Developing an Ontology for Architecture Knowledge from Developer Communities

Mohamed Soliman*, Matthias Galster†, and Matthias Riebisch*
*Department of Informatics, University of Hamburg, Germany
Email: {soliman, riebisch}@informatik.uni-hamburg.de
† University of Canterbury, Christchurch, New Zealand
Email: mgalster@ieee.org

***Abstract*—Software architecting is a knowledge-intensive activity. However, obtaining and evaluating the quality of relevant and reusable knowledge (and ensuring that this knowledge is up-to-date) requires significant effort. In this paper, we explore how online developer communities (e.g., Stack Overflow), traditionally used by developers to solve *coding problems*, can help solve *architectural problems*. We develop an ontology that covers architectural knowledge concepts in Stack Overflow. The ontology provides a description of architecture-relevant information to represent and structure architectural knowledge in Stack Overflow. The ontology is empirically grounded through qualitative analyses of different Stack Overflow posts, as well as inter-coder reliability tests. Our results show that the architecture knowledge ontology in Stack Overflow captures architecture-relevant information and supports achieving practitioners' requirements and concerns.**

I. Introduction

In software architecture design, some design decisions impact the whole system and are difficult to change. These decisions are often referred to as Architecture Design Decisions (ADDs) [1]. Knowledge and experience play a crucial role for making good ADDs. Researchers proposed approaches for managing and sharing Architectural Knowledge (AK). For example, AK repositories (e.g., [2]) provide catalogs of architectural solutions (e.g., patterns [3], technologies [4]) and AK concepts (e.g., ADD, design rational). However, due to the complexity and effort required to collect and codify AK, knowledge repositories tend to be incomplete. Moreover, technologies evolve constantly, which demands significant maintenance effort to keep repositories up-to-date.

Online developer communities (e.g., technology forums like Stack Overflow) allow software developers to ask and answer questions about software development problems. The main success factors of such communities is that they provide a "social motivation" to not only ask questions (i.e., to *benefit* from the community), but also to help others by answering questions (i.e., to *contribute* to the community). By contributing to a community, users can build up their reputation. Furthermore, these communities provide useful knowledge management features, such as the assessment of the quality of answers (through voting), and the continuous evolution of knowledge when new questions and answers are added. The benefits provided by developer communities could complement existing AK repositories and AK management approaches.

However, developer communities have traditionally been used by developers to solve *coding-related problems* [5]. These problems are often not relevant to architects, because they focus on lower level implementation details. Nevertheless, architects may also benefit from developer communities to solve *architectural problems*. In our previous work, we found that useful AK is indeed available in Stack Overflow (SO) [6]. We classified architecture-relevant posts (ARPs) in SO and evaluated them with practitioners to ensure that they are really architecture-relevant. An example of such post is "What are the benefits and trade-offs of using MSMQ over a SQL Table" [380052][1]. SO showed to be particularly useful for solving design issues which involve technology decisions that architects face after making higher-level conceptual decisions [2].

As a next step in this line of research, this paper investigates the following research question: **How can we represent and structure architecture knowledge from architecture-relevant Stack Overflow posts?** Answering this question is the first step towards understanding what AK concepts actually exist in developer communities, and how are they represented in text. This is needed to specify AK concepts in developer communities and to *search* and *capture* (e.g., [7]) AK. Since the structure of knowledge is different in different communities and to ensure practical relevance of this work, we selected SO as an example: SO is the most popular community, offers methods for evaluating the quality of contributions and contributors and provides interfaces for downloading the posts for processing and analysis. We randomly selected a sample of ARPs from a corpus of SO posts created previously [6]. We then performed qualitative content analysis to identify knowledge concepts in these posts (see Sec. II). This resulted in an empirically-grounded ontology for AK in SO (see Sec. III). This ontology specifies how each AK concept in ARPs is composed and bridges the gap between existing theoretical AK concepts (e.g., [8]) and their textual representation in SO.

II. Research Process to Define Ontology

A. Data Gathering

To make the analysis of SO posts practically feasible, we focused on ARPs in a single domain to explore questions on SO of different purposes and with different solutions. In our previous work, we created a corpus of posts for the middleware domain [6]. Middleware was selected since it is an established topic in software architecture [3]. The corpus[2] includes 858 ARPs and 1,653 programming posts (i.e., posts which address

[1]To access full posts mentioned in this paper, concatenate *http://stackoverflow.com/questions/* with the provided post number.

[2]*https://swk-www.informatik.uni-hamburg.de/ soliman/ICSA2017.zip*

978-1-5090-5729-0/17 $31.00 © 2017 IEEE
DOI 10.1109/ICSA.2017.31

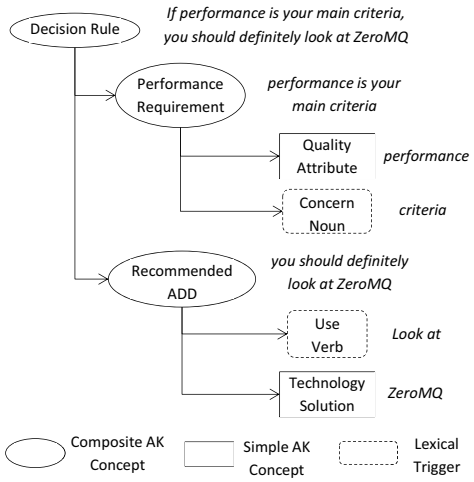

Fig. 1: An example for an annotated statement in ARP

a programming rather than an architectural problem) as classified by practitioners. As a sample to define the ontology we randomly selected 105 ARPs from the 858 ARPs. To better represent the population of ARPs, we selected ARPs using stratified random sampling [9], where ARPs have questions with different purposes and different types of solutions.

B. Data Analysis

We applied qualitative content analysis [10]. We used Atlas.ti, a qualitative data analysis software[3]. It helps human annotators annotate text segments within their context, and to assign them to categories. We annotated selected segments of text from our sample of ARPs. Each annotation is a tuple (*s*,*c*): *s* is a segment of annotated text. This could be a word or a clause or a sentence in a SO ARP question or answer. *c* is an ontology class. An ontology class is an "explicit specification of a conceptualization" [11]. Our content analysis process consists of two phases (Explication and Structuring) [10]:

Explication: Explication aims at interpreting and comprehending text segments in ARPs to obtain a concrete definition for each ontology class. We started the annotation process with a list of ontology classes from known AK concepts (e.g., *Quality Attribute*, *ADD*) in existing AK models (e.g., [8]). However, it was unknown how AK concepts are represented in ARPs. The first author annotated and analysed iteratively sentences, clauses and words in 40 randomly selected ARPs.

During annotation, the representation of each ontology class started to emerge. Some of the AK concepts are represented in text as single words (e.g., words related to concept *Quality Attribute*), we called these ontology classes *Simple AK Concepts*. On the other hand, we found AK concepts, which are represented in text as statements or clauses rather than as single words. We called them *Composite AK Concepts*. This type of classes is composed of other ontology classes in order to construct the semantics of statements or clauses. In addition to existing AK concepts, we discovered an additional type of classes, which are normal English words and not specifically related to a certain AK concept. Still, they deliver important meanings for composite AK concepts. Words in this class are further assigned to existing linguistic semantic categories (e.g., [12]). We called these ontology classes *Lexical Trigger*.

Fig. 1 shows an example of an annotated statement "If performance is your main criteria, you should definitely look at ZeroMQ" [17806977] and annotated clauses and words in a hierarchical representation. Since the whole statement is an if-statement, it was annotated as a *Decision Rule* (i.e., a *Composite AK Concept*), while the condition clause "performance is your main criteria" has been annotated as a *Requirement* (i.e., a *Composite AK Concept*), and the result clause "you should definitely look at ZeroMQ" as a *Recommended ADD* (i.e., a *Composite AK Concept*). Within these two clauses, words with relevant semantics have been annotated. For example, "performance" is annotated for being a *Quality Attribute* (i.e., a *Simple AK Concept*), and "Look at" is annotated as a *Use Verb* (i.e., a *Lexical Trigger*).

Based on this interpretation, the first author created an initial description of each ontology class. This description was the first version of our coding guide. Based on it, we conducted two inter-coder reliability tests, where the first and second author independently annotated randomly selected statements and then both authors met to reconcile disagreements. As a result of this phase, we obtained a mature coding guide[2] with good agreement on the definition of each ontology class. This coding guide was used in the structuring phase.

Structuring: The structuring phase extracted the structure of the ontology. We followed three steps:

1) Complete annotation: Based on the coding guide, we annotated the remaining 65 ARPs. In total, we created more than 3,800 annotations.

2) Determine significant ontology classes: We faced two challenges: a) We had more than 100 composite ontology classes. To refine our ontology, we merged several composite ontology classes based on their frequencies and semantics to have 11 main composite ontology classes. b) We had more than 200 simple ontology classes and lexical triggers, which co-occured with the composite classes. To reach a concrete definition for each composite ontology class, we applied Pearson $\tilde{\chi}^2$ significance test [13] between composite classes and their co-occurring children/composing ontology classes. For example, for the two ontology classes *(REQ) Requirement* and *(QA) Quality Attribute*, we considered frequencies for the following four situations: i) Text annotated as REQ intersect with annotations for QA. ii) Text annotated as REQ intersect with annotations other than QA. iii) Text annotated with class other than REQ intersect with annotations for class QA. iv) Text annotated with class other than REQ intersect with annotations other than QA. We excluded co-occurrences with $\tilde{\chi}^2 < 10$ to ensure that all co-occurrences were statistically significant at $\rho < 0.05$. Simple ontology classes and lexical triggers that did not appear as statistically significantly co-occurring with any composite ontology class have been either merged or excluded from the ontology.

3) Final reliability test: We conducted the final inter-coder reliability test to assess the agreement on the definition of ontology classes. We focused on composite ontology classes, because they are represented in sentences, which make it more

[3]http://atlasti.com/

TABLE I: Significant *Simple AK Concept* ontology classes

ID	Ontology Class Name	Examples of Words
TEC	Technology Solution	WCF, EJB, Netty, RabbitMQ
PAT	Architecture Pattern	REST, messaging, layer, SOA
QA	Quality Attribute	scalability, availability, throughput
COM	Architecture Component	server, backend, service, application
CON	Architecture Connector	read, send, write, communicate
COME	Component Element	interface, operation, record, job, call
COND	Connector Data	message, payload, information, data
PROB	Software Problem	SPOF, error, out of memory
FT	Feature Term	serialization, binding, deployment

TABLE II: Significant *Lexical Trigger* ontology classes

ID	Ontology Class Name	Examples of Words
DIF	Difficulty Adjectives	lightweight, complex, overkill
ADV	Advise Verbs	recommend, suggest, propose
VAL	Value Adjectives	good, outperform, important
CONC	Concern Nouns	requirement, criteria, demand
REL	Rely Verbs	depend, implement, count on
WISH	Wish Verbs	need, require, want, demand, ask
SUPP	Support Verbs	offer, provide, supply, support
USE	Use Verbs	select, choose, use, prefer, go with
QUE	Question Word	which, what, when, how
VS	Versus Preposition	versus, vs., against, contrast
DFF	Difference Noun	difference, distinction
SPED	Speed Adjectives	fast, slow, heavy, quick

challenging to reach agreement for them. The test involved the first and the second author and 15% of the total annotated statements. We calculated Cohen's Kappa reliability coefficient [14] of 0.844 (i.e., reliability and agreement beyond chance).

III. Architecture Knowledge Ontology in Stack Overflow

By combining the hierarchical relationships between the referenced ontology classes across all annotations, we formed a natural language ontology. Our final ontology consists of 54 ontology classes (11 composite AK, 14 simple AK, 29 lexical triggers). In this section, we will present the significant ontology classes and how they are represented and distributed among ARPs. A complete description for the ontology is part of the coding guide[2]. Table I and II show examples of words from our analysis sample for the most significant *Simple AK Concept* and *Lexical Trigger* ontology classes, while Table III presents the main "composite AK concept" ontology classes. In Table III, each composite AK ontology class is briefly described and supported with examples of annotated clauses or statements. In column "Composing Classes", we also provide the significance $\tilde{\chi}^2$ value for each composing class from which the composite ontology class is constructed.

Fig. 2 shows an example for an annotated ARP (additional examples for ARPs are available online[2]). The annotations use the IDs of the composite ontology classes in Table III. The question in the ARP stated a design issue (ontology class *DI*). The design consists of two applications communicating in a cloud environment (ontology class *CONF*). The user then described the need to evaluate possible messaging technologies to decide on the communication between both applications (ontology class *UR*), considering prioritized quality attribute requirements (ontology class *REQ*). One user (Answer 2) shared a success story (ontology class *CASE*) for a well-known system, which solves a similar problem. At the end of the discussion, the user who posted the question posted an

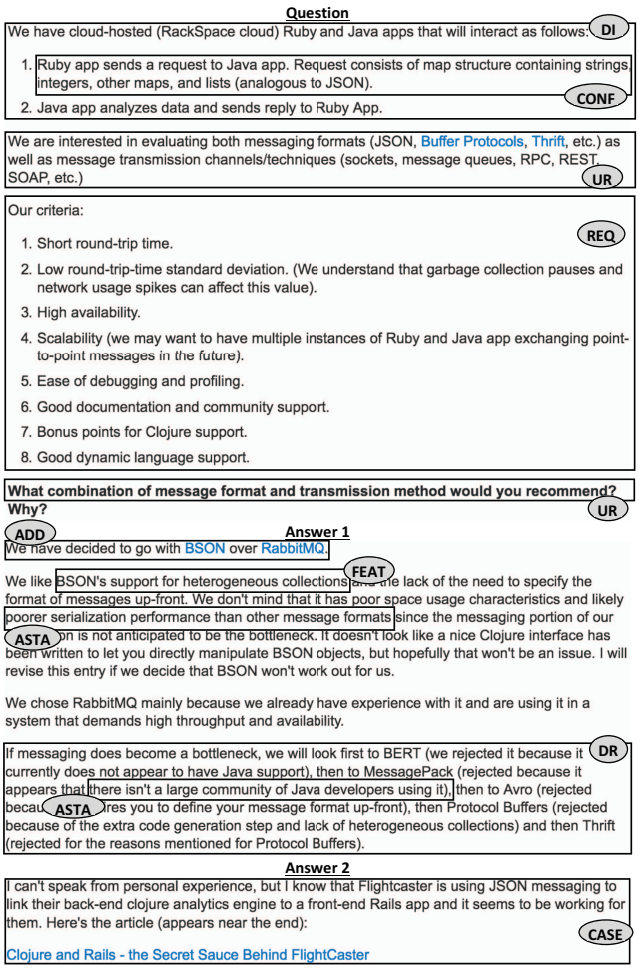

Fig. 2: An example for an annotated ARP [4473567]

answer (Answer 1) to describe the taken decision (ontology class *ADD*) "go with BSON over RabbitMQ", including the rationale for taking this decision, which include technology features (ontology class *FEAT*), their benefits and drawbacks (ontology class *ASTA*), and the logic behind choosing the technology among other alternatives (ontology class *DR*).

IV. Limitations and Threats to Validity

One limitation of our work is related to the number of posts used to develop the ontology from a single domain "middleware". This is caused by the nature of our work, because before posts could be used in our research, we needed to get feedback from practitioners about the architectural relevance of posts. This manual classification requires experienced practitioners rather than novice or less experienced practitioners. Also, manually classifying posts is very time-consuming: It took practitioners around four hours to classify 100 posts (details are provided in [6]). However, we tried to ensure that the annotated ARPs cover different ARPs through a stratified sampling. All in all, we believe our results provide an initial hypothesis for other future studies on AK in developers community.

TABLE III: Significant *Composite AK Concept* classes: description, examples and structure

(ID) Name and Description	Examples	Composing Classes ($\tilde{\chi}^2$)
(**CONF**) *Architecture Configuration*: represents part of an architectural model, which consists of one or more component names associated with an architecture connector verb or name.	"Pushing data from the server to the client [12783677], "Rubby app sends a request to Java app" [4473567]	CON(441.4), COM(326), COND(193.9), COME(41.7)
(**CB**) *Component Behavior*: describes the behavior of an architecture component. It gives an overview about the type of implemented logic and complexity. Sometimes internal operations are mentioned during the description.	"service can be viewed as the business layer of the application"[1582952], "process will run asynchronously"[380052]	COM(85.7), COME(64.6)
(**EX**) *Existing System*: describe part of an architecture of an existing software system. It additionally describes the possible problems in the system.	"An existing process changes the status field of a booking record in a table" [380052]	PROB(158), COM(123.7), CONF(42), CON(35.8)
(**DI**) *Design Issue*: users express their design problems through describing the architecture configurations of a planned design, or the architecture configuration design of an existing software system.	"I want to send a batch of 20k JMS messages to a same queue. I'm splitting the task up using 10 threads." [4741713]	CONF(110), EX(95.3), CB(43), WISH(37.4), USE(14.7)
(**REQ**) *Requirement and Constraint*: two main types of requirements were found: 1) Quality attribute requirements, and 2) Technology features requirements. In addition, we found three types of constraints: 1) Technical skills constraint. 2) Development time constraint. 3) Solution constraint.	"Our criteria (...) Short roundtrip time (...) High availability (...) Scalability (...) Ease of debugging and profiling (...) Good documentation and community support" [4473567]	WISH(213.7), QA(108.7), CONC(74.16), DIF(12.17)
(**UR**) *User Request*: exist in ARP question or title in a form of questions or needs. It complements design issue, requirements and constraints by showing the type of architecture activity (evaluation or synthesis).	"How do I choose between WCF, REST, POX and RIA services for a new Silverlight application" [1582952]	QUE(324.16), TEC(238.8), USE(116), VS(42), DFF(37)
(**FEAT**) *Technology Features*: Two main types of technology features: 1) Development features are expressed through certain programming activities (e.g. debugging) or programming features and tools (e.g. code generation), 2) Behavioral features are expressed through technology specific component and class names, as well as their implemented architectural patterns or their relationship with other technologies.	"EMS is centralized (hub and spoke) on a specific server(s) and can traverse subnets no problem"[1429318], "ActiveMQ is a widely used message broker that offers FIFO queues" [10375137]	TEC(90), PAT(74.8), FT(44.1), REL(35.7), SUPP(13.6)
(**ASTA**) *Technology Benefits and Drawbacks*: They are distinguished through the extensive usage of adjectives and adverbs in combination with technology features and quality attributes. The adjectives or adverbs are used to express the advantages or disadvantages of certain technology solutions or features.	"It is much easier to debug Webservices (...) , which can be easily captured via sniffing tools" [100993], "performance difference will be negligible and in many cases worse for NIO" [19758215]	VAL(130), DIF(102.7), QA(57.4), SPED(49.7), TEC(18.9)
(**CASE**) *Technology Use-Cases*: These are either success or failure stories for the usage of technology solutions at certain contexts. The stories could be coming from personal experiences of users, or well-known examples for existing systems. The context associated with stories could include domain description, architecture configurations, infrastructure, and constraints.	"An application I'm working on has a similar architecture, and I'm planning to use SignalR to push updates to clients, using long polling techniques (...) I have implemented this now, and it works very well" [12783677]	CONF(106), CONC(48.7), USE(45.8), ASTA(29.7), REQ(15.5), ADD(13.3)
(**ADD**) *Recommended Design Decisions*: They are recommendation from users based on their experience or opinion for certain architectural solutions.	"I would highly recommend using WCF; and use the WCF Service Library project over the Silverlight-enabled web service" [361491]	ADV(355), TEC(72), USE(41), CONF(18.49)
(**DR**) *Decision Rules*: Conditional recommendation for architectural solutions. The rule condition might involve other ontology classes such as requirements, constraints and architectural configuration. recommendations involve recommended ADDs for certain technology solution or architecture configuration.	"go with WCF only if you're willing to take on the learning curve." [807692]	ADD(369.4), REQ(226.7), CB(85), ASTA(60.7), CONF(54.9), FT(12.65)

V. Conclusion and Future Work

Finding effective methods for AK sharing and reuse is important to make architects benefit from the power of existing knowledge and experience during their design activities. In this paper, we analyzed textual discussions in one example of a developers community (SO) to explore the structure of architecture knowledge from *initially unstructured information* (posts in Stack Overflow). Based on our analysis, we operationalized the fuzzy concepts of AK into concrete ontology. Our future work involve using and experimenting the developed ontology to implement knowledge capturing functionality (e.g., information extraction and retrieval).

References

[1] A. Jansen and J. Bosch, "Software architecture as a set of architectural design decisions," in *WICSA*, 2005, pp. 109–120.

[2] O. Zimmermann, J. Koehler, F. Leymann, R. Polley, and N. Schuster, "Managing architectural decision models with dependency relations, integrity constraints, and production rules," *Journal of Systems and Software*, vol. 82, no. 8, pp. 1249–1267, 2009.

[3] F. Buschmann, K. Henney, and D. C. Schmidt, *Pattern-Oriented Software Architecture, Volume 4: A Pattern Language for Distributed Computing*. Chichester, UK: Wiley, 2007.

[4] M. Soliman, M. Riebisch, and U. Zdun, "Enriching architecture knowledge with technology design decisions," in *WICSA*, May 2015, pp. 135–144.

[5] C. Treude, O. Barzilay, and M.-A. Storey, "How do programmers ask and answer questions on the web? (nier track)," in *Proceedings of the 33rd International Conference on Software Engineering*, ser. ICSE '11. New York, NY, USA: ACM, 2011, pp. 804–807.

[6] M. Soliman, M. Galster, A. R. Salama, and M. Riebisch, "Architectural knowledge for technology decisions in developer communities: An exploratory study with stackoverflow," in *IEEE/IFIP WICSA 2016*, April 2016, pp. 128–133.

[7] D. C. Wimalasuriya and D. Dou, "Ontology-based information extraction: An introduction and a survey of current approaches," *J. Inf. Sci.*, vol. 36, no. 3, pp. 306–323, Jun. 2010.

[8] R. C. de Boer, R. Farenhorst, P. Lago, H. van Vliet, V. Clerc, and A. Jansen, *Architectural Knowledge: Getting to the Core*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 197–214.

[9] R. Rosnow and R. Rosenthal, *Beginning Behavioral Research: A Conceptual Primer*. Pearson/Prentice Hall, 2008.

[10] P. Mayring, *Qualitative Content Analysis. Theoretical Foundation, Basic Procedures and Software Solution*. Beltz, 2014.

[11] T. R. Gruber, "A translation approach to portable ontology specifications," *Knowledge Acquisition*, vol. 5, no. 2, pp. 199 – 220, 1993.

[12] G. A. Miller, R. Beckwith, C. Fellbaum, D. Gross, and K. Miller, "Wordnet: An on-line lexical database," *International Journal of Lexicography*, vol. 3, pp. 235–244, 1990.

[13] K. Pearson, "On a criterion that a given system of deviations from the probable in the case of correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling," pp. 157–175, 1900.

[14] J. Cohen, "A Coefficient of Agreement for Nominal Scales," *Educational and Psychological Measurement*, vol. 20, no. 1, p. 37, 1960.