Tool Support for the Architectural Design Decisions in Software Ecosystems

Srdjan Stevanetic
Software Architecture
University of Vienna, Austria
srdjan.stevanetic@univie.ac.at

Konstantinos Plakidas
Software Architecture
University of Vienna, Austria
konstantinos.plakidas@univie.ac.at

Tudor B. Ionescu
Siemens AG
Vienna, Austria
tudor.ionescu@siemens.com

Fei Li
Siemens AG
Vienna, Austria
lifei@siemens.com

Daniel Schall
Siemens AG
Vienna, Austria
daniel.schall@siemens.com

Uwe Zdun
Software Architecture
University of Vienna, Austria
uwe.zdun@univie.ac.at

ABSTRACT

Software architecture entails the making of architectural decisions based on a set of both functional and quality requirements, as well as trade-offs between them, which have to be considered to achieve design goals. Access to accumulated and documented architectural knowledge facilitates this process. In this paper, we present a set of tools that support creative decision making in the different stages an architecture specification goes through. These tools are structured around a central repository, where acquired knowledge is stored for reuse. The approach is motivated by the challenges arising from the particular needs of the software ecosystem environment, where the software design process is characterized by the participation of multiple and diverse stakeholders and the existence of multiple software applications built on a common platform. Our aim is to provide tool support for making quality-driven design decisions in a flexible and reusable manner, facilitating the system's evolvability, as well as enhancing its understandability to the stakeholders involved.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Computer-aided software engineering; D.2.11 [**Software Engineering**]: Software Architectures

General Terms

Design, Documentation, Measurement

1. INTRODUCTION

Software architecture can be considered as the collection of key decisions on the design of a software system [9]. Such architectural design decisions (ADDs) are driven by various influences that converge throughout the development process, from the target system requirements to unquantifiable factors such as the current technical or business environment, and the architect's own experience [1]. System requirements are typically divided into functional and non-functional requirements (NFRs); the latter reflect specific quality attributes (QAs) [1]. A decision to implement a specific functionality may often impact the system's QAs, so that trade-offs between functional requirements and QAs have to be considered.

Architectural knowledge is codified and made available for reuse through the use of architectural and design patterns (see for example [6, 4]) or tactics [1], which address specific QA concerns. A pattern might realize several tactics, and the implementation of any single tactic will impact other tactics as well as the pattern it is included in [7]. Implementation of desired QAs in a software system is complicated by their often subjective interpretation and different evaluation by different stakeholders in different contexts.

This is particularly evident in software ecosystems, which bring together several business partners (stakeholders) developing different applications for the same platform. This particular setting poses a series of challenges [3], such as managing variants of ADDs, facilitating modularity and reusability, or increasing system understandability between stakeholders and software architects involved in the architecture specification (AS) process.

The approach in this paper is motivated by consideration of an industrial large-scale software ecosystem in the smart city domain. In particular, we propose to integrate a set of tools that support reusable quality-driven ADDs. The tools are combined to support the different stages a software AS usually goes through: *exploratory architecture*, *architecture specification*, and *architecture review*. The given 3 stages require both logical reasoning based on a systematic study of existing architectural knowledge, e.g. design patterns, tactics, QAs, etc., and analogical reasoning based on a comparison of a problem to be solved with similar problems that have been solved in the past. Therefore, our approach supports creative decision making and design together with recurring decisions. Required knowledge related to software patterns, tactics, QAs and their relationships as well as other

ECSAW '15, September 07 - 11, 2015, Dubrovnik/Cavtat, Croatia
Copyright 2015 ACM 978-1-4503-3393-1/15/09 ...$15.00.
http://dx.doi.org/10.1145/2797433.2797480.

well-documented knowledge is captured in a repository and can be reused and evolved. In addition, we propose ways to integrate acquired architectural knowledge on ADDs in our repository through the domain experts' feedback or through the semi-automatic mining of documented ADDs.

This study is organized as follows: Section 2 discusses related work. In Section 3 we describe a motivating example for our approach. In Section 4 we describe the evolutionary stages of a software AS, and in Section 5 we link these stages to our tools. In Section 6 an example on how our approach can be utilized in practice is presented. Finally, in Section 7 we discuss the implications of our approach in the process of making ADDs and in Section 8 we draw the study conclusions.

2. RELATED WORK

Several approaches have been proposed to systematize the development of software architecture. Some of them are mainly driven by quality goals, such as the Cost Benefit Analysis Method [11], the Attribute-Driven Design [2], or the Architecture Tradeoff Analysis Method [1]. These approaches however do not support either reusable ADDs or creative decision making in the different stages an AS goes through.

Other approaches focus on other aspects, such as reducing architectural knowledge vaporization [8], knowledge-sharing decisions [5], etc. However, none of these approaches provide a comprehensive computer-aided support system with enough detail to make ADDs and design an architecture.

Drawing from our previous work on the integration of reusable architectural decision making and QAs [15], this paper proposes a combination of tools from both industry (Siemens) and academia (the University of Vienna) for supporting the process of making ADDs in different stages, i.e. starting from the examination of candidate design elements like design patterns, tactics, and QAs until the generation of reusable ADD models and their evaluation.

In the context of software ecosystems, the relationship between QAs and ADDs as well as ecosystem-specific QAs like portability and scalability have been studied [10], but to the best of our knowledge, there exist no similar approaches in the context of software ecosystems that provide a creative and systematic process of generating reusable ADDs.

3. MOTIVATING EXAMPLE

The proposed approach is motivated by our experience in developing an industry-grade service delivery platform for the smart city ecosystem [14], which aims at enabling a wide range of stakeholders to collaboratively deliver smart city services, thus reducing cost and time-to-market in the service development and delivery process. At the same time, new business models have started to emerge from the collaborations between smart city stakeholders [13], which pose significant challenges to the traditional approaches to the architecting process of large-scale software.

In this context, we have observed in particular the difficulties in achieving three architectural QAs: Understandability, Reusability, and Acceptability.

Understandability in a software ecosystem goes beyond the documentation and communication of the usage of software. The stakeholders have to understand how the offerings from a third party are transferred to the chosen services, and how the changes of a third-party service impact one's ongoing business. For example, for the end users of an energy saving service in smart buildings, the choice of OEM devices, such as gateways and sensors, made by application providers, has strong impact on the scope and capability of software services offered by the application providers.

Reusability is an internal software quality that indicates the effort required to adapt and redevelop a software component for more use cases. In the architectural design process in software ecosystems, reusability means to maximize the reuse of architectural decisions that are made in different contexts with different stakeholders, to effectively document such decisions in order for them to be referenced in the future, and to facilitate the search of relevant decisions in similar contexts.

Acceptability testing is a test conducted to determine if the requirements of a specification are met. It marks the final phase of product delivery and is a well-known concept in software and other industries. However, acceptability in the context of the smart city ecosystem implies a dynamic and iterative process that concerns the views of multiple stakeholders. For example, for a domain-independent service platform, application developers are focused on the platform's scalability, availability and API usability, while the government is concerned more about its security and policy compliance.

4. CREATING ARCHITECTURE SPECIFICATIONS FOR ECOSYSTEMS

Software development processes such as the Rational Unified Process (RUP) [12] are highly iterative and incremental. In our approach we focus on creating software AS which can be coarsely related to the first two phases in the RUP process, i.e. the Inception and Elaboration phases, but before the start of the Construction phase (see [12] for more details). Please note that our approach does not depend on any specific development process though.

Focusing on the different stages a software AS usually goes through, this process can be broadly divided into the following three stages, regardless of the external requirements to the system: exploratory architecture, architecture specification, and architecture review [17].

- During the **exploratory architecture** stage, software architects create an initial architectural concept able to meet the requirements of the system. This concept is often captured in sketches and meeting minutes. To test its feasibility, a minimalist prototype implementation often accompanies the architecture concept, which is not well-described formally (i.e. in a consolidated AS document) [17]. Often there exists a solution concept document focusing on how different use cases can be implemented using different technologies (e.g. database, web server, embedded board, etc.) and high-level architectural patterns (client-server, service-oriented architecture, etc.). Exploratory architecture is an intuitive and creative stage, since architects are faced with a new problem for which they have to come up with a new solution approach.

- The exploratory architecture flows into the **architecture specification** stage in the form of architectural diagrams and draft lists containing architecturally relevant functional and non-functional requirements as well as key design decisions aimed at meeting these requirements. If a prototype has been developed, the key design decisions mainly reflect the technical solutions implemented in the prototype for different design concerns. Using these preliminary materials, an AS can be written. The AS contains the artefacts (diagrams, key design decisions, architecturally relevant requirements, etc.) created during the exploratory architecture stage, refined into a more precise form, while focusing on those design concerns which were not addressed there. These can be, for example, QAs that must be fulfilled in a deployment scenario different from the one used for the prototype. The specification also contains a specification of architectural qualities, which argues the fulfilment of all QAs by referencing those key design decisions which endorse a particular architectural quality, for all qualities being discussed.

- When a consolidated draft of the AS document is released, it enters the **architecture review** stage of our process. The purpose of reviews is to find eventual design flaws and blind spots in the AS. Architecture reviews can be formal and informal, depending on the application context. For example, safety-critical software needs to be certified according to domain-specific standards, such as ISO 26262 for automotive applications or DO-178C for airborne systems. These standards require that formal reviews be performed and review results be provably taken into consideration in the reviewed architecture. If the software does not have to be certified, informal reviews may suffice.

The three stages can be distinguished by considering the organizational interfaces that an AS passes through. An exploratory architecture usually stays within the group or department assigned to create the AS. Aspects of the exploratory architecture are most often only informally discussed with stakeholders and people from other departments. The AS is usually released for review by experts from other departments and groups as well as by stakeholders (i.e. customers). The reviewed architecture usually represents a deliverable to the customer and/or certification authority, thus exiting the boundaries of the organization.

The described process is essentially feed-forward, as the inputs from a previous stage must be generated anew in order to trigger changes into a later stage. However feedback from the implementation phase and from stakeholders and customers must be also be integrated into the architectural design. To refine the architecture during development time, the data exchange format between the tools must be additive, allowing adding and removing design decisions from a preliminary or consolidated decision model at any given time in the software development lifecycle (e.g. JSON arrays). The entry points into the 3-stage process are therefore flexible.

5. TOOL SUPPORT

The tools supporting the architectural design process must take into consideration the specificities of each stage. Tools related to the exploratory architecture stage should aim at fostering intuition and creativity, whereas tools supporting AS should be rather normative in order to facilitate a critical and rigorous evaluation of design decisions and architectural qualities.

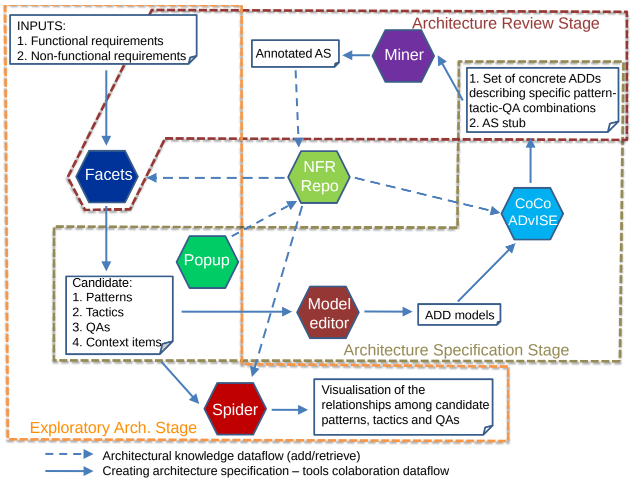

INPUTS:
1. Functional requirements
2. Non-functional requirements
Architecture Review Stage
Annotated AS
Miner
1. Set of concrete ADDs describing specific pattern-tactic-QA combinations
2. AS stub
Facets
NFR Repo
CoCo ADvISE
Popup
Candidate:
1. Patterns
2. Tactics
3. QAs
4. Context items
Model editor
ADD models
Architecture Specification Stage
Spider
Visualisation of the relationships among candidate patterns, tactics and QAs
Exploratory Arch. Stage
Architectural knowledge dataflow (add/retrieve)
Creating architecture specification – tools colaboration dataflow

Figure 1: Tools supporting ADDs in the 3 given stages

Not every project requires passing through all three architectural stages. At design time, architects can skip the exploratory architecture stage if the project does not pose considerable new challenges and decision models can be reused. This opens up the possibility of creating and applying reusable architectural models based on rigorous algorithmic reasoning. However, the inputs to these reusable models must also stem from a human-driven exploratory process. Conversely, if during the implementation phase a blocking problem is revealed in the architecture, it must be possible to return to the exploratory architecture stage and add, replace, or modify design decisions. The inputs and outputs of the following architectural stages will adapt accordingly. Consequently, preliminary and consolidated design decisions can be inferred from the new requirements driven by the stakeholders and customers or by practical implementation problems.

To support the 3-stage model we envision a chain of loosely coupled web-based tools[1] (Figure 1). The interaction between these tools is based on a simple JSON-based data format. While passing information between each other, different tools may add new items to a JSON record. This makes the resulting workflow essentially feed-forward. The interactions between different tools are stage-specific, which means that not every tool is called into action at every stage.

The NFR Engineering Repository (`NFR Repo`) is a wiki-like repository for design patterns, tactics, and QAs. The repository currently contains records for over 200 design patterns, over 50 tactics, and about 80 QAs. The records were added in the course of several months from software architecture books but also from Internet sources such as MSDN and Wikipedia. For every pattern, tactic, and QA the repository contains at least a short definition and a reference. Relations between patterns, tactics, and QAs are defined as following: each QA can be related to one/many other QAs and can be addressed by one/many design tactics. The relations between QAs, tactics, and patterns are of the type

[1]Only the `Model Editor` tool (see Figure 1) is for now not web-based.

"many-to-many" (i.e., there can be several design tactics associated with one QA and several patterns associated with one tactic and vice versa). As the relationships between QAs, tactics, and patterns as well as the reflexive relations depend on the application domain (e.g., embedded, cloud computing, automotive, etc.), relationships binding a QA-tactic-pattern set with a specific domain (decision tuples) are defined.

All the other tools use the `NFR Repo` for different purposes and tasks (see below) depending on which of the 3 stages is considered. Users of the tool set are encouraged to vote existing decision tuples up or down, thus creating the premise for more elaborate decision models based on the input of experienced software architects.

At any stage of our architecting process, the context popup form (`Popup`) can be used to add patterns, tactics, QAs, domains, context items, and associations between them to the `NFR Repo`. This can be done by visiting a website or opening a local html page and calling `Popup` from the bookmarks bar, which will bring up the popup page. The popup script will automatically parse the content of the page you are visiting to find matching QAs, tactics, and patterns from the `NFR Repo`. Having read the content of the underlying page, one is able to construct decision tuples which can be associated with this context item. One can also select an additional QA, tactic, or pattern from the drop-down list above the check box lists or add a completely new QA, tactic, or pattern to the repository. New records can be previewed before adding them to the database.

This simple way of adding new records to the `NFR Repo` enables architects to generate preliminary decision models "on the fly" without needing an exhaustive NFR repository during the exploratory architecture stage. The popup also allows for a community to form around the `NFR Repo`, with contributors from all over the organization.

5.1 Exploratory Architecture Stage

The inputs to this stage are the functional and non-functional requirements of the system. These may exist in a formally or informally written format or just as common knowledge shared by a group of developers, including emails, sketches, etc. Regardless of the format used, in this phase the requirements to the system serve the purpose of providing the architect with information about what the system has to do in a given context and domain. Using this information, the architect has to make some fundamental design decisions concerning, for example, the type of the architecture (e.g., layered, service-oriented, distributed). The output of this stage is a set of candidate patterns, tactics, QAs and context items.

The tools involved in this stage are the faceted search (`Facets`) and `Spider` tools (see Figure 1). The `Facets` tool provides an interface for the `NFR Repo` that allows a user to search the stored knowledge by selecting multiple facets: Context Types, Domains, QAs, Tactics, Patterns, and Contributor. The tool is shown in Figure 2, with the available facets on the left and the results of the search on the right. The article displayed in the frame is a context item for the decision tuples displayed below. A context item is always associated with at least one decision tuple. Users can vote these tuples up and down for the given context item or simply remove them. Decision tuples thus also contain a link to a context item (i.e., article, definition, example, etc.) and a vote count and are stored in the `NFR Repo` using the JSON format.

The `Spider` tool draws upon the `NFR Repo` to visualize all possible decision tuples. At the end of the exploratory architecture stage, it can be used to display the subset of the candidate patterns etc. selected for the next stage.

5.2 Architecture Specification Stage

Input in the AS stage consists of the set of candidate patterns, tactics, and QAs that has been determined in the exploratory architecture stage. This provides a defined decision space, wherein specific solutions can be sought depending on desired functional and quality requirements.

A model of the proposed design decision set (ADD model in Figure 1) is generated using the `Model Editor` tool. This is an Eclipse Modelling Framework-based graphical editor that allows the creation of ADD models by extending the Questions, Options, and Criteria design space analysis method [16]. In particular, for each design issue a set of questions along with potential options related to specific criteria, answers, and pattern-based solutions is specified. The information required to specify a concrete ADD model can be taken directly from the information contained in the context items of the identified candidate patterns (see [16] for more details). The generated ADD model is then used as input to the `CoCoADvISE` tool, which produces a questionnaire for making concrete design decisions. In the `CoCoADvISE` model, an individual decision consists of a design question, options, which lead to possible solutions, and QAs as decision drivers. The tool models the dependencies between each decision solution and other decisions, as well as the impact of a decision solution on QAs[15].

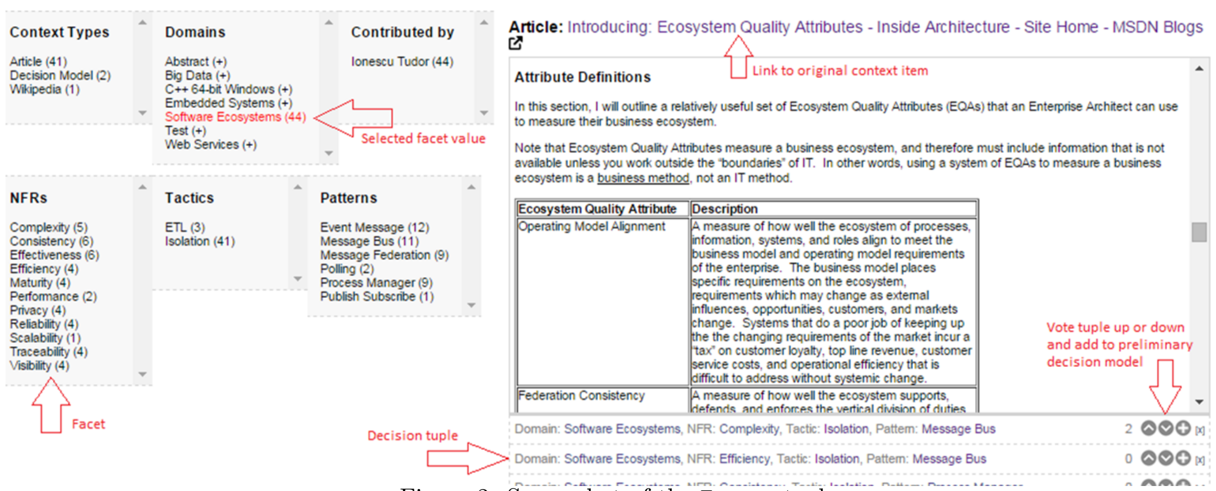
5.3 Architecture Review Stage

The review stage of the architecting process may be just as important as the previous two stages, depending on the application context. While in some cases it can be skipped, in most projects, especially those focused on safety-critical applications, architecture reviews are essential. For this reason, we envision the tool support for this stage to allow independent architects (i.e., reviewers) to assess an AS without extensive knowledge about the project. This can be accomplished by granting reviewers access to all resources used during the previous architectural stages, including requirements, preliminary and consolidated decision models, and all the tools used.

The `Facets` tool can be used by reviewers to infer alternative design decisions based on different patterns and tactics and to check the conformity of the architecture with respect to QAs. Using `Facets` and different visualizations of decision tuples, reviewers can quickly come up with alternative design decisions challenging the authors of the architecture. Once the review results are integrated into the architecture specification, a text mining tool (`Miner` in Figure 1) can be used to extract reusable design decisions and feed them back into the `NFR Repo`.

6. EMPLOYING THE TOOLS FOR THE SMART CITY ECOSYSTEM

In this section we provide a detailed example of our approach during the process of defining relevant design decisions during the exploratory architecture stage, and creating a reusable ADD model during the architecture specifica-


Selected facet value
Facet
Decision tuple
Link to original context item
Vote tuple up or down and add to preliminary decision model

Context Types
Article (41)
Decision Model (2)
Wikipedia (1)

Domains
Abstract (+)
Big Data (+)
C++ 64-bit Windows (+)
Embedded Systems (+)
Software Ecosystems (44)
Test (+)
Web Services (+)

Contributed by
Ionescu Tudor (44)

NFRs
Complexity (5)
Consistency (6)
Effectiveness (6)
Efficiency (4)
Maturity (4)
Performance (2)
Privacy (4)
Reliability (4)
Scalability (1)
Traceability (4)
Visibility (4)

Tactics
ETL (3)
Isolation (41)

Patterns
Event Message (12)
Message Bus (11)
Message Federation (9)
Polling (2)
Process Manager (9)
Publish Subscribe (1)

Article: Introducing: Ecosystem Quality Attributes - Inside Architecture - Site Home - MSDN Blogs

Attribute Definitions

In this section, I will outline a relatively useful set of Ecosystem Quality Attributes (EQAs) that an Enterprise Architect can use to measure their business ecosystem.

Note that Ecosystem Quality Attributes measure a business ecosystem, and therefore must include information that is not available unless you work outside the "boundaries" of IT. In other words, using a system of EQAs to measure a business ecosystem is a business method, not an IT method.

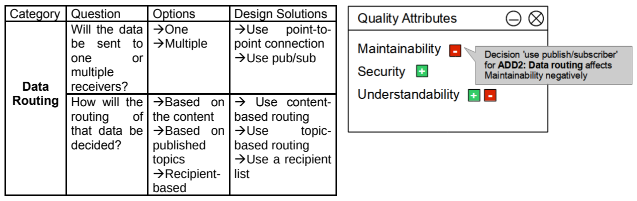
Ecosystem Quality Attribute	Description
Operating Model Alignment	A measure of how well the ecosystem of processes, information, systems, and roles align to meet the business model and operating model requirements of the enterprise. The business model places specific requirements on the ecosystem, requirements which may change as external influences, opportunities, customers, and markets change. Systems that do a poor job of keeping up the the changing requirements of the market incur a "tax" on customer loyalty, top line revenue, customer service costs, and operational efficiency that is difficult to address without systemic change.
Federation Consistency	A measure of how well the ecosystem supports, defends, and enforces the vertical division of duties

Domain: Software Ecosystems, NFR: Complexity, Tactic: Isolation, Pattern: Message Bus 2 [x]
Domain: Software Ecosystems, NFR: Efficiency, Tactic: Isolation, Pattern: Message Bus 0 [x]

Figure 2: Screenshot of the Facets tool

tion stage, drawn from the data management context in the smart city software ecosystem [15]. This ecosystem entails the input of and access to a large volume of data from a wide variety of stakeholders. This is done at different times, from different interfaces, and subject to varying policies, security concerns, etc. This results in a wide range of possible data management solutions.

The whole process started with collecting relevant use cases in the data management context. To define a set of relevant decision points, design alternatives and their solutions (e.g. design patterns, technology-related solutions), we analysed the collected use cases, investigated the existing related literature, and pursued discussions with domain experts.



Category	Question	Options	Design Solutions
Data Routing	Will the data be sent to one or multiple receivers?	→One →Multiple	→Use point-to-point connection →Use pub/sub
	How will the routing of that data be decided?	→Based on the content →Based on published topics →Recipient-based	→ Use content-based routing →Use topic-based routing →Use a recipient list

Quality Attributes
Maintainability 🟥
Security 🟩
Understandability 🟩 🟥

Decision 'use publish/subscriber' for **ADD2: Data routing** affects Maintainability negatively

Figure 3: Data Routing example for a reusable ADD model with QA evaluation

For example, for the Data Routing decision, we used `Facets` to retrieve context items and decision tuples that can be used in this context, e.g. the publish/subscribe pattern. Based on a collection of context items, we formulated a reusable ADD model for the given design decision (5.2 for more details) that is shown in Figure 3. However, the `NFR Repo` lacked associations between the publish/subscribe pattern and QAs which were found by domain experts to be relevant in the given context. These were subsequently added to the repository with the `Popup` tool. During this process we found that the choice of patterns to be included in the design solution impacted multiple QAs to various degrees. For instance, the publish/subscribe pattern assures loose coupling and thereby increases the understandability of the system. At the same time it reduces performance and increases complexity, which in turn has a negative impact on understandability. Therefore on some qualities there are positive and negative impacts at the same time. The final choice of the solution depends on a trade-off between different NFRs. By selecting options in the obtained reusable ADD model (shown in Figure 3), the recommended solutions for a concrete decision are indicated. At the same time, based on the recommended solutions, the QAs of interest were evaluated based on the relationships between the solutions and the QAs that can be found in the NFR repository (shown in Figure 3).

7. DISCUSSION

In this section we discuss the implications of our approach in the process of making ADDs in a large-scale industrial ecosystem such as the smart city. A fundamental lesson learned is that a systematic and creative approach that supports an architect in the given process is of special interest since there can exist multiple design situations for different applications in the ecosystem. Many decisions are repeatedly encountered, but new decisions can also appear. The trade-offs among different QAs and their integration in the decision-making process need to be supported. With regard to that a set of tools that support the decision-making process in its different stages is presented.

In our view, the tools described facilitate not only the work of software architects, but are also of great assistance in enhancing perception of the system for stakeholders. Thus the inclusion of context items in the `NFR Repo` provides stakeholders with practical and intuitive examples, exemplary problems, explanatory diagrams and variant use cases, enhancing understandability. Furthermore, the `Facets` tool provides a flexible way of attacking a given problem from different angles (QAs, patterns, tactics, etc.) and reviewing the architecture design space from different views, while the `Popup` tool allows "on the fly" introduction of new ideas into the repository as they occur. This enables an iterative and dynamic approach that enhances acceptability. All these tools aid stakeholders and architects in consolidating their understanding of the architecture in question and enables them to make useful contributions to the overall design process. Last but not least, the generated final AS decisions can easily be used for technical and descriptive documentation. By increasing understandability and acceptability, we consider that our approach will reduce the necessary number of iterations during the development process. We expect similar benefits during the architecture review stage, thereby reducing overall time-to-market and costs for the design.

In addition, documented decision tuples, preliminary decision models generated from `Facets`, and the generated ADD models and decisions are inherently reusable, and complement the core element of our approach, the `NFR Repo`, which contains accumulated reusable knowledge.

Newly discovered concepts and insights gained during the design process, as well as expert feedback, can be easily and directly fed back into the `NFR Repo` for future reference. Many different actors from different domains can contribute new data, new algorithms, and new applications. Therefore our approach supports the evolution of the captured knowledge that is driven by a community instead of a single architecture specification that is updated from time to time.

There exist a variety of remaining challenges in the given context that require more research. For example, the various trade-offs among QAs that need to be made during the decision making process, the different interpretation of QAs from different stakeholders, the evaluation of some QAs above others, the different impacts of ADDs on QAs, the more precise quantifications of the impacts of ADDs on QAs, etc. The issue of intellectual property rights and potential conflicts arising from the collaboration of different stakeholders also remains to be resolved. We plan to address those in our future work.

8. CONCLUSION

In this paper we present a set of tools that support reusable quality-driven architectural design decisions in the context of software ecosystems. Our approach supports creative and systematic decision making and design together with recurring decisions and aims at increasing the understanding and minimizing the required effort in the whole process. The presented tools are combined to support the different stages a software architecture specification usually goes through: exploratory architecture, architecture specification, and architecture review. The acquired architectural knowledge related to software patterns, tactics, QAs and their relationships as well as the domain experts' feedback on the documented design decisions and other well-documented knowledge is captured in a repository and can be reused and evolved.

9. REFERENCES

[1] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2 edition, 2003.

[2] L. J. Bass, M. Klein, and F. Bachmann. Quality attribute design primitives and the attribute driven design method. In *Revised Papers from the 4th International Workshop on Software Product-Family Engineering*, PFE '01, pages 169–186, London, UK, UK, 2002. Springer-Verlag.

[3] J. Bosch. From software product lines to software ecosystems. In *Proceedings of the 13th International Software Product Line Conference*, SPLC '09, pages 111–119, Pittsburgh, PA, USA, 2009. Carnegie Mellon University.

[4] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Pattern-Oriented Software Architecture, Volume 1: A System of Patterns*. Wiley, Chichester, UK, 1996.

[5] R. Farenhorst, R. Izaks, P. Lago, and H. Van Vliet. A just-in-time architectural knowledge sharing portal. In *Software Architecture, 2008. WICSA 2008. Seventh Working IEEE/IFIP Conference on*, pages 125–134, Feb 2008.

[6] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1 edition, Nov. 1994.

[7] N. B. Harrison and P. Avgeriou. How do architecture patterns and tactics interact? a model and annotation. *Journal of Systems and Software*, 83(10):1735 – 1758, 2010.

[8] N. B. Harrison, P. Avgeriou, and U. Zdun. Using patterns to capture architectural decisions. *IEEE Softw.*, 24(4):38–45, July 2007.

[9] A. Jansen and J. Bosch. Software architecture as a set of architectural design decisions. In *Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture*, WICSA '05, pages 109–120, Washington, DC, USA, 2005. IEEE Computer Society.

[10] S. Jansen. How quality attributes of software platform architectures influence software ecosystems. In *Proceedings of the 2013 International Workshop on Ecosystem Architectures*, WEA 2013, pages 6–10, New York, NY, USA, 2013. ACM.

[11] R. Kazman, J. Asundi, and M. Klein. Quantifying the costs and benefits of architectural decisions. In *Software Engineering, 2001. ICSE 2001. Proceedings of the 23rd International Conference on*, pages 297–306, May 2001.

[12] P. Kroll and P. Kruchten. *The Rational Unified Process Made Easy: A Practitioner's Guide to the RUP*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[13] F. Li, S. Qanbari, M. Voegler, and S. Dustdar. Constructing green software services: From service models to cloud-based architecture. In C. Calero and M. Piattini, editors, *Green in Software Engineering*, pages 83–104. Springer International Publishing, 2015.

[14] F. Li, M. Vogler, S. Sehic, S. Qanbari, S. Nastic, H.-L. Truong, and S. Dustdar. Web-scale service delivery for smart cities. *Internet Computing, IEEE*, 17(4):78–83, July 2013.

[15] I. Lytra, G. Engelbrecht, D. Schall, and U. Zdun. Reusable architectural decision models for quality-driven decision support: A case study from a smart cities software ecosystem. In *3rd International Workshop on Software Engineering for Systems-of-Systems (SESoS), May 2015*, May 2015.

[16] A. MacLean, R. M. Young, V. M. E. Bellotti, and T. P. Moran. Questions, options, and criteria: Elements of design space analysis. *Hum.-Comput. Interact.*, 6(3):201–250, Sept. 1991.

[17] C. Mazza, J. Fairclough, M. Bryan, P. Daniel, S. Adriaan, S. Richard, J. Michael, and G. Alvisi. *Software Engineering Guides*. Prentice-Hall International (UK), 1996.