A Domain Specific Language and a Pertinent Business Vocabulary for Cloud Service Selection

Mathias Slawik and Axel Küpper

Telekom Innovation Laboratories, Technische Universität Berlin
Service-centric Networking
mathias.slawik|axel.kuepper@tu-berlin.de

Abstract. As more cloud computing offerings become available globally, cloud consumers' efforts of gathering relevant information to support their service selection are raised considerably. On the one hand, high-volume marketplaces, such as Salesforce AppExchange, feature nonformalized offering descriptions. This abstinence of a service formalization impedes cloud consumers' capabilities to both rapidly assess the fulfillment of their selection criteria and to compare different services uniformly. On the other hand, contemporary research on formalized service marketplaces faces significant challenges in its practical application, especially its ease of use and pertinence. In this article we present a novel textual domain specific language for describing services, a pertinent business vocabulary of selection criteria, and a brokering component. These artifacts raise cloud consumers' capabilities while being practically applicable, pertinent to businesses, and easy to use.

Keywords: Cloud Service Brokering, Domain-specific Language, Service Description Language, Cloud Computing.

1 Introduction and Motivation

According to the NIST Definition of Cloud Computing [23] one of the cloud characteristics is *on-demand self-service*: consumers and providers are exempt from having human interaction in order to provision computing capabilities. Therefore, the description of cloud offerings becomes a crucial basis for service selection by cloud consumers.

The service descriptions found within high-volume SaaS marketplaces, such as Salesforce AppExchange [32] and the Google Apps Marketplace [13], rely on nonformalized information, e.g., free text, images, and some structured fields, e.g., author and category. While such content is appropriate for marketing purposes, other uses are impeded as unstructured text is insufficient for comprehensive search and uniform service comparison. Therefore, advanced use cases within service marketplaces require the formalization of service descriptions.

There are state-of-the-art research platforms for marketplaces and service descriptions, e.g., the Unified Service Description Language (USDL) marketplace [33], SPACEflight [37], and the FI-Ware Repository and Marketplace [34]. They

use advanced service models, such as USDL [25], as well as expressive ontologies, such as Linked-USDL [26], and the Web Service Modeling Ontology for the Internet of Services (WSMO4IoS) [39] to describe services.

Applying such research within a small and medium enterprises (SME) application domain is one concept of the TRESOR project, whose architecture we present in [42]. This publication presents the contributions of the TRESOR service broker which matches customer requirements to cloud service capabilities in order to support the service selection decisions by SME cloud consumers. The broker provides a service description mechanism, a matchmaking component, as well as means to support cloud consumers in their service selection.

While realizing the TRESOR service broker we observed four challenge areas in the application of contemporary research within an SME domain. These motivate our contribution and are described in the following paragraphs:

1. Business pertinence. Advanced models and ontologies can expressively and extensively describe services as well as support their global publication and interlinking using Linked Data principles. However, we did not find any model or ontology which specifically represents the selection criteria of SME cloud consumers. To ensure the pertinence of the TRESOR broker to businesses we contribute a derivation of such a vocabulary based on current empiric research.

2. Tooling complexity. We observed an absence of specialized knowledge and skills for effective application of semantic technologies and tools in the SME application domain. To be able to successfully establish the cloud broker in this domain, it has to utilize common technologies and feature a simple design. Consequently, we created the Service Description Language - Next Generation (SDL-NG) framework which features a *Ruby-based internal textual domain specific language (DSL)*. This framework presents simplified semantics and a clean syntax, while allowing the vocabulary and service descriptions to be processed by text-, XML-, as well as RDF-based technologies.

3. Documentation reuse. Due to the self-service nature of cloud computing, cloud services have to be extensively documented. Without changing this documentation (e.g., by introducing annotations), contemporary models and ontologies cannot easily reuse those descriptions, creating *information redundancy* in service descriptions. A common reuse mechanism is parsing cloud vendors websites. We observed that it is a challenging task to integrate website parsers with language frameworks without raising complexity. As our DSL is executed as Ruby-code it can integrate arbitrary operations into service descriptions, creating self-sufficient entities wherein static elements and possibly highly dynamic external content, such as cloud spot market prices, are easily blended.

4. Modeling defiances. There are some traits of cloud services, which cannot be represented effectively with contemporary models and ontologies, such as *modeling service variants*, *price calculation*, as well as *dynamic properties*. While we do not address these in our current work, we outline and delineate impending contributions in this area at the end of this publication.

Publication structure. We apply the Information Systems Research Framework by Hevner et al. [15] which also structures this publication: Section 2 for-

mulates the application environment and its requirements to ensure research *relevance*. To maintain research *rigor*, the related work is contrasted with the requirements in Section 3. The *design* of the approach is presented in Section 4. Section 5 contains its *assessment* and further development, while Section 6 concludes this publication.

2 Application Environment and Requirements Analysis

This section outlines the characteristics of the SME application domain, presents our requirements analysis methodology, and enumerates the resulting requirements. These are contrasted with the related work in Section 3 and provide the basis for the design evaluation in Section 5.

Application environment. The application environment are service ecosystems consisting of SMEs participating in public SaaS marketplaces as cloud consumers and providers. One example is the TRESOR ecosystem which brings together SMEs from the health sector to offer and consume secure and trusted cloud services.

Three fundamental reasons underly having SMEs as the application domain: First, SMEs play an important economic role. In Europe, for example, 99.8% of all enterprises are SMEs and 66.5% of the EU workforce is employed by an SME [44]. Second, SMEs receive significant benefits from cloud services as investigated by Lacity et al. in [20]. This provides a compelling reason for SMEs to participate in SaaS ecosystems as targeted by our approach. Third, the health sector aimed by TRESOR is dominated by SMEs. This allows us to incorporate first-hand experience and provides an appropriate evaluation environment.

Requirement analysis: methodology and results. To identify the requirements for our contribution we applied a multi-stage approach combining different methodologies. These are explicated in the following paragraphs:

First stage (Months 1-6). We first studied cloud-related literature to become familiar with the state-of-the-art. Concurrently, we joined multiple stakeholder group discussions to identify mutual and preliminary use cases, requirements and concepts. We published the results of this preliminary phase in [42].

Second stage (Months 7-18). We adjusted the Volere Requirements Specification Template [3] to capture stakeholder requirements iteratively: conducting topic-focused stakeholder workshops, analyzing and modelling the requirements using UML, and presenting, discussing, and adjusting the results. We identified 186 requirements which motivate previous publications [5], [6], [41], and [47]. We also joined a panel of experts from the accompanying research and other projects from the SME-focused "Trusted Cloud" intiative [12]. There, we compared all approaches to cloud description, matchmaking, and brokering. These group discussions let us assume that our requirements are not TRESOR specific, but are shared requirements for a range of cloud marketplaces. We assume that they can therefore also benefit from our contribution.

Current stage (Months 16-36). We derived implementation tasks from the requirements and evaluated the related work regarding its suitability to

provide the foundation for the cloud broker implementation. We identified six requirements, which allow pinpointing the deficiencies of the current state-of-the-art and the advancements presented by our approach. They consist of *functional* and *non-functional* requirements, as defined by ISO/IEC/IEEE 24765 [18]. The following requirements have been identified:

R1 (Functional): Capture service aspects pertinent to businesses. The cloud service formalization has to contain selection criteria which are pertinent to potential service consumers, as they use it for service selection on a marketplace.

R2 (Functional): Prevent redundant information. A new formalization has to prevent the introduction of redundant information by allowing to reuse existing description sources.

R3 (Functional): Support the service selection by cloud consumers. There are many SME service procurement agents which do not have extensive expertise in cloud computing. The formalization should therefore not only list selection criteria, but also support consumers in assessing the criteria fulfillment.

R4 (Non-functional): Low description effort. Any provider's description effort presents an entry barrier for cloud marketplace participation. Therefore, a new cloud formalization has to present measures to lower the efforts for describing services.

R5 (Non-functional): Low language definition effort. SaaS ecosystems have to adapt the service formalization to their audience, as only some selection criteria are universal. The cloud formalization has to provide a mechanism for easy adaptation to different scenarios.

R6 (Non-functional): Simple and usable tooling. Any software which is needed to interact with the formalization has to be usable with reasonable efforts by the respective target group, e.g., SME cloud service providers.

3 Related Work

This section presents the state-of-the-art in the area of our contribution and contrasts it to the previously stated requirements. Fulfillment or dissatisfaction of specific requirements are designated by $+$ and $-$ as a requirement number suffix. For partly fulfilled requirements, a $\circ$ is used. The end of this section summarizes the analysis of the related work.

3.1 Pertinent Selection Criteria for Businesses

Repschläger et al. have conducted two studies regarding the selection criteria of cloud consumers: In [28] they present "selection criteria for Software as a Service" based on literature review, an extensive market analysis, and an evaluation guided by expert interviews. The "Cloud Requirement Framework" (CRF) is outlined in [29]. It was devised using the same methodology and provides a well-grounded conceptual basis for structuring SaaS, PaaS, and IaaS selection criteria.

The "Cloud Service Check" [9] is a German catalog of cloud selection criteria with an extensive rationale providing guidance to assess different cloud service offerings. It is one of the results of the German research project Value4Cloud [10], whose application area are also SME cloud ecosystems.

The "Service Measurement Index" (SMI) is provided by the Cloud Service Measurement Initiative Consortium, led by Carnegie Mellon University [4]. It is far more extensive than related works. The SMI uses relative measures for assessing cloud offerings, i.e., "points" awarded for a service KPI are relative to the customer requirements and therefore differ for each respective service evaluation.

Analysis. The preceding works offer empiric knowledge whose incorporation into a solution design is the prerequisite for fulfilling Requirements 1 and 3.

3.2 Domain Specific Languages (DSLs)

Domain specific languages (DSLs) are tools to realize parts of computer systems. A comprehensive overview of DSLs is given by Fowler in [11] who defines DSLs as "a computer programming language of limited expressiveness focused on a particular domain". Fowler differentiates between *external* and *internal* DSLs: An external DSL is "external" to an implementation language, e.g., regular expressions, SQL, and XML. An internal DSL is "a particular way of using a general-purpose language", for example, a "fluid API" [7] and C++ template metaprogramming. Fowler recognizes a strong DSL culture in the programming language Ruby, which we chose for implementing our approach.

Analysis. According to Fowler, one of the main benefits of using a DSL is the domain specificity which fundamentally supports the communication with domain experts. DSLs can therefore help to fulfill Requirements 4 and 5.

3.3 High-Volume SaaS Marketplaces

Their revenue and the number of users make Salesforce AppExchange [32] and the Google Apps Marketplace [13] prototypical high-volume SaaS marketplaces. Instead of an elaborate cloud service formalization, they utilize data models with a small number of service attributes, such as free-text, images, provider info, and a categorization.

Analysis. Capturing information pertinent to businesses is solely dependent on the capability of the service provider to describe their services using unstructured information [R1$\circ$]. If providers participate in multiple marketplaces, the lack of a formalization introduces redundant information [R2$-$]. There is no support for consumers to assess criteria, as the information about selection criteria fulfillment has to be collected manually [R3$-$]. A guidance by a formalization is absent. Therefore, authoring service descriptions and enforcing a common structure between different descriptions is a non-trivial endeavor [R4$-$] [R5$-$]. The majority of contemporary software interacting with the simple data models of such marketplaces can be regarded as very mature, simple and usable [R6+].

3.4 Future Cloud Marketplaces and Service Description Languages

The features of future cloud marketplaces are postulated by Akolkar et al. in [2], who emphasizes on "intelligence" achieved by a solution repository and a deep question answering ability (DeepQA [17], such as the IBM Watson technology [16]). Akolkar refers to the work of Legner [21], who asserts the need for "more sophisticated classification schemes which reflect the vocabulary of the target customers". Our contribution contains such a vocabulary. To realize "intelligence" Akolkar proposes to use semantic technologies (e.g. OWL [45]) in a "vast knowledge base" using "recent advances in NLP, Information Retrieval, and Machine Learning to interpret and reason over huge volumes" [2]. There are recent research approaches sharing the mindset of Akolkar, such as FlexCloud (WSMO4IoS) and Linked-USDL:

FlexCloud (WSMO4IoS) The goal of the FlexCloud project is "developing methods and mechanisms for supporting a secure cloud lifecycle"[8]. It proposes a service description language, a registry, and a discovery system, which are presented by Spillner and Schill in [40]. The authors base their propositions on the analysis of twelve existing languages and their lack of meeting the authors' requirements of being "easily usable, freely available, versatile, extensible and scalable". Our requirement analysis yielded similar requirements. The authors created WSMO4IoS [39] which is based on one of the analyzed languages, the Web Service Modelling Language (WSML) [46]. Any work on WSML has ceased in 2008 having implications on the applicability of the work to current use cases. The usability of WSML tools is also impeded due to the outdated technological base of the WSMO IDE "WSMO Studio" [35].

Linked-USDL. Another recent proposal is the Linked-USDL, which is presented by Pedrinaci, Cardoso, and Leidig in [26]. It extends USDL proposed by Oberle et al. in [25]. According to Pedrinaci et al., USDL failed to gain adoption due to "complexity, difficulties for sharing and extending the model" [27]. USDL is based on the Eclipse Modeling Framework [43], Linked-USDL utilizes OWL [45] and reuses existing ontologies.

Analysis. Linked-USDL, WSMO4IoS, and our approach are motivated by the shortcomings of contemporary service description languages. Yet, we focus on supporting manual service selection by SME cloud consumers, while Linked-USDL and WSMO4IOS have a very broad scope: Linked-USDL aims "to maximise to the extent possible the level of automation that can be achieved during the life-cycle of services" [26]. WSMO4IoS has the goal of "covering as many XaaS domains as possible" and "unify these services as much as possible while restricting the domain specific service characteristics as little as possible"[40].

This broad scope and the lacking enterprise evaluation environment are the supposed reasons behind the failure of Linked-USDL and WSMO4IoS to capture most of the selection criteria identified in Section 3.1 [R1−]. Linked-USDL and WSMO4IoS cannot capture HTML descriptions [R2−] or support SME users in their cloud service selection [R3−]. Due to the expressive power of the utilized ontologies and the broad scope of both approaches, the description and language definition effort can be regarded as high in comparison to DSLs and JSON/XML-

based models, especially for SME users [R4−][R5−]. There are basic editors for Linked-USDL [22] and WSMO4IoS [38] which could be both reasonably used by SME users. Yet, the extension of both languages is only supported by ontology design software which targets expert users [R6○].

We have discussed both USDL and Linked-USDL with Leidig and Oberle to get design recommendations for our contribution.

3.5 Summary

The analysis of the related work clearly shows the lack of a solution fulfilling all of the enumerated requirements of SME cloud ecosystems. While there has been extensive research about pertinent selection criteria, almost none of them are reflected within service description languages, as well as contemporary and future cloud marketplaces. Furthermore, the benefits of using DSLs to capture service-domain specific information is not used in any of the analyzed contributions.

4 Formalizing Cloud Service Descriptions

Our contribution is divided into three main elements, which also structure this chapter: *The SDL-NG framework*[1] provides the metamodel and DSL framework for vocabulary definition and service description. A *pertinent business vocabulary*[2] uses the SDL-NG framework to formalize empiric knowledge about selection criteria. The *service broker*[3] builds upon the SDL-NG framework and the business vocabulary to provide a user interface to interact with the system.

4.1 SDL-NG Framework

The Service Description Language - Next Generation (SDL-NG) framework is built using the Ruby programming language to enable the specification of a service vocabulary as well as the description of services using this vocabulary in the form of domain specific languages.

The design of the SDL-NG framework is motivated by the requirements and the analysis of the related work: A DSL lowers description and language definition efforts (R4 & R5), especially in relation to service description languages using semantic technologies. From our past experience using Ruby as well as other programming languages we presume that a Ruby-based DSL can provide a simpler and more usable tooling (R6) than the language tools of WSMO4IoS and Linked-USDL. As the service description is in fact program code, it can be augmented by libraries to implement the integration of existing sources (R2), e.g., scraping HTML documents for feature descriptions. Another benefit of using Ruby is the low "syntactic noise", i.e., the overhead characters not conveying

[1] `https://github.com/TU-Berlin-SNET/sdl-ng`
[2] contained as `/examples/vocabulary` in the SDL-NG source
[3] `https://github.com/TU-Berlin-SNET/tresor-broker`

Listing 1.1: Example vocabulary definition

```
1  type :cloud_service_model
2  cloud_service_model :saas
3  cloud_service_model :paas
4  cloud_service_model :iaas
5  service_properties do
6     string :service_name
7     cloud_service_model
8  end
```

Listing 1.2: Example service description

```
1  service_name "Google␣Drive␣for␣Business"
2  cloud_service_model saas
```

semantic meaning (semicolons, brackets, etc.) are considerably lower in Ruby than in other programming languages, such as Java and C++.

The remainder of this subsection presents the SDL-NG *description life-cycle*, its *metamodel and type system*, as well as its *HTML parsing* and *export capabilities*.

Description life-cycle. The general life-cycle of service descriptions consists of the *vocabulary definition* and its *instantiation* in service descriptions. The main task of the vocabulary is to define the properties a service can have, their types, and their documentation. Vocabulary descriptions can be distributed over multiple files. An example SDL vocabulary description is shown in Listing 1.1. Line 1 defines a new `Type`: `CloudServiceModel`. Afterwards, some `CloudServiceModel` instances are defined. Starting from Line 5, new properties are added to the `Service` class: A string property named `service_name` and a property of the type `CloudServiceModel` with an auto-generated name `cloud_service_model`. Additional files contain the multi-lingual documentation of the vocabulary.

Listing 1.2 shows a service description, based on pairs of `Service` property names and their values. Setting the value of a property is in fact an invocation of a Ruby method on a `Service` instance. The predefined instances are referred by their identifier, e.g., the `saas` instance of `CloudServiceModel`.

Metamodel and type system. The SDL-NG metamodel and type system are based on Ruby classes forming a simple data model: `Type`s containing `Property` instances. Properties of `Type` instances are set to either one or many values, e.g., a `String` containing a service name, or other `Type` instances, e.g., the `Firefox` instance of the `Browser Type`. Any property value can be annotated to capture the "fine print" of service descriptions which cannot be modeled effectively using the vocabulary. Retrieving and setting properties is internally delegated to reflective methods, allowing custom persistence options, such as the default in-memory persistence and the MongoDB-based persistence used by the cloud broker. An SDL user does not need to know how to instantiate a specific Ruby class in order to set a property, as all classes are wrapped. These

Listing 1.3: HTML parsing with SDL-NG

```
1 fetch_from_url(<URL>[4], <CSS>[5]).each do |header|
2    feature header.content.strip, header.search('~p')[0]
3 end
```

wrappers allow simple class instantiation with strings and numbers, e.g., using the string `"1 $"` to create a money value instead of having the user know to write `Monetize.parse("1 $")`. Wrapped strings and numbers are retained in the wrappers to ease the serialization of property values.

HTML parsing. Preventing redundant information (R2) requires a simple mechanism to retrieve HTML SaaS descriptions. To ensure low efforts (R4) and simple tooling (R6), static service information and parsing commands should be defined in a self-sufficient service description. As SDL-NG descriptions are Ruby code, static information (e.g. `service_name "My service"`) as well as HTML parsing commands (e.g. `fetch_from_url(...)`) can be easily blended (e.g. `service_name fetch_from_url(...)`). Listing 1.3 illustrates the combination of HTML parsing and service description by showing the retrieval of `Feature` properties from the Google Drive for Business documentation. This example highlights the brevity, conciseness, and accessibility which can be achieved using language-internal DSLs. Besides statically reusing parts of websites for service description, the integration of external information can be highly dynamic in nature, e.g., when integrating spot market prices for cloud resources.

Export capabilities. To further prevent redundancies (R2), the vocabulary and descriptions should be consumable by other information systems. SDL-NG allows exporting descriptions as XML and the vocabulary as a corresponding XML Schema Definition (XSD) [36]. To support semantic processing, exporting descriptions to RDF is also possible. Generating a Web Ontology Language (OWL) [45] from the vocabulary is planned for one of the next releases.

4.2 A Pertinent Business Vocabulary

We have defined a vocabulary of selection criteria using the SDL-NG framework on the basis of empiric results from Repschläger et al. and the Cloud Service Check. It consists of 37 `Type` classes, 31 `Service` properties and 52 `Type` instances, covering a broad range of criteria, structured according to the Cloud Requirement Framework: **Characteristics**, e.g., cloud service model, service categories, **Charging**, e.g., charge unit (user account, floating license), **Compliance**, e.g., data location, audit options (e.g. audit logging), **Delivery**, e.g., billing and payment options, **Dynamics**, e.g., the duration for provisioning an end user, **Interop**, e.g., features, interfaces, and compatible browsers, **Optimizing**, e.g., maintenance windows and future roadmaps, **Portability**, e.g., exportable and importable data formats, **Protection**, e.g, the communication protection (HTTPS, VPN, TCTP), **Provider management**, e.g., support availability, **Reliability**, e.g., offline capabilities, **Reputation**, e.g., year of service establishment, and **Trust**, e.g, providers' financial statement, reference customers.

To achieve low description efforts (R4), `Service` properties are designed for high readability, e.g., "`employs 49829`", or "`is_protected_by https`".

4.3 The Service Broker

The service broker is part of the TRESOR ecosystem and offers an authoring interface for service descriptions, a repository query interface, cloud consumer search profiles, management of description versions, a description update workflow, and a module for booking and provisioning third party cloud services. The broker is built on the basis of the Ruby on Rails web framework [14] and uses MongoDB [24] for description persistence. To meet Requirements R4 and R6 (low description effort and simple tooling) the service broker includes a customized version of the Ace JavaScript editor [1] to allow comfortable authoring of service descriptions. To guide description authors, the broker can present a comprehensive listing of all properties and predefined instances of the business vocabulary in the form of a "cheat sheet". It is covered by 102 test cases, which cover 93,5 % of source lines of code.

5 Assessment and Outlook

We carried out an *analytical evaluation*, contrasting our contribution to its requirements, an *experimental evaluation*, where we applied the SDL-NG framework in an SME setting, and a preliminary *empirical evaluation*.

Analytical Evaluation. To ensure *capturing service aspects pertinent to businesses(R1)* we extensively and thoroughly compiled requirements from SME stakeholders, included empiric studies in our work, discussed it with the authors of the Cloud Requirement Framework and the Cloud Service Check, and participated in expert panels to assure its relevance for different use cases. To *prevent redundant information (R2)* the SDL-NG framework can integrate HTML descriptions as well as export them to XML/XSD and RDF. We *support the service selection by cloud consumers (R3)* by including rationale and extensive property documentation. By relying on text files containing property-value pairs SDL-NG presents *low description effort (R4)*. The language definition also relies on a simple textual syntax, leading to *low language definition effort (R5)*. Our *tooling is simple (R6)*, having a simple meta- and data model and consisting of only 1.200 source lines of code.

Experimental evaluation. The SDL-NG source repository contains descriptions of two widely used SaaS services, Google Apps for Business and Salesforce Sales Cloud. These descriptions test the vocabulary pertinence and simplicity and also present a trial of the integration of external HTML content, as well as the XML/XSD and RDF export functionality. So far, we did not encounter challenges in the application of the SDL-NG framework.

Empirical evaluation. While designing our contribution we regularly presented preliminary stages to different project and external partners to carry out

short explorative expert interviews to gain their first impressions on the respective state of our contribution. To test the usability of our approach in the SME domain we will carry out user studies at the end of the project.

Outlook. The outlook of our contribution is the "Open Service Compendium": by refining the vocabulary, creating an easy to use web interface, and opening up the service broker, we want to create a crowd-sourced "Wikipedia for Services", where any user can describe any service and use the broker to support their service selection. Four additional challenges have been identified which need to be addressed to create such a system:

Modeling service variants. Modeling the large amount of possible combinations of SaaS offering's variants and optional features is not feasible. The Open Service Compendium will use a *feature model* [19] based variant modeling. The features would contain `Service` property values, so that any offering description can be created through selecting features and combining properties.

Price modeling and calculation. Contemporary price models and ontologies are either lacking in sophistication, are too complicated, or cannot capture existing SaaS services. The Open Service Compendium should capture the price models of common SaaS offerings to the degree that potential service consumers can easily estimate their consumption charges. By combining the price and feature model, complex charging rules can also be supported.

Property brokering information. The Service Compendium should be able to assess the tendency of property values, i.e., if a value is better or worse than another value. This provides the basis for implementing analytic processes, such as the analytic hierarchy process [30] and the analytic network process [31] to support users with their service selection. We also want to include the findings of the CSMIC [4] to automate the measurement of service selection criteria fulfillment.

Dynamic vocabulary. To keep usage efforts of large vocabularies low, the vocabulary should be dynamically adaptable. For example, it should be possible to define category-specific properties. Furthermore, the definition of derived properties, e.g., "Cloud Storage per Euro" could help service comparison considerably.

6 Conclusion

Finding suitable cloud services is a laborious task, especially for SMEs. To support SMEs with their service selection better than current approaches we have created a novel DSL, a pertinent business vocabulary, and a brokering component. As we apply and evaluate the components in an appropriate SME environment we ensure their *relevance*. We have designed them *rigorously*, building upon empirical studies and established state-of-the-art. By publishing our work as open source software and by continuously engaging with the service community we want to further advance our work towards an "Open Service Compendium", which can support a broad range of cloud consumers in their service selection.

References

1. Ajax.org B.V.: Ace (2014), http://ace.c9.io
2. Akolkar, R., Chefalas, T., Laredo, J., Peng, C.S., Sailer, A., Schaffa, F., Silva-Lepe, I., Tao, T.: The Future of Service Marketplaces in the Cloud. In: 2012 IEEE 8th World Congress on Services. pp. 262–269 (2012)
3. Atlantic Systems Guild Ltd.: Volere Requirements Specification Template (2014)
4. Carnegie Mellon University: CSMIC: Cloud Service Measurement Initiative Consortium (2012), http://csmic.org/
5. Ermakova, T., Fabian, B., Zarnekow, R.: Security and Privacy System Requirements for Adopting Cloud Computing in Healthcare Data Sharing Scenarios. In: Proceedings of the AMCIS 2013 (2013)
6. Ermakova, T., Huenges, J., Erek, K., Zarnekow, R.: Cloud Computing in Healthcare – A Literature Review on Current State of Research. In: Proceedings of the AMCIS 2013 (2013)
7. Evans, E.: Domain-Driven Design: Tackling Complexity in the Heart of Software. Addison-Wesley (2003)
8. FlexCloud: FlexCloud (2014), http://flexcloud.eu/
9. fortiss GmbH: CloudServiceCheck (2014), http://www.value4cloud.de/de/cloudservicecheck
10. fortiss GmbH: Value4Cloud (2014), http://www.value4cloud.de
11. Fowler, M.: Domain-Specific Languages. Addison-Wesley (2011)
12. German Federal Ministry for Economic Affairs and Energy: Trusted Cloud (2014), http://www.trusted-cloud.de/
13. Google: Google Apps Marketplace (2014), https://www.google.com/enterprise/marketplace/home/apps/?pli=1
14. Heinemeier Hansson, D.: Ruby on Rails (2014), http://rubyonrails.org
15. Hevner, A.R., March, S.T., Park, J., Ram, S.: Design Science in Information Systems Research. In: Gupta, A. (ed.) MIS Quaterly, vol. 1, pp. 75–105 (2004)
16. IBM: Watson (2014), http://www.ibm.com/smarterplanet/us/en/ibmwatson
17. IBM Research: The DeepQA Research Team (2013), http://www.research.ibm.com/deepqa
18. ISO/IEC/IEEE: Systems and software engineering - Vocabulary (2010)
19. Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E., Peterson, A.S.: Feature-Oriented Domain Analysis (FODA): Technical Report (1990)
20. Lacity, M., Reynolds, P.: Cloud Services Practices for Small and Medium-sized Enterprises. In: MIS Quarterly Executive, vol. 13:1, pp. 31–44. Management Information Systems Research Center, Minneapolis (2014)
21. Legner, C.: Is There a Market for Web Services? In: Di Nitto, E., Ripeanu, M. (eds.) Service-Oriented Computing - ICSOC 2007 Workshops, Lecture Notes in Computer Science, vol. 4907, pp. 29–42. Springer, Berlin, Heidelberg (2009)
22. Leidig, T.: Simple editor for Linked USDL descriptions (2013), https://github.com/linked-usdl/usdl-editor
23. Mell, P., Grance, T.: The NIST Definition of Cloud Computing (2011)
24. MongoDB, Inc.: mongoDB: Agile and Scalable (2014), http://www.mongodb.org
25. Oberle, D., Barros, A., Kylau, U., Heinzl, S.: A unified description language for human to automated services. Information Systems 38(1), 155–181 (2013)
26. Pedrinaci, C., Cardoso, J., Leidig, T.: Linked USDL: A Vocabulary for Web-Scale Service Trading. In: Presutti, V., d'Amato, C., Gandon, F., d'Aquin, M., Staab, S., Tordai, A. (eds.) The Semantic Web: Trends and Challenges, Lecture Notes in Computer Science, vol. 8465, pp. 68–82. Springer International Publishing (2014)

27. Pedrinaci, C., Cardoso, J., Leidig, T.: Presentation: Linked USDL: a Vocabulary for Web-scale Service Trading (2014), http://slideshare.net/cpedrinaci/linked-usdl-a-vocabulary-for-webscale-service-trading
28. Repschläger, J., Wind, S., Zarnekow, R., Turowski, K.: Selection Criteria for Software as a Service: An Explorative Analysis of Provider Requirements. In: Proceedings of the AMCIS 2012 (2012)
29. Repschläger, J., Zarnekow, R., Wind, S., Klaus, T.: Cloud Requirement Framework: Requirements and Evaluation Criteria to adopt Cloud Solutions. In: Pries-Heje, J., Chiasson, M., Wareham, J., Busquets, X., Valor, J., Seiber, S. (eds.) Proceedings of the 20th European Conference on Information Systems (2012)
30. Saaty, T.L.: What is the Analytic Hierarchy Process? In: Mitra, G., Greenberg, H.J., Lootsma, F.A., Rijkaert, M.J. (eds.) Mathematical Models for Decision Support, NATO ASI Series, vol. 48, pp. 109–121. Springer-Verlag (1988)
31. Saaty, T.L.: Analytic network process. In: Encyclopedia of Operations Research and Management Science, pp. 28–35. Springer US (2001)
32. Salesforce: AppExchange (2014), https://appexchange.salesforce.com
33. SAP AG: USDL Marketplace (2012), http://sourceforge.net/projects/usdlmarketplace
34. SAP AG: FI-Ware Marketplace and Repository Reference Implementation (2013), https://github.com/service-business-framework
35. Simov, A., Dimitrov, M.: WSMO Studio (2008), http://sourceforge.net/projects/wsmostudio/files
36. Sperberg-McQueen, C.M., Thompson, H.: XML Schema (2014), http://www.w3.org/XML/Schema
37. Spillner, J.: SPACEflight — A versatile live demonstrator and teaching system for advanced service-oriented technologies. In: IEEE (ed.) Proceedings of the 21st CriMiCo. pp. 455–456. IEEE (2011)
38. Spillner, J.: wsmo4ios-editor (2012), http://serviceplatform.org:8000/trac/browser/packaging/scripts/develtools/wsmo4ios-editor
39. Spillner, J.: WSMO4IoS (2013), http://serviceplatform.org/spec/wsmo4ios/
40. Spillner, J., Schill, A.: A Versatile and Scalable Everything-as-a-Service Registry and Discovery. In: Desprez, F., Ferguson, D., Hadar, E., Leymann, F., Jarke, M., Helfert, M. (eds.) CLOSER 2013 Proceedings. pp. 175–183. SciTePress (2013)
41. Thatmann, D., Slawik, M., Zickau, S., Küpper, A.: Deriving a Distributed Cloud Proxy Architecture for Managed Cloud Service Consumption. In: CLOUD 2013 Proceedings. pp. 614–620. IEEE (2013)
42. Thatmann, D., Slawik, M., Zickau, S., Küpper, A.: Towards a Federated Cloud Ecosystem: Enabling Managed Cloud Service Consumption. In: GECON 2012 Proceedings. Springer-Verlag, Berlin, Germany (2012)
43. The Eclipse Foundation: Eclipse Modeling Framework Project (EMF) (2014), http://www.eclipse.org/modeling/emf
44. The European Commission: A Recovery On The Horizon: Annual Report on European SMEs 2012/2013 (2013)
45. W3C OWL Working Group: OWL 2 Web Ontology Language Document Overview: W3C Recommendation 11/12/2012 (2012), http://www.w3.org/TR/owl2-overview
46. WSML Working Group: Web Service Modeling Language (2008), http://www.wsmo.org/wsml
47. Zickau, S., Küpper, A.: Towards Location-based Services in a Cloud Computing Ecosystem. In: Ortsbezogene Anwendungen und Dienste, pp. 187–190 (2012)