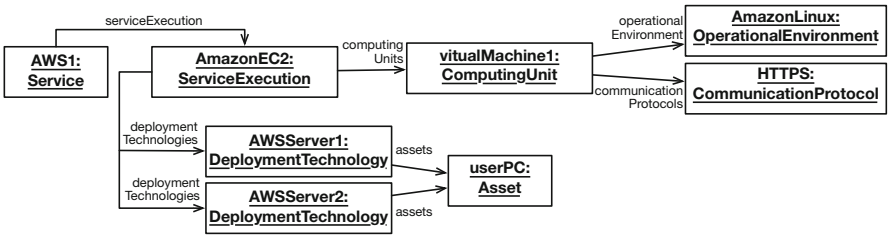

Fig. 5. The UML object diagram of an exemplary AWS service execution

2) Modeling of Custom Features: Custom features are used to design the concepts that are not part of the domain model. Such features can be a CustomApplicationFeature in the ApplicationArchitecture or a CustomInfrastructureSoftware and CustomInfrastructureHardware in the InfrastructureArchitecture.

4.2.2.1 ***Application and Infrastructure Decisions of Architecture #1*** Figure 6 shows the application and infrastructure features in relation to a part of the business architecture designed using the SecoArc notations. Currently, a microservice architecture is used, where the PoC CustomApplicationFeatures, i.e., the Chatbot and Configurator, are deployed on Compute Center 1 and Compute Center 2. To enable third-party contributions to a full extent, the team uses Git

as an open code repository (**v4.1**) (**v6.1**). The default way to publish is to `commit` new services into the repository. Because the services are encapsulated inside the `DockerContainers`, `Providers of ML Services` are allowed to develop both `Basic Service in Java` and `Basic Service in Python` (**v7.1**) (**v7.2**). During the service provision, the execution of basic services is handled by several docker containers. Figure 6 portraits `DockerContainer 1` that consists of two `ComputingUnits`, called `Executor 1`, `Executor 2`. The executors `register` with the `Gateway` in advance (**v8.1**). The `Gateway` is a `ResourceAllocator` that allocates each basic service to multiple executors (**v9.1**).

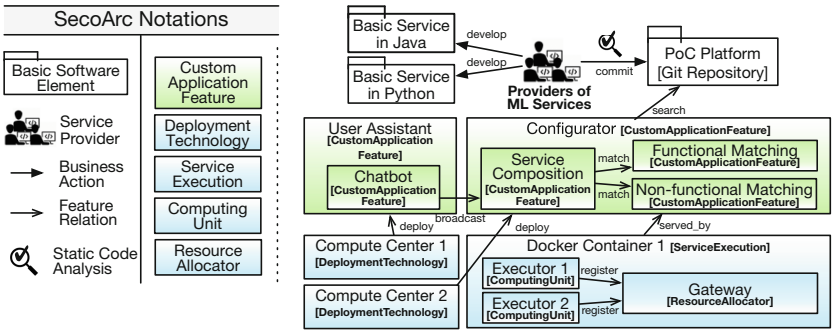

Fig. 6. *Architectures #1*: Application, infrastructure, and part of business architectures (Adopted from the original screenshot [14])

4.2.2.2 ***Application and Infrastructure Decisions of Architecture #2***
Upon deciding on an own marketplace by the PoC team, related software features of the marketplaces, e.g., rating, ranking, and reviewing, are included. To create a controlled development environment, the team considers providing the service providers with an IDE and testing environment. By a drastic increase in the number of service providers, some decisions in *Architecture #1* cause scalability issues: By static resource allocation (**v8.1**), the executors are permanently waiting for jobs, thus, wasting resources if they remain idle. Furthermore, by the distributed execution (**v9.1**), if many executors become incorporated in execution of a service, the network payload will be tremendous. Therefore, the team decides that, for every basic service, a Docker container is generated that contains all the necessary environments as well as incorporated basic services (**v9.2**). A respective container is distributed in a compute center on demand and will be freed up after the execution (**v8.2**).

4.3 Assess Fulfillment of Business Objectives and Quality Attributes Using the SecoArc Architectural Analysis Technique

Ecosystem architecture can be further analyzed by using a pattern-matching technique in the modeling workbench. As previously mentioned, the architectural

Table 2. Comparison of *Architectures #1* and *#2* (The complete report of analysis can be found in the text.)

	OSS-based Ecosystem: Innovation						***Partner-based Ecosystem: Strategic Growth***						***Resale Software Ecosystem: Business Scalability***								
	Creativity						Profitability						Sustainability						Interoperability		
	Open Entrance	Open Publish	Open Platform	Free Platform	Free Licensing	Choice of Programming Language	Platform Fee	Entrance Fee	Monetized Documentation	Monetized APIs	Commercial Licensing	Closed Source Service	Rating	Reviewing	Ranking	Testing Framework	Issue Tracking	Multi-dev. Lines	BYOL	IDE	Service Execution
Architecture #1	✓	✓	✓	✓	✓	✓	×	×	×	×	×	×	×	×	×	×	✓	✓	×	×	✓
Architecture #2	✓	×	✓	✓	×	×	×	✓	✓	×	✓	✓	✓	✓	✓	✓	✓	×	✓	✓	✓

Architectural Design Decision Realized (✓) / Not-realized (×)

patterns in Sect. 2.1 are described using sets of design decisions. The architectural analysis checks to which extend an ecosystem architecture realizes those design decisions. With this respect, architects can 1) assess suitability of an ecosystem architecture and 2) compare alternative architectures. Specifically, the analysis has three outcomes: firstly, to which extend the architecture supports the business objectives, i.e., *innovation*, *strategic growth*, and *business scalability*. Secondly, to which extend it supports the quality attributes, i.e., *creativity, profitability, sustainability*, and *interoperability*. Finally, according to the value of contextual factors, suggestions to apply specific patterns and a list of exemplary real-world ecosystems are given.

4.3.1 Analyzing and Comparing Architectures #1 and #2

The PoC architect conducts the analysis to evaluate suitability of *Architectures #1* and *#2*. A report of the analysis is generated. Table 2 shows how the two architectures match with the patterns. Accordingly, *Architectures #1* and *#2* respectively fulfill OSS-based Ecosystem (*100%*, *50%*), Partner-based Ecosystem (*0%*, *66.6%*), and Resale Software Ecosystem (*33.3%*, *88.8%*). This means, *Architecture #1* fulfills **innovation** the most, while *Architecture #2* enhances **business scalability** and **strategic growth**. Furthermore, the majority of decisions in *Architecture #1* contributes to **creativity** while **sustainability** and **interoperability** are partially supported. The main shift in *Architecture #2* is to include partners, who can generate revenue. Decisions like entrance fee and commercial licenses support **profitability**.

The results of analysis show that, comparing the two architectures, *Architecture #1* is a more suitable candidate with respect to the business vision in Sect. 3,

as it fulfills innovation to a better extend (*100%* vs. *50%*). However, sustainability can be improved by considering the relevant design decisions, e.g., including rating and ranking. In addition, the analysis shows that existing ecosystems, with contextual factors similar to the PoC (cf. Sect. 4.1.1), conform to the OSS-based Ecosystem pattern, which confirms the suitability of this pattern for the PoC. Exemplary ecosystems provided are Mozilla, Eclipse, and Apache Cordova. The full report of analysis can be found in [14].

5 Evaluation

In our evaluation, we investigate on *whether SecoArc supports architects in modeling diverse design decisions and analyzing alternative ecosystem architectures with respect to the business objectives and quality attributes*. For this purpose, we design a questionnaire with which we can interview architects on the quality characteristics of SecoArc. The questionnaire is aligned with the ISO/IEC 25010 quality model [16] and serves the goal to assess the *functional suitability* of SecoArc. In this context, the functional suitability is the degree to which SecoArc satisfies the need of architects and provides value during decision-making. The questionnaire will be used to conduct a semi-structured interview with the chief architect of the PoC, who is responsible to make design decisions and to integrate the various components into one architecture.

5.1 Questionnaire

We design the questionnaire by referring to the sub-characteristics of functional suitability, i.e., *completeness*, *correctness*, and *appropriateness* [16] and casting doubt on the main elements of SecoArc, i.e., *design process*, *modeling language*, and *architectural analysis technique*, as well as *tool support*. As shown in Table 3, the questionnaire consists of *17* questions. The interviewee responds the questions in three scales, i.e., *No/Partially/Yes*, while reasoning about the responses. Using the semi-structured interview technique [15], we collect data on strong and weak aspects of SecoArc based on open conversations. Full responses can be found in [14].

5.2 Results

In summary, the interviewee expresses that the ready-to-use components of SecoArc contribute to an improved decision-making. Table 3 briefly refers to the responses. In the following, we report the results.

1) Modeling Language & Design Process: The architect approves that the mode ling language of SecoArc is a domain-specific language. With the design process, they help with familiarizing oneself with the novel and complex architecture of the ecosystem around the platform. The built-in features provide a correct basis to explicitly design the ecosystem-specific design decisions of PoC. Furthermore, custom features can appropriately be added. This adds flexibility

Table 3. Interview questionnaire and results

Modeling Language & Design Process	Q1	*Does the domain knowledge of SecoArc contribute to familiarizing you with critical design decisions of the ecosystem around your platform?*	+
	Q2	*Does the design process appropriately guide you through the ecosystem design?*	+
	Q3	*Are the business decisions, application and infrastructure modeling features complete in terms of being sufficient for modeling your ecosystem?*	+
	Q4	*Do you find the suggested business decisions and built-in application and infrastructure features correct (or rather contradictory or wrong)?*	+
	Q5	*Does grouping of features help you to better understand the design space?*	+
	Q6	*Are you able to appropriately express custom decisions of your ecosystem?*	+
	Q7	*Are the visual notation of business decisions and application and infrastructure features appropriate?*	+
	Q8	*Are the architectural models easy to change or reuse, so you can use them during the system evolution?*	o
	Q9	*Can you appropriately express the relationships between the three architectures?*	+
Architectural Analysis	Q10	*Is the knowledge of existing ecosystems, embedded in the patterns, beneficial for your decision-making?*	+
	Q11	*Does the analysis help you to consider complementary business decisions and application and infrastructure features wrt. your business objectives?*	+
	Q12	*Does the analysis make you aware of the correctness of your design decisions? E.g., by revealing the contradictory decisions to your business objectives and quality attributes?*	+
	Q13	*Do you find the quality attributes and business objectives complete to address your needs?*	o
	Q14	*Is the comparison of alternative architectures beneficial to your decision-making?*	+
Tool Support	Q15	*Is the modeling workbench easy-to-use to design the ecosystem around your platform?*	+
	Q16	*Are the provided specification and guide materials satisfactory to start using the tool?*	+
	Q17	*Does the tool provide the functionality and features promised by the specification?*	+

No (-) / Partially (o) / Yes (+)

to the language, because the custom features are treated the same way as the built-in features, e.g., to group them or relate them to the human actors. Another point of appropriateness is the abstraction provided by the visual notations and the central presentation of business decisions.

2) Architectural Analysis Technique: The architect finds it beneficial during the decision-making to have an estimation of suitability of the PoC architecture based on the practice-proven knowledge of existing ecosystems and being able to compare different architectures. This also helps to take further actions by including complementary decisions or re-considering the ones that threaten achieving the business objectives and quality attributes.

3) Tool Support: Availability of the tool and guide material impact the uptake by enabling the architect to work with the ecosystem-related concepts right away. The tool is consistent with the functions described in the specification.

4) Limitation and Future Work: Although our approach is concerned with modeling support at design time, there is an inherent need for consistency between the models and code during the system evolution (**Q8**). A way to address this issue is to apply Aspect-Oriented Software Development to weave the decisions in the code, another way is to introduce traceability links [19]. Furthermore, future research on extensive quality models for software ecosystems is required, to address more business objectives, quality attributes, and their trade-offs (**Q13**).

6 Related Work

To our knowledge, there is no previous work that provides a modeling language supporting automatic analysis with respect to business, application, and infrastructure aspects, by considering the knowledge of diverse platforms. ADLs are

the early approaches to describe system architecture [20]. Existing ADLs, most relevant to open platforms, support specific domains, e.g., mobile applications [6] and automotive systems [21] while business aspects are less in focus.

A body of work in literature conceptualizes economic aspects. Boucharas et al. [22] formalize modeling ecosystems in three levels: the software ecosystem, supply network, and software vendor. Yu and Deng [23] propose a graph-based approach based on the i^* modeling framework for strategic buyer-supplier relationships. While this work focuses on software supply chain processes, our focus in on structure of ecosystem architecture. However, this work complements ours and can be leveraged during the business modeling activity.

Further work mainly investigates technical aspects. Christensen et al. [24] present a modeling approach for the organizational, business, and software structures. However, the approach does not deal with the relation between the technical architecture and actors. Sadi et al. contribute to a goal-oriented ecosystem design by an NFR-based specification and analysis method for openness requirements [9]. Furthermore, modeling sustainable collaborations in [25] helps clarify developers' objectives and their dependencies. In comparison to our work, SecoArc embeds the existing domain knowledge and provides automated tool support. Other work [3] surveys ecosystem variabilities. Our work is built on this work while extending them by modeling and analysis support.

7 Conclusion and Future Work

Opening software platforms became a novel architectural approach to enable third-party service development. Designing open platforms faces many architectural variabilities, and informed decision-making is crucial for platform providers to achieve their business objectives. However, there is a lack of comprehensive architectural guidelines that can help to tackle the variabilities while relating the design decisions to business objectives. This paper presents a pattern-driven methodological support comprising a *design process*, *modeling language*, and *architectural analysis technique*, which consolidates the knowledge of *111* open platforms and is ready-to-use as a tool. Our contributions support platform providers in informed decision-making based on the knowledge of best practices. Future work is required to enhance business modeling coupled with application and infrastructure architectures, facilitate trade-off analysis of more quality attributes, and establish consistency between architecture and code.

References

1. Bosch, J.: Software ecosystems: taking software development beyond the boundaries of the organization. J. Syst. Softw. **7**(85), 1453–1454 (2012)
2. REVaMP^2 Project. www.revamp2-project.eu. Accessed 20 Mar 2020
3. Berger, T., et al.: Variability mechanisms in software ecosystems. Inf. Softw. Technol. **56**(11), 1520–1535 (2014)
4. Malavolta, I., et al.: What industry needs from architectural languages: a survey. IEEE Trans. Softw. Eng. **39**(6), 869–891 (2013)

5. Hohpe, G., et al.: The software architect's role in the digital age. IEEE Softw. **33**(6), 30–39 (2016)
6. Dörndorfer, J., Hopfensperger, F., Seel, C.: The SenSoMod-modeler-a model-driven architecture approach for mobile context-aware business applications. In: Cappiello, C., Ruiz, M. (eds.) CAiSE 2019. LNBIP, vol. 350, pp. 75–86. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-21297-1_7
7. Woods, E., Bashroush, R.: Modelling large-scale information systems using ADLs–an industrial experience report. J. Syst. Softw. **99**, 97–108 (2015)
8. Zimmermann, A., et al.: Evolving Enterprise Architectures for Digital Transformations. Gesellschaft für Informatik eV (2015)
9. Sadi, M.H., Yu, E.: Accommodating openness requirements in software platforms: a goal-oriented approach. In: Dubois, E., Pohl, K. (eds.) CAiSE 2017. LNCS, vol. 10253, pp. 44–59. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59536-8_4
10. CIO Magazine: 5 Mistakes to Avoid When Deploying an Enterprise App Store. cio.com/article/2394413. Accessed 20 Mar 2020
11. Jazayeri, B., et al.: Patterns of store-oriented software ecosystems: detection, classification, and analysis of design options. In: Latin American PLOP (2018)
12. Jazayeri, B., Zimmermann, O., Engels, G., Kundisch, D.: A variability model for store-oriented software ecosystems: an enterprise perspective. In: Maximilien, M., Vallecillo, A., Wang, J., Oriol, M. (eds.) ICSOC 2017. LNCS, vol. 10601, pp. 573–588. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-69035-3_42
13. Jazayeri, B., Platenius, M.C., Engels, G., Kundisch, D.: Features of IT service markets: a systematic literature review. In: Sheng, Q.Z., Stroulia, E., Tata, S., Bhiri, S. (eds.) ICSOC 2016. LNCS, vol. 9936, pp. 301–316. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46295-0_19
14. SecoArc Material & Supplementary Documents of the Case Study. sfb901.uni-paderborn.de/secoarc. Accessed 20 Mar 2020
15. Hove, S.E., Anda, B.: Experiences from conducting semi-structured interviews in empirical software engineering research. In: METRICS 2005, pp. 10–pp. IEEE (2005)
16. ISO/IEC 25010. iso.org/standard/35733.html. Accessed 20 Mar 2020
17. Hohpe, G., et al.: Twenty years of patterns' impact. IEEE Softw. **30**(6), 88–88 (2013)
18. Ben Hadj Salem, M.A.: Performance measurement practices in software ecosystem. Int. J. Prod. Perform. Manag. **62**(5), 514–533 (2013)
19. Könemann, P., Zimmermann, O.: Linking design decisions to design models in model-based software development. In: Babar, M.A., Gorton, I. (eds.) ECSA 2010. LNCS, vol. 6285, pp. 246–262. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15114-9_19
20. Medvidovic, N., et al.: A classification and comparison framework for software architecture description languages. IEEE Trans. Softw. Eng. **26**(1), 70–93 (2000)
21. Kolagari, R.T., et al.: Model-based analysis and engineering of automotive architectures with EAST-ADL: revisited. Int. J. Concept. Struct. Smart Appl. **3**(2), 25–70 (2015)
22. Boucharas, V., et al.: Formalizing software ecosystem modeling. In: International Workshop on Open Component Ecosystems, pp. 41–50. ACM (2009)
23. Yu, E., Deng, S.: Understanding software ecosystems: a strategic modeling approach. In: The 3rd International Workshop on Software Ecosystems, pp. 65–76 (2011)

24. Christensen, H.B., et al.: Analysis and design of software ecosystem architectures–towards the 4S telemedicine ecosystem. Inf. Softw. Technol. **56**(11), 1476–1492 (2014)
25. Sadi, M.H., Dai, J., Yu, E.: Designing software ecosystems: how to develop sustainable collaborations? In: Persson, A., Stirna, J. (eds.) CAiSE 2015. LNBIP, vol. 215, pp. 161–173. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-19243-7_17