2015 IEEE/ACM 3rd International Workshop on Software Engineering for Systems-of-Systems

Reusable Architectural Decision Models for Quality-driven Decision Support: A Case Study from a Smart Cities Software Ecosystem

Ioanna Lytra*, Gerhard Engelbrecht†, Daniel Schall† and Uwe Zdun*

*Research Group Software Architecture
University of Vienna
Vienna, Austria
{firstname.lastname}@univie.ac.at

†Siemens AG
Vienna, Austria
{firstname.lastname}@siemens.com

***Abstract*—Architectural design decisions and architectural knowledge are becoming relevant in the current practice of software architecture. In addition, reusable architectural knowledge has gained much importance in the industrial practice. In the process of architectural decision making, quality attributes constitute key drivers for designing software systems, therefore, it is important to document quality attributes along with the decisions captured. However, most of the current tools for management of architectural decisions focus mainly on capturing or sharing of design decisions. We propose to enrich a reusable architectural decision meta-model with quality attributes and introduce a corresponding tool. Our goal is to support software architects during decision making based on reusable decisions driven by quality attributes. Our approach was motivated by and applied in an industrial case study on a large-scale software ecosystem for smart cities, that constitute a complex and challenging system-of-systems domain. We applied our proposal in a few scenarios in the smart cities domain, in which the consideration of quality attributes is required to model reusable architectural knowledge adequately.**

I. Introduction

Architectural design decisions (ADDs) are regarded as first-class citizens in the documentation of software architectures [1]. Systematic decision-making approaches demand capturing and documenting multiple alternatives that have to be evaluated during the evolution of the system. However, as making decisions implies that competing requirements must be satisfied for different stakeholders' concerns, the evaluation of quality attributes (QAs) in the architecture may trigger additional decisions that must be also evaluated regarding their impacts and risks [2]. QAs, such as performance or interoperability, are commonly used in software architecture to describe the non-functional aspects of the architecture [3]. That is, many different design solutions for a functionality can be chosen, leading to different levels of the QAs and also to different trade-offs between them. For example, choosing a solution with better adaptability often leads to trade-offs in terms of performance, as the indirections necessary for enabling adaptations slow down the system. As a consequence, although many ADDs concern functionalities of the system, the QAs are often among the most important decision drivers [3]–[6].

While many approaches in the literature, such as the Architecture Tradeoff Analysis Method (ATAM) [3], the Cost Benefit Analysis Method (CBAM) [7], and the Attribute Driven Design (ADD) [8], assist architectural design and evaluation mainly driven by quality goals and scenarios, the majority of architectural decision support methods and tools focus on other aspects, such as reducing architectural knowledge (AK) vaporization [9], reusability of ADDs [4], knowledge sharing decisions [10], and group decision making (such as [11]). The subjectivity of quality concerns that can be interpreted differently for different stakeholders and in different contexts, the importance of some qualities that are evaluated above others, and the impact of QAs on the decisions made increase the complexity of AK methods and tools. For these reasons, supporting QAs entangled with ADDs is challenging and more research is needed to provide methods for capturing and documenting the quality properties along with the ADDs and the relationships among them.

For the approach presented in this paper, we derived the requirements from the needs of an industrial case study on a large-scale software ecosystem in the smart cities domain. Specifically, we propose to integrate reusable ADDs with QAs in order to provide quality-driven decision making support. This is especially important in a software ecosystem context, as ecosystems focus on a set of businesses functioning as a unit and interacting with a shared market for software and services, together with relationships among them [12]. Thus, for ecosystem decisions not only a single main development organization but many interacting players in the ecosystem must be taken into account. In addition, in a system-of-systems as a smart city, many possible applications must be considered when making ecosystem decisions. In this context, various recurring design situations need to be addressed. To tackle these problems and raise the quality of recurring decisions in the smart cities domain, we propose to use a reusable architectural decision making approach. The goal is

978-1-4673-7091-2/15 $31.00 © 2015 IEEE
DOI 10.1109/SESoS.2015.14

to base the ecosystem decisions on established AK, such as existing software patterns [13] or other well-documented AK, in order to address the broad nature of ecosystem decisions and support reuse of knowledge. We further propose, as our main contribution, to integrate reusable ADDs with QAs to enable quality-driven decision support for recurring design problems at hand. In particular, we model ADDs similar to the Questions, Options, and Criteria approach [14] along with their impact – e.g., positive or negative – on QAs. In addition to the approach itself, we present its application in the smart cities ecosystem case study [15] and discuss the lessons learned from that case.

The remainder of the paper is structured as follows. In Section II, we present our case study on smart city software ecosystems. We get into the details of our approach and accompanying tool in Section III and then present how we applied our approach in the case study in Section IV. We discuss lessons learned, limitations, and challenges in Section V. In Section VI, we discuss the related work and, finally, we draw conclusions and outline future work in Section VII.

II. Smart Cities Ecosystem Case Study

We motivate our approach by introducing our case study on software ecosystems in a complex system-of-system, namely the smart cities domain. Software ecosystems are defined as a set of businesses functioning as a unit and interacting with a shared market for software and services, together with relationships among them [12]. The relationships are mainly realized through the exchange of information, resources, and artifacts. Software ecosystems entail multiple product developments relying on a common architecture platform and allow companies to make a platform available to other parties outside its organizational boundary. This motivates also the decision of our industrial partner to develop a software ecosystem for the smart cities domain.

Our case study concerns a system that is being developed at Siemens with the purpose to provide a platform for a large-scale software ecosystem for various smart cities projects with partner organizations inside and outside of Siemens. From an industry point of view, one motivation for the smart city movement is the need to run urban infrastructures such as the electrical grid or water distribution systems more efficiently, effectively, cleaner and more secure. Siemens has various initiatives in the context of smart cities. To name two initiatives, the sustainable cities program developing the city intelligence platform and the smart city living lab in Vienna, Seestadt Aspern, have both their headquarters in Vienna, Austria. At the heart of each of those two initiatives, smart city ICT platforms enable the realization of various novel applications. The idea of these platforms is to provide a software ecosystem for smart city applications (so-called apps). Apps can be implemented by Siemens internal developers or by external partners using Web APIs. From an architecture point of view, the platform layers (see Figure 1) include:

- *Data Integration.* Extract Transform Load (ETL) is performed to obtain data from external systems and to load data into an integrated data schema. External systems include meter data management, water asset management, and building energy monitoring and controlling, deliver data in various formats, and require flexible and robust integration techniques.
- *Messaging.* The messaging layer enables the integration of real time data. Whereas ETL performs batch import of data in mostly fixed time intervals, the messaging layer follows an event driven approach following a pub/sub mechanism. The data integration choice (ETL vs. message driven) depends on the capabilities and interfaces of the external system and the application requirements.
- *Data Storage.* Within the database/data-warehouse layer, data is stored in a persistent manner for querying and analysis. Both, SQL-based and NoSQL database technologies are supported.
- *Business Logic.* On top of the data storage layer, a business logic layer provides APIs to access data structures. In addition, an analytics runtime and modules provide capabilities to implement data mining algorithms.
- *Services.* Web APIs provide access to data and functionality in a standardized manner (RESTful services with JSON/XML data support).
- *GUI.* Web dashboards provide the means for visualization and presentation of results to a variety of users. A dashboard is configurable to provide visualizations depending on the information need of a user.

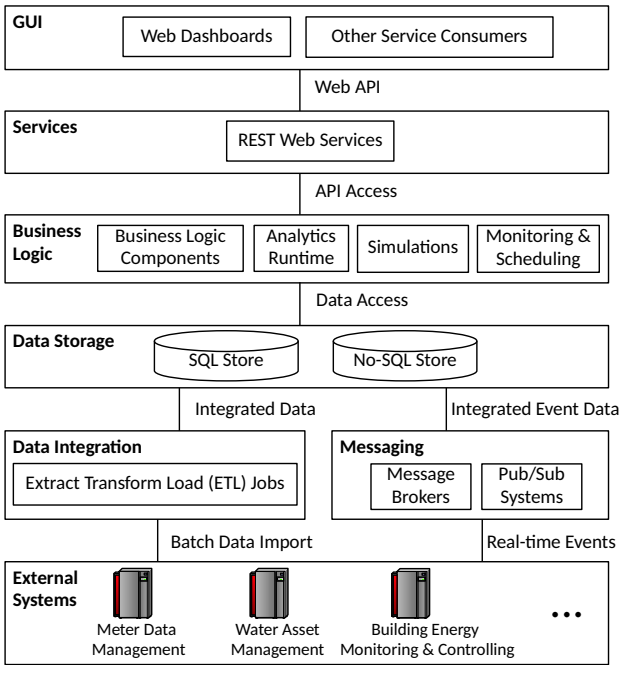


Figure 1. Smart City Case Study

The platform offers various points for extensibility in each layer so that new applications and new smart city deployments can be realized. The different design strategies and tactics are

guided by both functional requirements and QAs. The QAs depend on the specific applications to be realized and on the various smart city stakeholders. Some possible extension scenarios and their potential impact on QAs are:

- ETL jobs can be implemented and automated to load data into the platform. Data quality, data completeness, and reliability must be considered when designing new jobs.
- Subscriptions to event driven data sources can be added within the messaging layer. Here, performance and scalability play an important role.
- The data model can be extended by new entities to accommodate new data sources. Maintainability of the overall data model needs to be ensured.
- New APIs in the business layer and new data mining algorithms can be added. Maintainability of the APIs is important, as well as performance of the data mining algorithms.
- New Web APIs and services can be implemented. Here, security, performance, and reliability are important.
- Configuration of new dashboards can be performed by users. Here, usability and privacy issues play a role.

The question about which QAs are important and their priorities depend on the needs of the stakeholders. Various stakeholders are involved in different parts and activities of a smart city ecosystem. For instance, *Data Providers* provide data stores of information that is being monitored (i.e., organization, structuring, and delivery of information), *Facility Operators* are the main users of the smart city system that processes inputs from heterogeneous sources (e.g., smart grid networks, mobile devices, etc.), and *Application Developers* are responsible for the visualization, analysis and generation of reports summarizing information from different resources for which *Smart City Stakeholders* and *Smart Citizens* are the main users.

According to their needs, the various stakeholders will access data at different times, from different data sources, and using different interfaces or protocols. In addition, policies, security issues, and business rules must be considered during data access and processing. For instance, *Facility Operators* may have JDBC access directly to the database(s), while internal *Application Developers* are allowed to use Object-Relational Mapping (ORM) APIs, and external *Application Developers* have access through a REST API to the corresponding applications. These three technology options, however, impose quite different performance characteristics on the applications that use them and the analyses they are able to perform. The *Facility Operators* may also have access to building control systems (e.g., heating, air-conditioning) using BACnet, a data communication protocol for building automation. Sensitive data should not be provided to any of the stakeholders unless it is anonymized. For the *Smart City Stakeholders* and *Smart Citizens* only aggregated data is provided based on privacy regulations. While some data is publicly available (such as traffic information or aggregated information offered by the city), encryption is required for personal data such as a citizen's energy consumption.

As a result, a software architect working on the design of such complex systems-of-systems as the smart cities is typically confronted with a set of complex design decisions on how to realize application requirements for the corresponding software ecosystems without violating QAs. This is a nontrivial task given the multitude of extension points, the number of different participating applications and stakeholders, and the range of QAs to be considered in that context. Our main contribution here is to propose quality-driven reusable architectural decision making support, in order to base the ecosystem decisions on established AK, such as existing software patterns or other well-documented knowledge, integrated with QAs.

III. CoCoADvISE Approach and Tool Support

CoCoADvISE is an approach and a corresponding tool for supporting architectural decision making and documentation based on reusable ADDs. Although the introduced decision meta-model is inspired by the Questions, Options, and Criteria (QOC) approach [14] the CoCoADvISE meta-model is rather an extension than an application of QOC. The CoCoADvISE meta-model includes among others dependencies between options, options and decisions, and options and questions, as well as a categorization of reusable solutions (options in QOC). To support, additionally, QAs during decision making, we have extended the underlying meta-model to include relationships between design solutions and QAs, as well as interrelationships among QAs.

A. CoCoADvISE Meta-model Extension

The CoCoADvISE decision meta-model (see Figure 2) consists mainly of *Decisions*, *Questions*, *Options*, *Solutions*, *Decision Drivers*, and various relationships among them. For each design issue, a set of *Questions* providing multiple *Options* have to be modeled. Examples of relationships are that a selection of an *Option* triggers a next *Decision* or an *Option* is incompatible with or enforces another *Option*. A selection of an *Option* may lead to a suggestion of a *Solution* for the software architect. This *Solution* will be applied using an *Architecture Guidance*, that is, for instance, a *Design Pattern* or *Architectural Pattern*. In addition, the CoCoADvISE ADD meta-model contains explicit links from *Solutions* to *Decision Drivers*, an abstraction for *Quality Attributes*. Thus, a *Solution* can have or erase an *Impact* (e.g., positive or negative) on a *Decision Driver*. The same relationship applies alternatively for an *Architecture Guidance*. A *Decision Driver* can finally affect positively (be in synergy with) or negatively (be in contradiction with) other *Decision Drivers*.

B. Quality-driven Decision Support

The advantage of the CoCoADvISE reusable decision models is that they need to be created only once for a recurring design situation. In similar application contexts, corresponding questionnaires can be automatically instantiated and used for making concrete decisions. Based on the outcomes of the

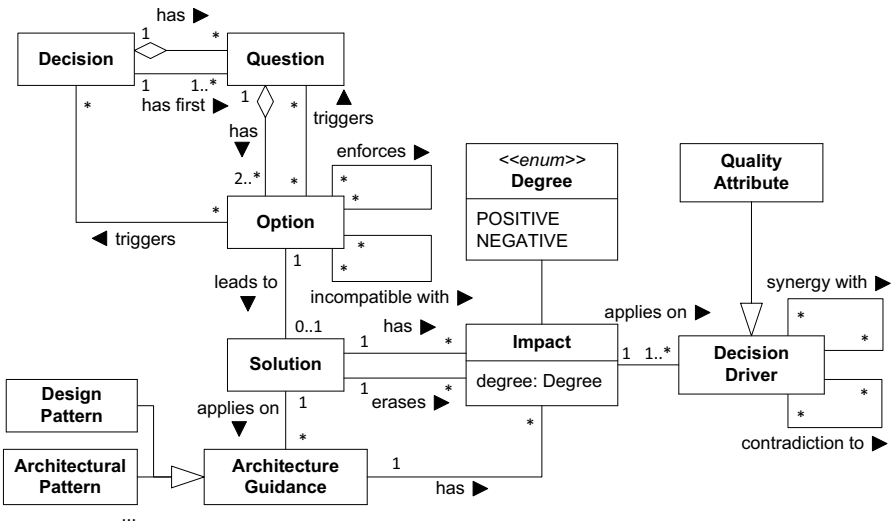


Figure 2. Reusable Architectural Decision Meta-model

questionnaires answered by software architects through the decision making process, CoCoADvISE can automatically resolve potential constraints and dependencies (e.g., reveal follow-on questions and decisions, deactivate options, etc.), recommend best-fitting design solutions, and visualize the impact of recommended solutions on QAs of interest.

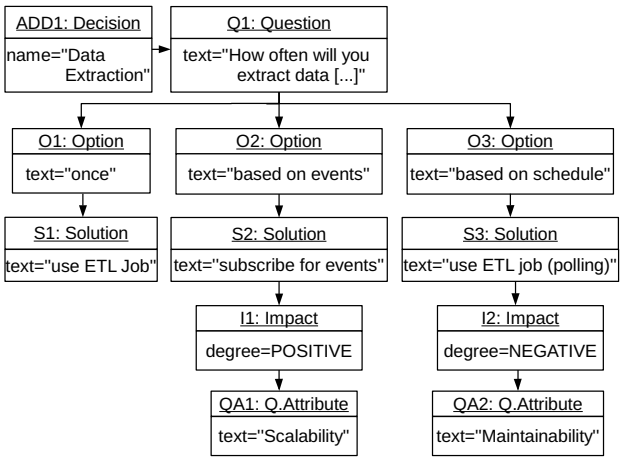


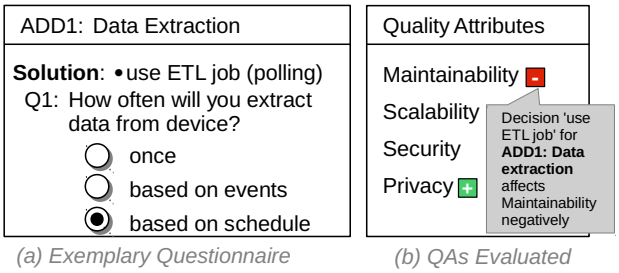
Figure 3. Exemplary Architectural Decision Model

In Figure 3, we give an excerpt of a reusable decision model for data management, consisting of one Decision (i.e., "Data Extraction"), related to one Question (i.e., "How often will you extract data from device?"), providing three Options: (a) once, (b) based on events, and (c) based on schedule. The selection of each option leads to a different solution, that is, if we need to extract data only once or periodically we may use an ETL job, and consider the polling mechanism if we extract data based on a schedule; if data is extracted from device once it is available, we will implement publish-subscriber and need to subscribe for new events at the corresponding device. ETL jobs are rather complex to program and introduce maintenance costs, especially when business rules change over time and data quality varies. An advantage of publish-subscriber is that it offers high scalability when the number of subscribers increases. Thus, modeling these relationships means connecting the solutions "subscribe for events" and "use ETL job (polling)" to the QAs "Maintainability" and "Scalability" with NEGATIVE and POSITIVE impact respectively.

From such a reusable decision model, software architects can instantiate questionnaires many times in similar design situations. By selecting options the recommended solutions are indicated for a concrete decision as soon as they are applicable. For instance, the selection of the third option of the question of Figure 4(a) will reveal the recommended solution "use ETL job (polling)". At the same time, the QAs of interest are evaluated according to the recommended solutions and based on predefined relationships between e.g., the solutions and the QAs (see Figure 4(b)). Plus (+) indicates positive effect, while minus (-) indicates negative effect of a specific decision on the QA. Thus, software architects receive guidance with respect to two concerns: (a) how are the QAs affected by their (recommended) design solutions and (b) what is the rationale behind positively or negatively evaluated QAs. The first is supported in the tool by the visualization of the impact of the design options on the QAs with plus (+) and minus (-) "votes" while the second is achieved by providing related tooltip information (e.g., *Decision "use ETL job" for ADD1: Data extraction affects Maintainability negatively*).

A live demo of the CoCoADvISE tool with the setting presented in the smart cities case study (see Section IV) is available at https://andromeda.swa.univie.ac.at/cocoadvise/[1].

[1]Username: "test" (no password is required).



(a) Exemplary Questionnaire (b) QAs Evaluated

Figure 4. Exemplary CoCoADvISE Questionnaire and Affected QAs

IV. Case Study

In this section, we elaborate on using our approach in a case study from the smart cities domain. In particular, we demonstrate how to model data management related ADDs in software ecosystems for smart cities in a reusable decision model. Afterwards, we model the relationships between the various design solutions and QAs, that is, for a set of QAs of interest we investigate and capture the positive or negative effect of the corresponding ADDs on them.

Two researchers and two domain experts in smart cities design were involved in the case study, in three phases and multiple refinements for each phase. First of all, a set of use cases with respect to data management in systems for smart cities were collected and analyzed. Based on these use cases, we defined related decision points, options, and design alternatives, as well as related reusable AK (e.g., design patterns, technology-related solutions, etc.) – based mainly on the existing related literature and discussions with the domain experts – and organized them in categories of ADDs that need to be made in the context of data management. Afterwards, this information was organized in a reusable decision CoCoADvISE model. Table I provides exemplary decisions of the reusable decision model consolidated in the first step of the case study design, consisting of data management related decisions, divided in six categories (*Data Extraction*, *Data Processing*, *Data Routing*, *Data Storage*, *Data Presentation*, and *Data Privacy*).

In the second step of the case study, we collected QAs that are considered, according to the domain experts, key QAs for the design of the smart city ecosystem. For instance, *Scalability* is important in this context as distributed systems need to be accessed and big amounts of data need to be managed at real-time. In addition, *Privacy* and *Security* are key concerns whenever sensitive personalized data is being requested. Other QAs like *Performance*, *Reliability*, and *Maintainability* influence strongly the architects' ADDs for the smart cities software ecosystem as well. The final list of considered QAs consists of 11 QAs: Availability, Data Completeness, Data Quality, Extensibility, Maintainability, Performance, Privacy, Reliability, Security, Scalability, and Usability.

Often, in architectural decision making, we need to deal with competing requirements and therefore, these QA interdependencies also need to be considered when making trade-offs [16]; for instance, security comes usually at costs of usability. Using the CoCoADvISE meta-model we expressed such relationships between QAs that occurred in the decisions, i.e., a QA is in *synergy with* or in *contradiction with* another QA. Once a constraint is in place, the tool automatically checks it during decision making.

After eliciting a list of related QAs, we investigated whether and to what extent the design solutions of the reusable decision model affect these QAs. Of course, not all QAs are relevant for every design solution and not all design solutions are related to one of the aforementioned QAs. The outcome of this step was a list of design solutions along with their positive or negative impact on the QAs. A number of prominent examples of such impacts in our reusable decision model for data management are shown in Table II.

Table II
Examples of Impacts of Design Solutions on QAs

Design Solution	*Impact on QA* positive	negative
Configurable dashboard	Usability	-
Batch requests	Usability, Performance	-
Vertical/horizontal scaling	Scalability	-
Offline algorithms	Performance	-
Anonymization	Security, Privacy	Performance
Cache data	Performance	-
Publish-subscriber	-	Security

The last step of our case study was to document ADDs for the use case scenarios collected in the first step, as instances of the reusable architectural decision model for data management. That is, we tested our model by designing a number of app-level architectures.

V. Discussion

A central learned lesson of our work is that a systematic approach to deal with the software architecture and architectural qualities is specifically important in the context of a software ecosystem for a large-scale system-of-system, such as the smart city. As many decisions are taken over and over again for multiple design situations in different applications, we decided to use reusable decisions as a basis for our systematic approach. As this approach had not yet been integrated with QAs, we proposed – to the best of our knowledge – the first systematic approach for integrating reusable architectural decision models and QAs.

Basing decision for multiple applications – whose requirements are often unknown at design time of the ecosystem platform – only on past experiences is not enough, but knowledge reuse from both internal experiences and external sources is needed. External sources are mainly the literature and the Web, including design and architecture patterns with their quality impacts documented as pattern consequences or other discussions of established design or architecture solutions with clear statements on the impacts on QAs.

From an architecture point of view, one of the biggest challenges in a smart city software ecosystem is the realization of new applications and extensions of the platform given the

Table I
EXEMPLARY QUESTIONS OF REUSABLE ARCHITECTURAL DECISION MODEL FOR DATA MANAGEMENT IN SOFTWARE ECOSYSTEMS

Category	Question	Options	Design Solutions
Data Extraction	How often will you extract data from device?	• Once • Based on events • Based on schedule	→ Use ETL Job → Subscribe to device for events (publish-subscriber) → Use ETL Job (with polling)
Data Processing	Do you need to translate extracted data (from different sources) in a common format?	• No • Yes	- → Use a normalizer/canonical model for defining a common format
Data Routing	Will the data be sent to one or multiple receivers?	• One • Multiple	→ Use point-to-point connection → Use publish-subscriber
Data Storage	How would you characterize the size of data that needs to be stored?	• Small/medium • Big	→ Scaling not required → Use vertical (add resources to a node) or horizontal (add nodes to a system) scaling
Data Presentation	Where will the data be persisted? How will the data be visualized for the end user?	• File • Standard UI • Configurable	→ Store in file system → Non-customizable UI → Use a configurable dashboard
Data Privacy	Will you need to anonymize data?	• No • Yes	→ - → Use data filter + aggregator

wide variety of design choices and available design patterns. Each choice may have different implications with respect to QAs. QAs also need to be analyzed and specified for the concrete domain they apply to. Without a systematic decision process as presented in this work, software architects need strong guidance by senior software architects to make the right decisions. Here, the reusable decision process helps to solve this problem.

Another challenge for software architects is the deep domain knowledge needed to realize smart city applications. Each domain may have its own specific protocols, standards, data formats, etc. that need to be considered when designing an application. The advantage of the presented approach is that guidance tailored to a specific domain can be given. Individual steps in the overall decision model can be customized for specific domains (e.g., using a particular data format in the context of a smart grid or smart building domain). Thus, software architectures are guided through domain specific design decisions and do not need to seek help and advice from domain experts.

Finally, the reusable decision model can evolve as the body of AK evolves. Within a smart city software ecosystem, many different actors from different domains contribute new data, new algorithms, and new applications. Lessons learned on how to solve a given architectural problem or the most efficient way to solve an algorithmic problem can be added to the model. Thus, the model and the knowledge are driven by a community instead of a single architecture specification that is updated from time to time.

In the elaboration of our case study, we presented, a reusable decision model focusing on data management issues, however, we have been working on gathering other reusable AK, e.g., related to service-based integration [6] that can also be adapted to the smart cities domain. In addition, our approach has only been tested in the context of the smart cities domain. However, we believe it can be generalized to many different contexts. In general, using our approach requires a certain system complexity. For very simplistic domains or small-scale systems too much upfront investment might be required. The extra effort required for our approach is probably only justified from a certain system size on and with a certain level of reuse. The break-even point for this would need to be determined in future research. Apart from these effort aspects, we consider our approach to be applicable also in other domains and for other system sizes. The extra effort for our approach is mainly the upfront investment of (a) building a design space with quality annotations for the target domain and (b) tailoring it to the domain of the ecosystem. Both efforts are considerable, especially if detailed data on the effect on QAs is missing in the literature. But as the design space needs to be built only once and can then be reused, it was acceptable in our case study's context.

The various trade-offs of QAs that need to be made during the decision making process still remains a complex and challenging task. The subjectivity of quality concerns that can be interpreted differently for different stakeholders and in different contexts, the importance of some qualities that are evaluated above others, the different levels of impact on QAs, and so on, increase the complexity of architectural decision making driven by QAs. More research on automation regarding QA-based decisions is needed, but this would require detailed data on how specific QAs are influenced by different reusable decision options. So far the documented knowledge in the literature is mainly anecdotal and informal and more efforts from the community and practitioners are required to completely integrate reusable ADDs with QAs.

VI. RELATED WORK

In the context of software ecosystems, the design choices the software architects make have been studied with respect to their QAs [17]. Some important quality properties of software ecosystems, like portability, openness, and scalability (see [17]) as well as design decisions [18] have been investigated for the ecosystem architecture as a whole. That means,

that these existing works have documented design decisions and QAs in a form of best practices, but have not introduced guidance for concrete (reusable or not) design decisions. Also, the concept of reusable decision models, although it has been used in other domains (e.g., SOA solutions [19]), it has never been considered in the context of software ecosystems.

For the software architecture community, capturing ADDs is important for analyzing and understanding the rationale and implications of these decisions and for gathering AK. A substantial amount of work has been done in the direction of documenting the AK using architectural decision modeling. The CoCoADvISE meta-model we introduced in this paper has many similarities to other decision meta-models (such as [4]), however, it can be used additionally to model the impact of ADDs on QAs.

Although many approaches in the literature, such as the Architecture Tradeoff Analysis Method (ATAM) [3], the Cost Benefit Analysis Method (CBAM) [7], and the Attribute Driven Design (ADD) [8], assist architectural design and evaluation mainly driven by quality goals and scenarios, the majority of architectural decision support methods and tools have in most cases different focus [20]. They aim, for instance, at reducing AK vaporization [9], reusability of architectural design decisions [4], knowledge sharing decisions [10], and group decision making [11]. Architecture patterns have been systematically integrated with QAs to enhance the usability of patterns [21]. However, to the best of our knowledge, our proposal is the first one to relate ADDs in reusable architectural decision models with QAs, in order to provide quality-based decision support.

VII. Conclusions and Future Work

In this work, we proposed to integrate QAs with reusable ADDs to support architectural decision making. For this, we introduced a reusable decision meta-model for modeling among others the impact of design solutions on QAs, as well as a tool prototype for supporting architectural decision making driven by QAs. We applied afterwards our proposal in a large-scale software ecosystem in the smart cities domain. Our experiences show that reusable architectural decisions can help to systematize the decision making process in ecosystem contexts and that enriching them with QA support enables us to model many relevant QAs that must be considered during the decisions already at the time when the reusable decision model is created. Still many challenges regarding quality-driven decision support based on reusable AK need to be addressed, as discussed in Section V. Essentially, part of our upcoming work is to model and use further reusable decision models for other aspects than data management in the context of the smart cities software ecosystems. The goal of such reusable architectural decision models is to be used, updated, and configured among various projects, in similar domains, and even across development group, department, and company boundaries. As the next step, the decision model(s) and the Web-based tool will be tested by an entire community of Siemens software architects.

References

[1] A. Jansen and J. Bosch, "Software Architecture as a Set of Architectural Design Decisions," in *5th Working IEEE/IFIP Conf. on Software Architecture (WICSA), Pittsburgh, PA, USA*. IEEE Computer Society, 2005, pp. 109–120.

[2] F. Bachmann, L. Bass, M. Klein, and C. Shelton, "Designing Software Architectures to Achieve Quality Attribute Requirements," *Software, IEEE Proc.*, vol. 152, no. 4, pp. 153–165, Aug. 2005.

[3] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*. Addison-Wesley Professional, 2003.

[4] O. Zimmermann, T. Gschwind, J. Küster, F. Leymann, and N. Schuster, "Reusable Architectural Decision Models for Enterprise Application Development," in *3rd Int'l Conf. on Quality of Software Architectures (QoSA), Medford, MA, USA*. Springer, 2007, pp. 15–32.

[5] O. Zimmermann, U. Zdun, T. Gschwind, and F. Leymann, "Combining Pattern Languages and Reusable Architectural Decision Models into a Comprehensive and Comprehensible Design Method," in *7th IEEE/IFIP Conf. on Software Architecture (WICSA)*. IEEE, 2008, pp. 157–166.

[6] I. Lytra, S. Sobernig, and U. Zdun, "Architectural Decision Making for Service-Based Platform Integration: A Qualitative Multi-Method Study," in *Joint 10th Working IEEE/IFIP Conf. on Software Architecture & 6th European Conf. on Software Architecture (WICSA/ECSA), Helsinki, Finland*. IEEE Computer Society, 2012.

[7] R. Kazman, J. Asundi, and M. Klein, "Quantifying the Costs and Benefits of Architectural Decisions," in *23rd Int'l Conf. on Software Engineering (ICSE)*, 2001, pp. 297–306.

[8] L. J. Bass, M. Klein, and F. Bachmann, "Quality Attribute Design Primitives and the Attribute Driven Design Method," in *Revised Papers from the 4th Int'l Workshop on Software Product-Family Engineering*, ser. PFE'01. London, UK: Springer-Verlag, 2002, pp. 169–186.

[9] N. B. Harrison, P. Avgeriou, and U. Zdun, "Using Patterns to Capture Architectural Decisions," *IEEE Software*, vol. 24, no. 4, pp. 38–45, 2007.

[10] R. Farenhorst, R. Izaks, P. Lago, and H. Van Vliet, "A Just-In-Time Architectural Knowledge Sharing Portal," in *Seventh Working IEEE/IFIP Conf. on Software Architecture (WICSA)*, Feb 2008, pp. 125–134.

[11] M. Nowak and C. Pautasso, "Team Situational Awareness and Architectural Decision Making with the Software Architecture Warehouse," in *Proc. of the 7th European Conf. on Software Architecture*, ser. ECSA'13. Berlin, Heidelberg: Springer-Verlag, 2013, pp. 146–161.

[12] D. G. Messerschmitt and C. Szyperski, *Software Ecosystem: Understanding an Indispensable Technology and Industry*. Cambridge, MA, USA: MIT Press, 2003.

[13] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Pattern-Oriented Software Architecture Volume 1: A System of Patterns*. John Wiley & Sons, 1996, vol. 1.

[14] A. MacLean, R. Young, V. Bellotti, and T. Moran, "Questions, Options, and Criteria: Elements of Design Space Analysis," *Human-Computer Interaction*, vol. 6, pp. 201–250, 1991.

[15] C. B. Seaman, "Qualitative methods in empirical studies of software engineering," *IEEE Trans. Softw. Eng.*, vol. 25, no. 4, pp. 557–572, Jul. 1999.

[16] A. Egyed and P. Grünbacher, "Identifying Requirements Conflicts and Cooperation: How Quality Attributes and Automated Traceability Can Help," *IEEE Software*, vol. 21, no. 6, pp. 50–58, 2004.

[17] S. Jansen, "How Quality Attributes of Software Platform Architectures Influence Software Ecosystems," in *Proc. of the 2013 Int'l Workshop on Ecosystem Architectures*, ser. WEA 2013. New York, NY, USA: ACM, 2013, pp. 6–10.

[18] M. Che and D. E. Perry, "Architectural Design Decisions in Open Software Development: A Transition to Software Ecosystems," in *23rd Australian Software Engineering Conf. (ASWEC)*, 2014, pp. 58–61.

[19] O. Zimmermann, J. Koehler, and L. Frank, "Architectural Decision Models as Micro-Methodology for Service-Oriented Analysis and Design," in *Proceedings of the Workshop on Software Engineering Methods for Service-oriented Architecture (SEMSOA), Hannover, Germany*, D. Lübke, Ed., May 2007, pp. 46–60.

[20] M. Shahin, P. Liang, and M. R. Khayyambashi, "Architectural design decision: Existing models and tools," in *IEEE/IFIP Conf. on Software Architecture/European Conf. on Software Architecture (WICSA/ECSA)*. IEEE, 2009, pp. 293–296.

[21] N. B. Harrison and P. Avgeriou, "Leveraging Architecture Patterns to Satisfy Quality Attributes," in *First European Conf. on Software Architecture (ECSA)*. Springer, 2007, pp. 263–270.