# Supporting Quality-Driven Architectural Design Decisions in Software Ecosystems

Srdjan Stevanetic[1], Konstantinos Plakidas[2], Tudor B. Ionescu[1], Daniel Schall[1], and Uwe Zdun[2]

[1]Siemens AG , Vienna, Austria , name.surname@siemens.com

[2]Software Architecture Research Group , University of Vienna, Vienna, Austria , name.surname@univie.ac.at

## ABSTRACT

System quality attributes (QAs) are often considered as the most important decision drivers. In this paper, motivated by the decision making in a smart-city software ecosystem, we extend our previous approach that integrates reusable architectural design decisions (ADDs) with the QAs, by integrating tactics that support quality-driven decision making. In addition, we present an approach that enables system evolution, based on controlled and adaptable decision making and utilizing real data obtained during system monitoring. The approach integrates the previous approach that uses tactics with the existing model-driven development paradigm and the corresponding tools.

## 1. INTRODUCTION

Architectural design decisions (ADDs) are often considered to be the most important part in documenting software architectures [10]. ADDs are usually driven by several influencing factors including the system requirements, current technical and business environment, and the architect's experiences [1]. By the system requirements we refer to both functional and non-functional requirements [1]. The non-functional requirements reflect specific system QAs [1]. Implementing a specific system functionality might often influence the QAs that can further influence other QAs. For example, implementing a solution that enables authorisation or adaptability might negatively influence performance. Therefore, even though many ADDs concern functionalities of the system, the QAs are often considered as the most important decision drivers [20, 2].

Even though several approaches in the literature, such as the Architecture Trade-off Analysis Method [2], the Cost Benefit Analysis Method [11], and the Attribute Driven Design [3], aim to support architectural design and evaluation driven by quality goals and scenarios, the majority of ADD methods and tools consider other aspects, such as reducing architectural knowledge (AK) vaporization [9], reusability of ADDs [20], and knowledge sharing decisions [6]. Methods for enabling systematic and efficient quality-driven decision-making process are still missing. Creating such methods is very challenging because of different reasons: the exact evaluation of the QAs at the early stage of development such as making ADDs is difficult, making ADDs often implies dealing with competing requirements that must be satisfied for different stakeholders' concerns, different stakeholders often interpret the QAs in a different way depending on the context, some QAs need to be evaluated above others, etc.

In our previous work [14], we proposed to integrate reusable ADDs with the QAs in order to enable quality-driven decision support for recurring design problems. In this paper, we propose to extend the previous work by integrating tactics as design decisions that improve the individual QAs. In addition, we propose an approach that supports automated system evolution based on controlled and adaptable quality-driven decision making process. In particular, the impacts of the selected design solutions on the QAs are measured using real data obtained from both design-time and run-time. The obtained measures are then fed back to the decision making process in order to enable controlled and adaptable quality-driven decision-making. The approach is based on the model-driven development paradigm.

This paper is structured as follows. In Section 2, we describe the motivation for our work. In Section 3, we present the details of our approach for quality-driven decision making using tactics. In Section 4, we present the details of our approach for controlled and adaptable quality-driven decision making. In Section 5, an example of a reusable architectural decision model that uses tactics is presented and, finally, we draw conclusions and outline future work in Section 6.

## 2. MOTIVATION

The approach presented in this paper is derived from the needs of an industrial case study on a large-scale smart cities software ecosystem. A smart city ecosystem is a huge information system comprising of several smaller communicating components. Smart city components are integrated using service-oriented architecture that provides interoperability among different platforms by supporting modular design and software reuse and integration. Different services such as healthcare, education, transportation, water supply networks, waste management, etc. need to be integrated with each other. Because of the large diversity of many interacting players in a smart city, many applications that need to considered during decision making, and changes that may appear in all constituting components, various recurring design situations need to be addressed. Therefore, in a smart city ecosystem, the quality-driven decision making support is of crucial importance.

## 3. META-MODEL EXTENSION

In our previous work [14], we proposed an approach and a corresponding tool called CoCoADvISE that supports architectural decision making and documentation base on reusable ADDs. We modelled ADDs using the Questions, Options, and Criteria approach [16]. The result of a made ADD is captured by the established AK, such as existing software patterns [5] or other well-documented AK, which impact on the QAs is evaluated as positive or negative. In this paper, we extend the CoCoADvISE meta-model with tactics



*ECSAW '16, November 28-December 02, 2016, Copenhagen, Denmark*



DOI: http://dx.doi.org/10.1145/2993412.3003383

to further support systematic quality-driven decision making. The integration of tactics is pursued by systematically studying the existing work on the interactions of tactics with other related artefacts (i.e. software patterns, QAs, and other relevant AK). In the text below, we refer to the existing related work and explain our approach in more detail.

Tactics are design decisions that specifically concern QAs [2]. Each tactic aims at improving one QA but it can also have side effects, usually negative, on other QAs. Tactics concerning the same thing are grouped into categories which are called design concerns. For example, the group of tactics for security includes resisting attacks, detecting attacks, and recovering from attacks. The resisting attacks category contains the following tactics: authenticate users, authorize users, maintain data confidentiality, maintain integrity, limit exposure, and limit access. Tactics can be design-time or run-time [2]. Design-time tactics represent overall approaches for design and implementation, such as hide information to improve modifiability. Run-time tactics represent approaches that refer to the specific aspects of the QAs, such as ping-echo for improving the reliability. In this paper, we refer to run-time tactics because design-time tactics mostly cut across all design partitions and are implemented implicitly in the code [2]. Therefore, they are not as suitable for modelling at the architecture level as run-time tactics which have well-defined effects with patterns (see below for details).

The impact of tactics on architectural patterns has been studied by several authors (see for example [7, 8]). They have found that tactics affect patterns in different ways. In some cases, a tactic is compatible with a pattern, meaning that it can be easily integrated in the pattern's structure and has compatible behaviour. In other cases, a tactic may require significant changes to the pattern's structure and behaviour. Harrison and Avgeriou [7] have found the following types of changes that describe how difficult is to implement a tactic in some pattern:

- **Implemented in:** implemented within a pattern component with minor changes
- **Replicates:** a pattern component is duplicated
- **Add, in the pattern:** a new component that follows the structure of a pattern is added
- **Add, out of the pattern:** a new component that does not follow the structure of a pattern is added
- **Modify:** the structure of several pattern components changes to a lesser or greater extent
- **Delete:** a component is removed

As mentioned above, each solution for an ADD in the CoCoADvISE approach implies the application of the related AK, mostly captured by software patterns but also other kinds of AK like technology specific AK. Each solution can have positive, negative, or neutral impact on the observed QAs. Of course, it is often the case that chosen patterns and other AK cannot satisfy all desired QAs, that need to be improved using corresponding tactics.

To further support quality-driven ADDs, we propose to integrate tactics in the previous CoCoADvISE approach. In that sense, for each obtained ADD solution, we propose to provide a list of tactics that are applicable in the context of that solution and that can be used to improve the observed QAs. The explanation of the impacts of the provided tactics on observed QAs will be also provided (see an example below). As mentioned above, that impact can be either positive, negative, or neutral. Furthermore, for each added tactic, we propose to add how difficult is to implement it in the context of the obtained solution (e.g. a selected software pattern). To do that, we adopt the five-point scale defined in the work by Harrison and Avgeriou [7] (see above).

Figure 1 shows a new, extended CoCoADvISE meta-model.

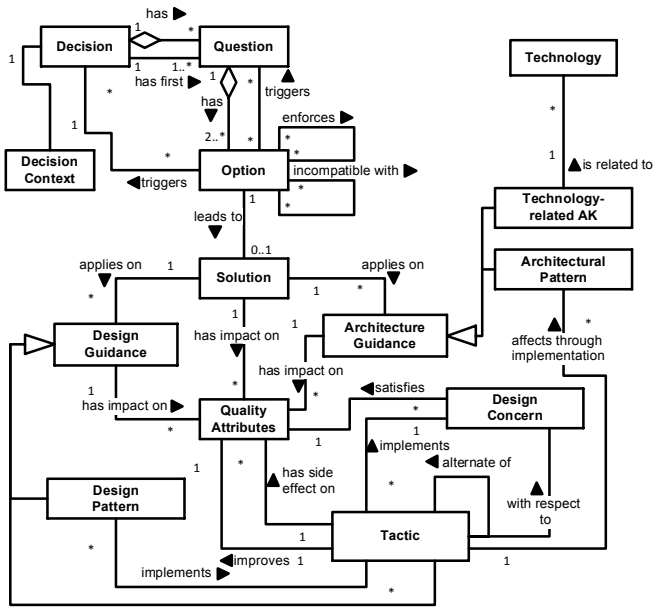


**Figure 1: A New Reusable Architectural Decision Meta-Model**

To enable the integration of tactics in our previous approach, the interactions between them and the QAs as well as the AK resulted from the obtained ADD solutions must be systematically studied in order to crate a reusable model (i.e. an instance of the given meta-model) that can be used as input to the CoCoADvISE tool. Creating the reusable input model is the most demanding task in making ADDs with CoCoADvISE. However, the existing approaches and tools can facilitate it to a large degree. For example, as mentioned above, the interactions between architectural patterns and tactics have already been studied in the literature and they can be used as a good basis for creating the model. Regarding the interactions between tactics, QAs and other AK, like design patterns and technology-related AK, there also exist some work. For example, several authors investigated the relationships between some design patterns and tactics, and found that design patterns implement different tactics [1, 19]. Technology-related AK such as concrete technology solutions (e.g. frameworks) to implement the architecture might differ with regard to several QAs [13]. Therefore, technology-related solutions can influence some decisions by supporting or limiting some QAs. However, beside existing knowledge, each model is related to a specific decision context, e.g. the context of a smart-city software ecosystem. Therefore, the relationships between the model elements (i.e. tactics, QAs, patterns, etc.) have to be considered within that context which may require certain changes or adaptations of the existing work solutions. For example, the impacts of tactics or patterns on some QAs might differ in different contexts because QAs are always viewed within the specific context: given a system state and input, the output is required to be within specific limits [2]. Please note also that the order of integrating patterns and tactics in the final architecture is important. For example, some tactics can take advantage of some patterns that are compatible with them or some already made changes (e.g. a new integrated component) required to integrate some previous tactic (see [7] for more details). To further facilitate and accelerate the creation of a reusable input model for CoCoADvISE, we proposed in our previous work an approach that enables capturing and reusing acquired knowledge using a set of tools [17].

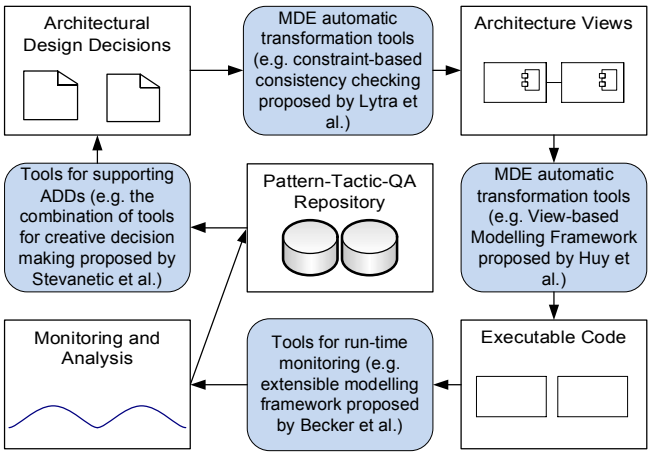


**Figure 2: Adaptable and Controllable Decision-Making - An Overview**

# 4. ADAPTABLE DECISION-MAKING

In this section, we present an approach that supports automated system evolution based on controlled and adaptable quality-driven decision-making using real data obtained during system monitoring. The approach is based on the integration of the previously explained approach for enabling quality-driven decision making using tactics with the existing model-driven development (MDE) paradigm and the corresponding tools. The MDE paradigm enables (semi-)automatic transformation and mapping between different application artefacts starting from architectural decisions and designs until system implementation. Based on the feedback obtained from monitoring the desired QAs both during design-time and run-time, architects can get a precise picture on how the chosen ADD solutions, i.e. patterns and tactics, impact the desired QAs.

The development processes usually used today are highly iterative and incremental such as the Rational Unified Process (RUP) [12]. Following the logic in these approaches, our approach aims at enabling continuous adaptations of the system by adapting its ADDs with regard to the observed QAs.

Figure 2 shows an overview of the approach. The Pattern-Tactic-QA repository represents a wikilike repository for collecting related knowledge about patterns, tactics, and QAs together with their relationships. We have already created one such repository described in our previous work mentioned above [17]. The repository currently contains records for over 200 design patterns, over 50 tactics, and about 80 QAs. The Pattern-Tactic-QA repository is used as a knowledge base for creating a reusable input model for the CoCoADvISE tool, that is then used for capturing and documenting concrete ADD solutions. For that purpose, our previously explained approach (see Section 3) can be used, i.e. an architect can utilize the provided impacts of the selected ADD solutions on the desired QAs as well as the information on how difficult is to implement selected patterns-tactics combinations. For making ADDs the first time, the impacts of selected patterns-tactics combinations on the QAs are only coarsely assessed as positive, negative or neutral (as explained in Section 3). When all required decisions are made, model-driven techniques can be utilized to transform the obtained ADD solutions into architectural views, automatically and in a reusable way (for example using the approach by Lytra et al. [15]). Furthermore, high-level views such as component and connector models can be refined and enriched to generate lower level views such as technology and platform-specific views. For that purpose a technique such as View-based Modelling Framework (VbMF) proposed by Tran et al. can be used [18]. In VbMF, based on the view models, source code and configuration artefacts are automatically generated for some common constructs. The hand-written code can be added to the generated code in order to complete the implementation. To deploy the generated source code, as usual for the service-based applications, deployment descriptors and configuration files are created. During the system execution, the desired QAs can be monitored. Of course, we refer here to variable and dynamically changing QAs that are hard to compute at design-time. Other, static QAs can be verified at design time. Run-time monitoring frameworks like the one proposed by Becker et al. [4] can be used for monitoring dynamically changing QAs. In particular, services can be run in some runtime monitoring engine that is used for measuring the dynamically changing QAs. The obtained measurements can be analysed and then fed back to the Pattern-Tactic-QA repository. Those measures precisely quantify the impacts of the chosen ADD solutions on the observed QAs and can be used by architects to adapt ADDs accordingly by comparing impacts from different solutions.

So far, the part of the approach shown in Figure 2 related to the Pattern-Tactic-QA repository and tools for supporting ADDs has been implemented. The MDE transformation tools as well as the tool for run-time monitoring need to be integrated.

The approaches presented in the previous two sections are general enough and can in principle be applicable for other types of systems (or ecosystems) beside those that are service-based. However, the approaches and tools considered in the adaptable decision making approach are specifically related to the service-based systems. Additional investigations are necessary to better explain how the approach can be concretely applied for other types of systems.

# 5. DECISION MODEL EXAMPLE

In this section, we provide an example of a reusable architectural decision model for data processing. The example is shown in Figure 3. As explained in Section 3, for each solution represented by the design or architectural guidance, a set of tactics with regard to the considered QAs is added. In our example, we provide only three tactics because of space limitations: Authorization (Security), Ping-Echo (Availability), and Introduce Concurrency (Performance). After each solution, a set of directly affected QAs from that solutions is shown in square brackets (see the figure). Then, the given tactics are provided, describing the effort required to integrate them in the given solution (based on the scale provided in Section 3) and their impacts on the considered QAs. The Authorization and Ping-Echo tactics generally require a single central component where other components need to be appropriately adapted, therefore they have the MODIFY + ADD-OUT impact [7]. The Introduce Concurrency tactic can be implemented in the first filter for Pipes and Filters (ADD2 in the figure) and probably need to be implemented in each component for linear processing of dependent programs (ADD3 in the figure). Please note that all tactics cannot be implemented for all solutions, which depends on the solution. Also, the effort to integrate tactics is provided without considering the order of tactics integration. For example, when the Authorisation tactic is already implemented, the Ping-Echo tactic can be added using the same central component.

With regard to the approach described in Section 4, each solution in the example would need to be further transformed to the corresponding architecture view using the given MDE tools. For example, an architectural component model for the solution with Pipes and Filters and Ping-Echo would have filter components together with an additional central component for supporting Ping-Echo. Architectural views would be then further processed as explained in Section 4.

# 6. CONCLUSIONS

In this paper, we propose an approach for quality-driven architectural decision making and documenting, by inte-

**Context**: A data processing task or set of data processing tasks
**Considered QAs**: Development Effort (DE), Performance (P), Flexibility (F), Availability (A), Security (S)
**ADD1**: Are multiple different processing steps needed to process the data?

- Yes -> ADD2
- No -> Single Data Processing Component
  [DE +, P -, F -, A - (single point of failure), S -]
  - MODIFY + ADD-OUT: Authorization [DE -, P -, S +, A -]
  - MODIFY + ADD-OUT: Ping-Echo [DE -, P -, S -, A + (Ping-Echo), A - (single point of failure)]

**ADD2**: How the different data processing steps are interconnected?

- They need to be flexibly composed for different data processing tasks ->
  Pipes and Filters [DE -, F +, A +, P -, S -]
  - IMPLEMENTED-IN: Introduce Concurrency [P +, DE -]
  - MODIFY + ADD-OUT: Authorization [DE -, P -, S +, A -]
  - MODIFY + ADD-OUT: Ping-Echo [DE -, P -, S -, A + (Ping-Echo), A - (single point of failure)]
- They follow mostly a fixed linear order -> ADD3

**ADD3**: Are the data processing components independent programs?

- Yes -> Use Batch Sequential [P -, F -, DE -, A -, S -]
  - Is it possible to modify the independent programs?
    - YES
      - MODIFY + ADD-OUT: Authorization [DE -, P -, S +, A -]
      - MODIFY + ADD-OUT: Ping-Echo [DE -, P -, S -, A + (Ping-Echo), A - (single point of failure)]
    - NO
- No -> Use programming logic to interconnect multiple processing components [DE -, P -, F -, A -, S -]
  - MODIFY: Introduce Concurrency [P +, DE -, A -, S -]
  - MODIFY + ADD-OUT: Authorization [DE -, P -, S +, A -]
  - MODIFY + ADD-OUT: Ping-Echo [DE -, P -, S -, A + (Ping-Echo), A - (single point of failure)]

**Figure 3: An Instance of a Reusable Architectural Decision Meta-Model from Figure 1**

grating tactics with reusable ADDs together with their impacts (from both tactics and ADDs) on the system QAs. A new meta-model describing the relationships among reusable ADDs (modelled as Questions, Options, and Criteria), established AK (e.g. existing software patterns) that captures the ADD solutions, tactics, and QAs is presented. In addition, we propose an approach for supporting automated system evolution based on controlled and adaptable quality-driven decision making using real data obtained during system monitoring. The approach is based on the integration of the given approach that uses tactics with existing model-driven development paradigm and the corresponding tools. In our future work we plan to fully implement the given approaches and test them in the context of a real software ecosystem.

# 7. 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. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. Addison-Wesley Professional, 2003.

[3] 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*, PFE'01, pages 169–186, London, UK, 2002. Springer-Verlag.

[4] C. Becker, H. Kulovits, M. Kraxner, R. Gottardi, and A. Rauber. *An Extensible Monitoring Framework for Measuring and Evaluating Tool Performance in a Service-Oriented Architecture*, pages 221–235. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

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

[6] 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)*, pages 125–134, Feb 2008.

[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 and P. Avgeriou. *Implementing Reliability: The Interaction of Requirements, Tactics and Architecture Patterns*, pages 97–122. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[9] N. B. Harrison, P. Avgeriou, and U. Zdun. Using Patterns to Capture Architectural Decisions. *IEEE Software*, 24(4):38–45, 2007.

[10] 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.

[11] R. Kazman, J. Asundi, and M. Klein. Quantifying the Costs and Benefits of Architectural Decisions. In *23rd Int'l Conf. on Software Engineering (ICSE)*, pages 297–306, 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] N. Lim, T. Lee, and S. Park. A comparative analysis of enterprise architecture frameworks based on EA quality attributes. In *10th ACIS International Conference on Software Engineering, Artificial Intelligences, Networking and Parallel/Distributed Computing, SNPD 2009, in conjunction with 3rd International Workshop on e-Activity, IWEA 2009, 1st International Workshop on Enterprise Architecture Challenges and Responses, WEACR 2009, Catholic University of Daegu, Daegu, Korea, 27-29 May 2009*, pages 283–288, 2009.

[14] 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 *Proceedings of the Third International Workshop on Software Engineering for Systems-of-Systems*, SESoS '15, pages 37–43, Piscataway, NJ, USA, 2015. IEEE Press.

[15] I. Lytra, H. Tran, and U. Zdun. Constraint-based consistency checking between design decisions and component models for supporting software architecture evolution. In *16th European Conference on Software Maintenance and Reengineering (CSMR)*, pages 287–296, Szeged, Hungary, March 2012. IEEE Computer Society.

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

[17] S. Stevanetic, K. Plakidas, T. B. Ionescu, F. Li, D. Schall, and U. Zdun. Tool support for the architectural design decisions in software ecosystems. In *Proceedings of the 2015 European Conference on Software Architecture Workshops*, ECSAW '15, pages 45:1–45:6, New York, NY, USA, 2015. ACM.

[18] H. Tran, U. Zdun, and S. Dustdar. View-based and model-driven approach for reducing the development complexity in process-driven soa. In *Proceedings of International Conference on Business Processes and Services Computing*, Leipzig, Germany, September 2007.

[19] J. Zheng and K. E. Harper. Concurrency design patterns, software quality attributes and their tactics. In *Proceedings of the 3rd International Workshop on Multicore Software Engineering*, IWMSE '10, pages 40–47, New York, NY, USA, 2010. ACM.

[20] 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*, pages 15–32. Springer, 2007.