MoSAIC: a Middleware-induced Software ArchIteCture design Decision Support System

Francesco Nocera
Polytechnic University of Bari
Bari, Italy
francesco.nocera@poliba.it

Marina Mongiello
Polytechnic University of Bari
Bari, Italy
marina.mongiello@poliba.it

Eugenio Di Sciascio
Polytechnic University of Bari
Bari, Italy
eugenio.disciascio@poliba.it

Tommaso Di Noia
Polytechnic University of Bari
Bari, Italy
tommaso.dinoia@poliba.it

ABSTRACT

Software Architecture design is a relevant issue in the software development. It is used for communication among the system's stakeholders, and facilitates their understanding about design decisions and design rationale. In this field, architectural knowledge comprises more than design decisions and capture their relationships with requirements and architecture design.
In this work, we present MoSAIC, a decision support system based on a knowledge-based approach for managing and reasoning on design decisions of Middleware-induced Software Systems Architecture. The approach is based on a fuzzy ontology to model relationships among Architectural, Functional and Non-Functional Requirements, design decisions and architecture design.

CCS CONCEPTS

• **Computer systems organization** → **Architectures**; • **Software and its engineering** → **Software organization and properties**; • **Information systems** → *Information retrieval*;

KEYWORDS

Software Architecture, Middleware, Software Requirements, Architectural design, Ontology

ACM Reference Format:
Francesco Nocera, Marina Mongiello, Eugenio Di Sciascio, and Tommaso Di Noia. 2018. MoSAIC: a Middleware-induced Software ArchIteCture design Decision Support System. In *Proceedings of ECSA'18*. ACM, New York, NY, USA, Article 4, 4 pages. https://doi.org/10.475/123_4

1 INTRODUCTION AND MOTIVATION

Software Architecture plays a critical role in determining the success of a system. Today, the software design phase has evolved from an ad-hoc, and sometimes overlooked phase, to an essential phase of the development life-cycle [3]. Furthermore, the increasing complexity of today's systems induced by the upcoming technological revolution including the Internet of Things (IoT) and Cloud Computing [21, 23, 26], has shaped a set of particular challenges that makes it hard for software engineers to meet the continuous customer request for higher software quality. These challenges have encouraged software engineers to pay closer attention to the design process to better appreciate, apply, and promulgate well known design principles, processes, and qualified practices to overcome these challenges.
In this context, a middleware can offer common services for applications and ease application development by integrating heterogeneous computing and communications devices, and supporting interoperability within the diverse applications and services running on these devices [22].
In software development the main goals to accomplish are customer and quality requirements satisfaction, correct execution of the software systems, and cost effective adaptation to future changes [1]. In order to reach these challenging goals, tools and techniques may help to manage and retrieve the knowledge necessary for decision making processes. For this reason, recent research trends are focused to strengthen the reasoning and decision-making process to reach these goals [1]. The development of a software system is in fact determined partly by its functionality called Functional Requirements (FRs) i.e., what the system does -, by Architectural Requirements (ARs) [10, 28] and by requirements about Quality Attributes (QAs). Such requirements are known as Non-Functional Requirements (NFRs). During architectural design the selection of NFRs is a relevant task, since can be used as selection criteria for choosing the proper design solution among several ones. On the other hand, exploiting design decisions during development emerges as a Software Architecture [16]. The software architecture is the result of the work of an architect or of a designers team and can be modeled by taking into account the needed QAs. To reach this goal, an architect can use primitive design techniques. These primitives are known as tactics. Generally a tactic can be considered as a modeling solution and is related to satisfying a given QA. An architectural strategy is made up of several tactics. The architect takes design decision based on a strategy, hence selects a set of tactics, and hence a set of patterns. The design process requires a choice of the best combination of tactics to achieve the system's desired goals [4]. For this reason during the design of Middleware-induced Software Architecture a challenging tasks are the selection

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
ECSA'18, September 24-28 2018, Madrid, Spain
© 2018 Copyright held by the owner/author(s).
ACM ISBN 123-4567-24-567/08/06.
https://doi.org/10.475/123_4

of *(i)* best patterns and *(ii)* best existing Middleware able to satisfy desired requirements [14, 20, 28]. A main difficult derives from the relationship between requirements and patterns that can be complementary, can be composed, sometimes cooperate to solve a larger problem or are exclusive in a modeling task [8]. Existing classifications or quality models have been defined to categorize QAs and requirements, anyway a systematic classification of NFRs towards architectural design and a description of their use during system modeling are missing [2, 24]. Moreover, only little work has been done in using a knowledge-based approach to support such activities [18].

The main objective of this research is the *implementation of a semi-automated tool aimed at supporting decision makers to derive knowledge able to solve architectural problems.*

More precisely we develop a Decision Support System for supporting designers and software architect – having greater or lesser experience – in the architectural design of Middleware-induced Software Architecture. Specifically, we will show how the following task can be addressed: *given a desired set of FRs, NFRs and ARs perform the tasks of (i) retrieving the smallest subset of patterns that best match them and (ii) recommending existing middleware can support all the necessary requirements.*

No other method similar to the proposed one was found. This paper considerably extends our previous works [13, 25] where a theoretical framework using Fuzzy DL for modeling knowledge about NFRs, pattern and a reasoning task have been proposed.

2 BACKGROUND

2.1 Requirements, Design and Architectural patterns

Design and Architectural Pattern. Reusable solutions of design models are available mainly realized using patterns. These approaches are important vehicles for constructing high-quality software architectures since provide already tested solutions [9]. Design patterns were proposed during the last decades are reusable solutions to modeling recurrent problems. They are mainly based on the expert's experience that use solution proposed for similar problems [9, 15]. Therefore, the use of patterns or tactics [5, 24] for architectural modeling constitutes an effective solution for addressing design decisions [17]. They also support the construction and documentation of software architectures. In summary, patterns provide a set of predefined design schemes for software systems organization, and provide an abstract formalization of the design solution[9]. According to [8], design patterns can be classified into families that identify a problem area addressing a specific topic [27]. The pattern language proposed in [8] includes 114 patterns, which are grouped into 13 main problem areas addressing a specific technical topic (Distribution Infrastructure, Component Partitioning, Object Interaction, Interface Partitioning, Synchronization, Application Control, Event Demultiplexing and Dispatching, Adaptation and Extension, Modal Behavior, Database Access, Concurrency, Resource Management and Cloud).

Non-Functional Requirements(NFRs). A software system design is obtained as the results of both FRs implementation i.e., what the system does - and requirements about development specifications or QAs. Such requirements concern development features, operational costs, but also quality attributes [10]. NFRs are crucial in software design since help designers in selecting the supposed best design solution among several alternatives. For this reason, best practices in taking into account these requirements is the basis for ensuring successful development. Also taking properly into account requirements can prevent errors which may have negative impact on management and costs of the whole software project. NFRs lend to a qualitative assessment more than an objective definition, for this reason identifying a particular NFR is a challenging task. Besides, identifying NFRs is an architectural task, so the availability of reusable solutions that ensure satisfaction of a given NFR or of a set of NFR gives a valid support to performing architectural modeling. Description of knowledge about NFRs is proposed in NFR catalogues [11, 12, 19] those enabling reuse and catalogation and a useful way of addressing the need for help on NFR elicitation. For a detailed description of this requirements and its relations with Design pattern and Pattern Families we refer to our previous analysis[13], where these relations have been identified.

Middleware in Internet of Things (IoT) and its requirements. Middleware is necessary to ease the development of the diverse applications and services in IoT. Many solutions have been proposed and implemented, especially in the last couple of years. These solutions are highly diverse in their design approaches, level of programming abstractions, and implementation domains (e.g., WSNs, RFID, M2M, and SCADA) [28]. The proposals are diverse and involve various middleware design approaches and support different requirements. Based on the analysis in recent surveys [28, 29], the existing middleware solution are grouped based on their design approaches, as below: Event-based (e.g. Hermes, EMMA, RUNES, PRISMA, SensorBus, Mires and so on), Service-oriented (e.g. Hydra, SOCRADES, ubiSOAP, MOSDEN and so on), Virtual Machine-based (e.g. VM, DVM, TinyVM, SwissQM, and so on), Agent-based (e.g. Ubiware, Impala, MASPOT, and so on), Tuple-spaces (e.g. LIME, TeenyLIME, TinyLIME, TS-Mid and so on), Database-oriented (e.g. SINA, COUGAR, IrisNet and so on), Application-specific (e.g. Milan, AutoSec, Adaptive Middleware and so on). Some middleware use a combination of different design approaches. In the survey by Razzaque et al. [28], all existing middleware in literature are described briefly. A middleware provides a software layer between applications, the operating system and the network communications layers, which facilitates and coordinates some aspect of cooperative processing. From the computing perspective, a middleware provides a layer between application software and system software. Based on previously described characteristics of the IoT's infrastructure and the applications that depend on it, a set of requirements for a middleware to support the IoT is outlined. These requirements are grouped into the following two sets: *(i) Middleware Service Requirements.* Middleware Service Requirements for the IoT can be categorized as both Functional and Non-Functional. FR capture the services or functions (e.g., resource discovery, resource management, data management, event management, code management) a middleware provides and NFRs (e.g., scalability, real-time o Timeliness, reliability, availability, security & privacy, ease-of deployment and popularity) capture QoS support or performance issue; *(ii) Architectural Requirements.* The Architectural Requirements support application developers: programming, abstraction, interoperable, service-based, adaptive, context-aware, autonomous

and distributed.
For a detailed description of this requirements we refer to the survey by Razzaque M.A. et al. [28] and its related references.

3 MOSAIC

In this section we present the implemented ontology-driven web application. The overall architecture is depicted in Figure 1.

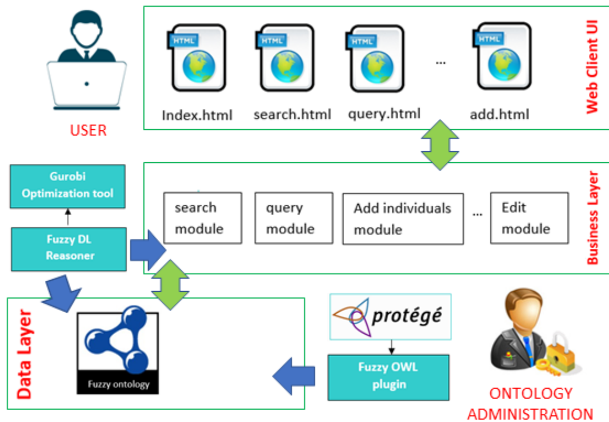

Figure 1: MoSAIC Architecture, and linked components.

MoSAIC [1] was developed using Eclipse as a Web development platform integrating Apache Tomcat [2] as application server. The business layer is consisting of a modules that implements the functionalities of the tool. First of all enable *(i)* search of Middleware, design patterns, NFRs, FRs, ARs and Pattern Families query the ontology showing their *annotations*; and *(ii)* solve the queries of retrieving the smallest subset of patterns that best match the input requirements and recommending existing Middleware can support all the necessary requirements.
Ontology Administration provides for three main functionalities, that are, the insertion of a new individual (NFR, FR, AR, Middleware or Design pattern), the insertion of a Fuzzy DL statements (relations intercurring between individuals) and the editing of ontology individuals. Before executing any change on the ontology, i.e. any of the administrator functionalities, a consistency check is performed. The Linked components are the following:

- **Fuzzy DL Reasoner** [7] is a java-based reasoner allowing to work with vague information, previously described;
- **Gurobi**[3] is a library for mathematical programming. It is focused on linear programming solver (LP solver), quadratic programming solver (QP solver), quadratically constrained programming solver (QCP solver), mixed-integer linear programming solver (MILP solver), mixed-integer quadratic programming solver (MIQP solver), and mixed-integer quadratically constrained programming solver (MIQCP solver) and it is used within the Fuzzy DL reasoner for MILP calculations;
- **Protégé**[4] is an open source ontology editor that supports the Web Ontology Language (OWL);
- **FuzzyOWL plugin**[5] is a plugin for Protégé that allows users to edit, save Fuzzy OWL 2 ontologies, and submit queries to the underlying inference engine FuzzyDL.

The MoSAIC core component is the Fuzzy ontology. We constructed a Fuzzy OWL 2 ontology [6] by extending our previous ontology [13] according to the Linked Data principles[6] by reusing URIs already available in the Web. We referred to DBpedia[7] as it is a “de facto” standard in the representation of entities in the Web. In Figure 2 is depicted the MoSAIC Fuzzy ontology classes hierarchy (The green nodes are reused classes).
We collected the knowledge about 109 design patterns, 28 pattern families, 37 NFRs, 61 existing Middleware in literature grouped based on their 7 design approaches and its 14 NFRs, 5 FRs, 8 ARs together with their mutual relations. Other important metrics associated to our ontology, whose consistency has been checked and validated with the *fuzzyDL* Reasoner are the following: *Axiom* (4.682), *Logical axiom count* (3.474), *Functional Data Property axioms* (69), *Data Property Range axioms* (69), *Class Assertion axioms* (280), *Object Property Assertion axioms* (1.560), *Data Property Assertion axioms* (1.434). We refer – due to lack of space – to our previous work [13] about the Fuzzy definition approach adopted in order to encode all the new relations between individuals retrieved from Razzaque et al. survey [28] (viewable thanks to the online documentation[8]). According to this metrics, the new knowledge encoded within the MoSAIC Ontology extends the previous ontology of around 308 %. In order to solve the queries of retrieving the smallest subset of software patterns and recommending existing Middleware can support all the necessary requirements, we adapted our previously defined Covering Answer Set task[13]. In order to retrieve the set of Middleware – in number, chosen by the user – with the high grade of satisfiability (Score) of all the needed requirements, a Ranking was performed with respect to the covered FRs, ARs, and its NFRs. The retrieval of the best pairs set of patterns that best satisfies the needed requirements, will be obtained by using Covering Answer Set task with respect to the covered NFRs and Pattern Families. As depicted in Figure 2, we defined a new *Non_Functional_Requirements_middleware* class in order to encode only the NFRs related to Middleware, also to improve the Covering Answer Set task computation.

4 CONCLUSION AND FUTURE WORK

In this work was proposed as main goal the development of a Decision Support System for supporting designers and software architect – having greater or lesser experience – in the process of modeling a Middleware-induced software system architecture. Main strengths of our approach is the use of a *fuzzy ontology* for modeling Knowledge that relates middleware design approaches and FRs, ARs, NFRs with patterns and the Families they belong to.

[1]MoSAIC's website at:
http://sisinflab.poliba.it/tools/softarch/mosaic. A demo video is at 'SiSInfLab' YouTube Channel: https://www.youtube.com/channel/UCtstyhaP_aaYnd57f8lZ5IA/
[2]Apache Tomcat software: https://tomcat.apache.org/
[3]The Gurobi Optimizer library is available at www.gurobi.com.
[4]It is available for free download at http://protege.stanford.edu/.
[5]Fuzzy OWL plugin: https://protegewiki.stanford.edu/wiki/FuzzyOWL2
[6]http://www.w3.org/DesignIssues/LinkedData.html
[7]http://dbpedia.org
[8]MoSAIC ontology is at http://sisinflab.poliba.it/semanticweb/ontologies/mosaic/

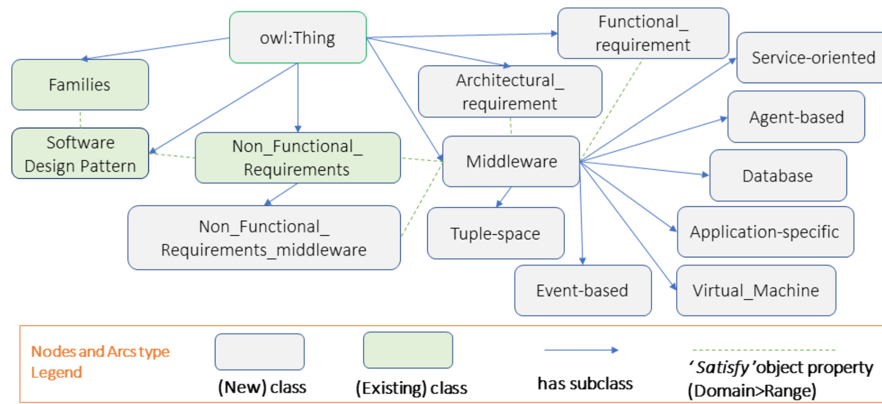

Figure 2: MoSAIC Fuzzy Ontology classes hierarchy.

We are currently working on how to take into account also information related to temporal interaction among software components thus extending the Fuzzy DL we propose with temporal operators.

ACKNOWLEDGEMENT

The authors acknowledge support of Salvatore Petroni, Marco Di Stani, Pasquale Moramarco and Roberto Laricchia for tool improvements.

REFERENCES

[1] Paris Avgeriou, John Grundy, Jon G Hall, Patricia Lago, and Ivan Mistrík. 2011. *Relating software requirements and architectures.* Springer Science & Business Media.

[2] Paris Avgeriou and Uwe Zdun. 2005. Architectural patterns revisited–a pattern language. In *Proc. 10th European Conf. Pattern Languages of Programs (EuroPLoP).* 431–470.

[3] Len Bass. 2007. *Software architecture in practice.* Pearson Education India.

[4] Len Bass, Paul Clements, and Rick Kazman. 2005. *Software architecture in practice.* Addison-Wesley, Boston ; Munich [u.a.].

[5] L. Bass, M. Klein, and F. Bachmann. 2002. Quality attribute design primitives and the attribute driven design method. In *Revised Papers from 4th Int.Workshop on Software Product-Family Engineering.* Vol. 2290. Springer, 169–186.

[6] Fernando Bobillo and Umberto Straccia. 2011. Fuzzy Ontology Representation using OWL 2. *International Journal of Approximate Reasoning* 52 (2011), 1073–1094. Issue 7.

[7] Fernando Bobillo and Umberto Straccia. 2016. The fuzzy ontology reasoner fuzzyDL. *Knowledge-Based Systems* 95 (2016), 12–34.

[8] Frank Buschmann, Kevlin Henney, and Douglas C Schmidt. 2007. *Pattern-Oriented Software Architecture, Volume 4, A Pattern Language for Distributed Computing.* Wiley.

[9] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. 1996. *Pattern-oriented software architecture: a system of patterns.* John Wiley & Sons, Inc., New York, NY, USA.

[10] Lawrence Chung, Brian A Nixon, Eric Yu, and John Mylopoulos. 2012. *Non-functional requirements in software engineering.* Vol. 5. Springer Science & Business Media.

[11] Luiz Marcio Cysneiros. 2007. Evaluating the Effectiveness of Using Catalogues to Elicit Non-Functional Requirements.. In *WER.* 107–115.

[12] Tommaso Di Noia, Marina Mongiello, and Eugenio Di Sciascio. 2014. Ontology-driven pattern selection and matching in software design. In *European Conference on Software Architecture.* Springer, 82–89.

[13] Tommaso Di Noia, Marina Mongiello, Francesco Nocera, and Umberto Straccia. 2018. A fuzzy ontology-based approach for tool-supported decision making in architectural design. *Knowledge and Information Systems* (27 Mar 2018). https://doi.org/10.1007/s10115-018-1182-1

[14] Alexander Egyed and Paul Grunbacher. 2004. Identifying requirements conflicts and cooperation: How quality attributes and automated traceability can help. *Software, IEEE* 21, 6 (2004), 50–58.

[15] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. 1994. *Design patterns: elements of reusable object-oriented software.* Pearson Education.

[16] David Garlan and Mary Shaw. 1994. An introduction to software architecture. (1994). technical report.

[17] Neil B Harrison and Paris Avgeriou. 2010. Implementing reliability: the interaction of requirements, tactics and architecture patterns. In *Architecting dependable systems VII.* Springer, 97–122.

[18] Zengyang Li, Peng Liang, and Paris Avgeriou. 2013. Application of knowledge-based approaches in software architecture: a systematic mapping study. *Information and Software technology* 55 (2013), 777–794.

[19] Claudia López, Luiz Marcio Cysneiros, and Hernán Astudillo. 2008. NDR ontology: sharing and reusing NFR and design rationale knowledge. In *Managing Requirements Knowledge, 2008. MARK'08. First International Workshop on.* IEEE, 1–10.

[20] Dewi Mairiza, Didar Zowghi, and Nurie Nurmuliani. 2009. Managing conflicts among non-functional requirements. In *12th Australian Workshop on Requirements Engineering.* University of Technology, Sydney, 11–19.

[21] Niko Mäkitalo, Francesco Nocera, Marina Mongiello, and Stefano Bistarelli. 2018. Architecting the Web of Things for the fog computing era. *IET Software* (April 2018). http://digital-library.theiet.org/content/journals/10.1049/iet-sen.2017.0350

[22] Marina Mongiello, Tommaso di Noia, Francesco Nocera, Eugenio di Sciascio, and Angelo Parchitelli. 2016. Context-Aware Design of Reflective Middleware in the Internet of Everything. In *Software Technologies: Applications and Foundations*, Paolo Milazzo, Dániel Varró, and Manuel Wimmer (Eds.). Springer International Publishing, Cham, 423–435.

[23] Marina Mongiello, Francesco Nocera, Angelo Parchitelli, Luigi Patrono, Piercosimo Rametta, Luca Riccardi, and Ilaria Sergi. 2018. A Smart IoT-Aware System For Crisis Scenario Management. *Journal of Communications Software and Systems* 14, 1 (2018), 91–98.

[24] P. Avgeriou N.Harrison. 2007. Pattern-driven architectural partitioning: Balancing functional and non-functional requirements. In *Second International Conference on Digital Telecommunications 2007. ICDT '07. IEEE.* IEEE, 21–26.

[25] Francesco Nocera. 2016. Fuzzy ontology-driven web-based framework for supporting architectural design: student research abstract. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing, Pisa, Italy, April 4-8, 2016.* 1361–1362. https://doi.org/10.1145/2851613.2852014

[26] Francesco Nocera, Tommaso Di Noia, Marina Mongiello, and Eugenio Di Sciascio. [n. d.]. Semantic IoT Middleware-enabled Mobile Complex Event Processing for Integrated Pest Management. In *In Proceedings of the 7th International Conference on Cloud Computing and Services Science (CLOSER 2017).* 610–617. https://doi.org/10.5220/0006369506380645

[27] Tommaso Di Noia, Marina Mongiello, and Umberto Straccia. 2015. Fuzzy Description Logics for Component Selection in Software Design. In *Software Engineering and Formal Methods - SEFM 2015 Collocated Workshops: ATSE, HOFM, MoKMaSD, and VERY*SCART, York, UK, September 7-8, 2015, Revised Selected Papers.* 228–239. https://doi.org/10.1007/978-3-662-49224-6_19

[28] Mohammad Abdur Razzaque, Marija Milojevic-Jevric, Andrei Palade, and Siobhán Clarke. 2016. Middleware for internet of things: a survey. *IEEE Internet of Things Journal* 3, 1 (2016), 70–95.

[29] Honbo Zhou. 2012. *The internet of things in the cloud: A middleware perspective.* CRC press.