Engineering Service Level Agreements: A Constrained-Domain and Transformation Approach

Josef Spillner, Stefan Illgen and Alexander Schill
Faculty of Computer Science, Technische Universität Dresden, 01062 Dresden, Germany
{josef.spillner,alexander.schill}@tu-dresden.de, stefan.illgen@mailbox.tu-dresden.de

Keywords: Service Level Agreements, Service Engineering

Abstract: Service Level Agreements (SLAs) are fundamental technical-juridical documents which in their function as contract govern the rights and obligations between service consumers and providers. In today's growing service ecosystems, individually negotiated SLAs have powerful additional roles such as serving as the input for automated service monitoring and health checking as well as binding service consumers through custom incentives. These advantages, in turn, require adequate engineering techniques to let service providers express the conditions under which a service can be consumed. While many SLA languages and tools exist, existing approaches are either severely limited or too complex to be used by a broad set of providers. Hence, we present a reduced effort approach based on transformations from existing domain-constrained public service descriptions. By demonstrating the application thereof in an Infrastructure-as-a-Service scenario, we show that SLAs can be prepared and improved in a very short time with suitable tools.

1 INTRODUCTION

Services brokered over the Internet, or even delivered through the Internet, are one of its most important functions beside the access to information and communication channels. Custom mobile applications, attractive web applications and flexible enterprise integration systems as human and technical interfaces to these services are considered the state of the art. Recently, cloud computing services along with on-demand provisioning and pay-as-you-go billing schemes have further increased the service utility and have been driving their wide-scale adoption (Armbrust et al., 2010). Most services on the Internet today can be categorised into one of the two big groups: Free services with no guarantees at all (called service level objectives, denoted as set O) or little best-effort protection and no compensation (denoted as percentage C) in the presence of violations (denoted as condition V), and paid services with a range from no guarantees to full compensation and even over-compensation for failures. Examples include the Hewlett-Packard compute cloud ($O = \{availability\}$, $V = \{availability \geq 99\%\}$, $C_{max} = 30\%$), Google Apps ($O = \{uptime\}$, $V = \{uptime \geq 95\%\}$, $C_{max} = 50\%$), the Google search engine ($O = \emptyset$, $C = 0$) and the DHL express delivery ($O = \{executiontime\}$, $C = 100\%$). The objectives, violation rules and compensation levels are captured as terms in legally binding agreements called Service Level Agreements (SLAs), sometimes appearing in the context of Terms of Service (ToS) for both technical and non-technical services (Mendes and da Silva, 2012). The conditions, monitoring periods and usage constraints under which the full or partial compensation applies are often manifold (Kotsokalis et al., 2011; Stantchev and Schröpfer, 2009) so that we omit their introductory discussion.

Beside these two groups with their fixed binary policy of either accepting the public SLA or not using the service, especially in the academic world and in privileged business contexts individually negotiated SLAs play a major role. These are often based on SLA templates (SLATs), either textual with natural language or of formal structure or a mixture of both, which are converted into SLA offers (SLAOs) during the negotiation process. In order to support service providers to express their fixed SLAs and their negotiable SLATs from the first guarantee plans until the submission to a web site or an open service market, SLA languages have been created and basic SLA engineering and processing tools exist (Alhamad et al., 2010; Kearney et al., 2010). There are also a number of general methodologies for service engineering and provisioning in open markets which involve SLAs in addition to other descriptive and executable service

artefacts (Spillner et al., 2011).

However, up until now, there has been little research on an effective SLA engineering in the context of open service markets with potentially thousands of available services. In particular, novice providers often need many hours to even start modelling their first SLA due to the complexity of both the languages and the tools. Even seasoned providers experience consistency problems between SLA documents and other information due to the low integration level of SLA engineering tools and between the tools and marketplaces. Hence, we propose a much more refined and guided approach based on functional domain constraints and automated transformations which cuts down the engineering time significantly, while still being applicable to generic services not specifically optimised for, and ensures consistency. We evaluate the approach with an editor which offers many integration points to reduce the engineering time even more.

In the next sections, we first describe previous SLA engineering methodologies in detail, including existing languages and tools. Then, we present our approach and focus on the novel aspects of domain constraints and transformations, as well as on the integration into engineering and provisioning toolchains for a reduced time to service markets. Afterwards, we validate our approach by observing a practical modelling experiment and conclude with remarks about anticipated future improvements.

2 SLA ENGINEERING METHODOLOGIES

In order to establish an SLA as a technically valid and legally binding document between a service consumer and a service provider or a service hoster acting on behalf of the provider, several actions need to be performed. In the case of fixed public SLAs, the SLA needs to be expressed as a document for a certain service or a group of services, then bound to each service by linking it from a webpage or registering it in a directory service or on a market. It becomes a binding document by the confirmation from the user. In the case of negotiable SLAs, rules for the negotiation need to be established including the option to negotiate manually, and a template for the SLA with placeholders for the negotiable parts need to be expressed. The template (SLAT) needs to be bound to the service and registered just like the SLA in the fixed case. The non-public SLA is derived in a negotiation process after which it becomes a binding document. Afterwards, SLA evaluation and processing actions happen, for instance monitoring, violation and dependency checking, evolutionary improvement and feedback. These actions are not subject to this work; we refer to (Wu and Buyya, 2010) for an extensive survey on overall SLA lifecycle components.

Beside ad-hoc and copy-paste approaches to establish SLAs, a number of systematic methodologies to create them from scratch for a certain service have been proposed in the literature (Spillner et al., 2011). These methodologies typically suggest a combination of preferred SLA language, modelling and editing tools and market integration tools. The SLA engineering scope as part of a service engineering and provisioning process is shown in Figure 1. In it, a service engineering environment under the control of a service engineer hosts editors for all parts of a service package, including an SLA editor. The service package, an abstract model for the purpose of combining executable and descriptive service artefacts, is then deployed on an open service platform on the Internet or an in-house test stage platform. The executable artefacts are then deployed in a service hosting repository, whereas all descriptive artefacts, among them SLA templates, will be registered at a service broker. At runtime, the SLA editor may fetch feedback information from the broker and incorporate it as part of an SLA template re-engineering process. All the languages and tools of relevance for SLA engineering and provisioning will be briefly presented and then summarised with a list of shortcomings.

2.1 SLA Languages

An SLA is a time-limited agreement about a specific service jointly enacted between the service provider and the service consumer which determines the overall usage conditions and the targets for the cost and quality of service execution and delivery. The complexity of SLA documents motivates the need for custom languages for the proper formal expression of the conditions and targets.

Languages to describe SLAs and SLA-related documents including templates and offers have originated from the web services and grid computing communities. WSLA (Keller and Ludwig, 2002), WS-Agreement and its extensions (Andrieux et al., 2007; Frankova et al., 2006), SLAng (Lamanna et al., 2003), NextGRID (Mitchell and McKee, 2006) and RBSLA (Paschke, 2005) are examples for such languages.

The Web Service Level Agreement (WSLA) language is an XML-based format specifically designed for web services. WSLA further defines a negotiation, monitoring and notification framework tightly coupled to the language. Similar to WSLA, the lan-

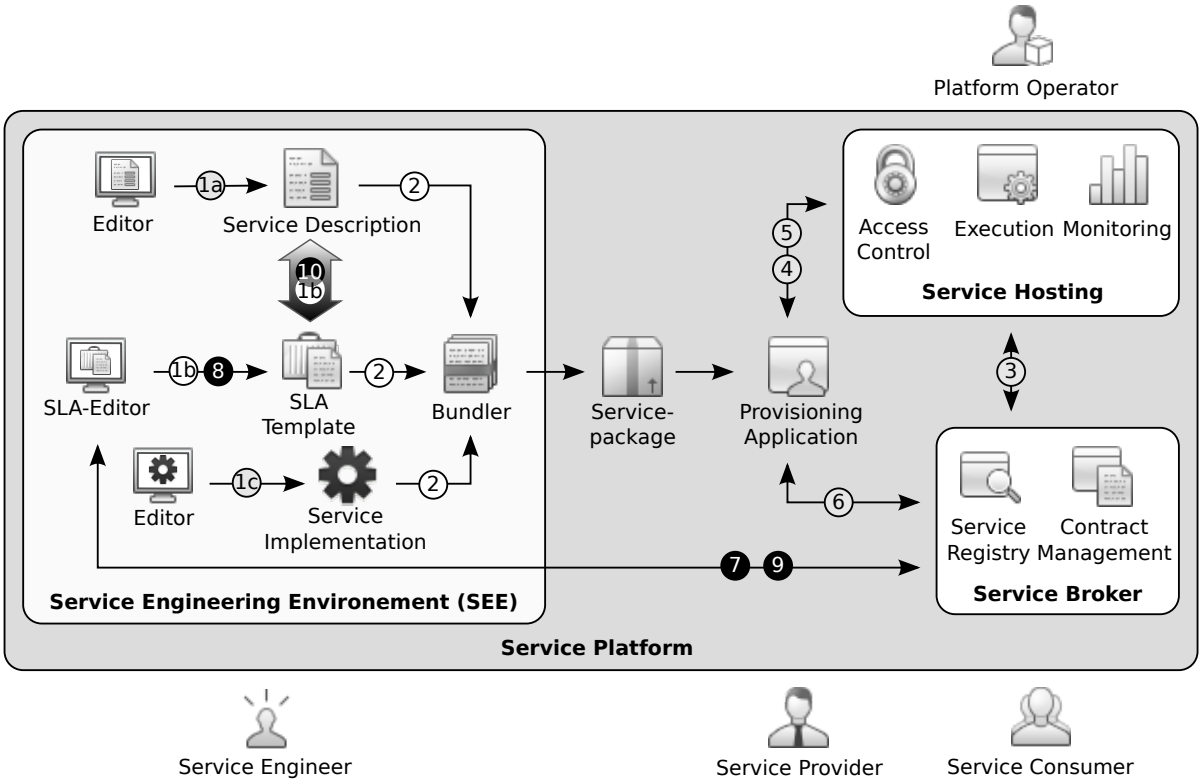

Figure 1: Engineering and provisioning toolchain for services in open markets

guage for Web Service Agreements (WS-Agreement) is XML-based with a structure defined by XML Schema. Beyond the language itself, the WS-Agreement framework defines web service operations to perform negotiation and monitoring tasks. As the language is very generic, a number of extensions have been proposed to introduce specialised sets of guarantees and to improve the negotiation protocols (Frankova et al., 2006; Jouve et al., 2006). SLAng is defined on an abstract modelling level in terms of meta-models in Essential Meta-Object Facility (EMOF) notation. It applies the Object Constraint Language (OCL) to specify quality constraints. NextGRID's language is another one defined by an XML Schema. Such SLAs contain three parts for the definition of participating parties, the negotiable properties and fixed statements, respectively. This language is heavily influenced by WS-Agreement. RB-SLA is a declarative rule language which enhances the XML-serialised RuleML language with useful constructs to express SLA terminology. An advantage is that off-the-shelf RuleML engines already exist for the processing of such SLAs.

Recently, there have also been proposals for a conceptual SLA model for cloud services (Alhamad et al., 2010), and one for generic services called SLA⋆ (Kearney et al., 2010). Nevertheless, for our work we chose WS-Agreement due to the well-understood scope and limitations and the existence of extensions and tooling. This choice, however, doesn't render our approach inapplicable with other languages, especially because most of them are quite similar in their syntax and semantics.

2.2 SLA Engineering Tools

Tooling around SLAs involves the tasks of SLA and SLAT creation through modellers and editors, the registration of these documents at markets, SLA negotiation before using a service, and SLA monitoring, processing and evaluation at runtime. Our work focuses purely on the engineering perspective and hence on the first two tasks. Editing and modelling tools for SLAs exist mainly for the Eclipse IDE, namely WSAG Editor for WS-Agreement, PLASTIC SLA Editor for SLAng, and g-Eclipse as rather generic approach for grid environments (Spillner et al., 2011; Bertolino et al., 2007; Gjermundrød et al., 2008). Web-based tools also exist, namely ADA and Dragon (Müller et al., 2010; Boissel-Dallier et al., 2009).

The WSAG Editor is an extensible rich client application to model SLAs, SLATs and SLAOs in

pure and extended WS-Agreement. Each part of the SLA-related documents can be manipulated with a dedicated editor component, which is either a default XML text editor or a more suitable graphical widget. Due to the binding to XML Schema, the widgets can be reused for common data types, e.g. SLA participant addresses or pricing terms. An integrated help system and a number of wizards make this editor more user-friendly. The scope of the PLASTIC SLA Editor is slightly different. It integrates a framework to generate monitors for the objectives defined in the SLAng language. The editor part exposes a structural UML modelling tool but doesn't offer further graphical support. Model-to-model and model-to-code transformations are available. G-Eclipse offers views, editors and wizards to manage grid resource services. SLAs can be generically modelled and published. Due to the focus on grid environments, both a Job Submission Description Language (JSDL) and a NextGRID SLA editing component are contained in g-Eclipse. A wizard exists to generate new SLAs based on the service domain selection, but it doesn't import non-functional properties from a service description. The Agreement Document Analysis (ADA) project has proposed WSAG4People, a human-readable syntax variant of WS-Agreement, along with a web-based editor called iAgree which has been implemented in JavaFX. The editor is however limited to consistency checks between SLA templates and offers while the modelling itself requires direct syntax modifications. Dragon is a SOA governance tool which incorporates a registry along with contracting tools and editors for WSDL descriptions and SLA documents. SLAs are modelled in WS-Agreement which are linked to WSDL documents. The editor targets administrators rather than providers, as there is no conceptual differentiation between SLATs and SLAOs.

As mentioned before, sometimes SLA engineering tools have been anticipated as part of general service engineering projects. Representative for these are a toolchain to connect and orchestrate separate light-weight modelling tools (Spillner et al., 2011), an integrated development environment with publishing functionality called the ISE Workbench (Scheithauer et al., 2011) and the SeCSE Suite, a repository of open source instruments to create service-centric systems (Baresi et al., 2006). In all of these cases, the guided SLA modelling workflow is very basic and less sophisticated than in the SLA-centric approaches mentioned before. For example, SeCSE assumes the expression of SLA conditions as commercial or QoS facets of a custom service specification language without offering specific support for expressing the facets.

Overall, formalisation and tool support for SLA engineering is available, but the state of the art is rather weak. In particular, among the editors, only the WSAG Editor seems to be maintained and up-to-date with current Eclipse versions. Furthermore, all of the editors require significant effort and still expose syntactical structures which make them unsuitable for the casual or novice service provider. Only the g-Eclipse editor offers integration with grid or cloud service hosting facilities. As a result, we choose to take the WSAG Editor and advance it with user-friendly modelling techniques and seamless service marketplace integration in order to demonstrate the usefulness of these techniques.

For the completeness of the tools analysis, it should be mentioned that a number of SLA frameworks exist beyond SLA engineering without graphical user interfaces or even no human interfaces at all, for instance to support negotiation and monitoring. A client-centric negotiation tool as well as the Akogrimo and SLA@SOI architectures are examples for such tools which run non-interactively on service platforms (Rady, 2012; D'Andria et al., 2007; Butler et al., 2011).

2.3 Limitations

The amount of available SLA engineering methodologies, languages and tools analysed in the previous sections shows that there is a common understanding of the importance of structured SLAs and the need to engineer them properly. Nevertheless, certain weaknesses could be identified which shall be summarised here to use them as input for the requirements analysis leading to a more suitable approach.

Most approaches only offer a generic SLA modelling perspective without taking specialised knowledge about certain domains into account. In reality, most service providers have a strong domain-specific background (Gillam et al., 2012). Furthermore, none of the approaches covered by the analysis strives for ensured consistency between modelling artefacts. Inconsistencies can become a major problem if service descriptions and SLA documents do not match or if there are different versions of them on the engineer's side and published on open markets.

3 PROPOSED APPROACH FOR SLA ENGINEERING

The proposed SLA engineering approach makes use of functional service domain constraints. Do-

main knowledge is often higher than technical or juridical knowledge with service providers, for instance knowledge about the intrinsic priorities in short-term, time-critical travel planning services or long-term, safety-critical cloud storage services. It is implied that the modelling and editing widgets are aware of domain-specific guarantees and can thus be presented to the user in a suitable form. The approach also relies heavily on transformations so that metadata and non-functional properties, including technical abilities and business context like pricing information, can be imported from service descriptions in order to speed up the modelling process. The import function is logically complemented by an export function which contributes further to the increased consistency and decreased modelling time. In the following paragraphs, the SLA model supported by the approach is specified, and all three distinct characteristics of the approach are explained in detail.

3.1 SLA Model

Our engineering approach assumes a flexible model which is aligned with existing specifications so that tools and services can be reused. In analogy with the WS-Agreement specification, an Service Level Agreement document (SLAT, SLAO or final SLA) consists of a number of Service Level Objectives (SLOs) together with auxiliary information about the agreement participants, the validity and standard legal terms. As a minimum requirement, the objectives are described as triples (property, predicate, metric(value, unit)). The property corresponds to a Non-Functional Property (NFP) of the service, which may be either generic or domain-specific. Some SLOs may be subject to a precondition, a Qualifying Condition (QC), which are triples referring to environmental properties in the service execution context and thus are converse to the objectives which refer to intrinsic service properties. Furthermore, Business Value Lists (BVLs) are used to define precise compensation and weighting among the SLOs. An example for an SLA which contains SLOs, QCs and BVLs is shown in Table 1. The objective values can be ranges instead of specific values, which may be flattened to smaller ranges or even single values during the offer creation (ratio scale). Alternatively, they can be lists from which a specific value may be picked (nominal scale). Other scale types are in between these two and could be modelled accordingly, although we don't consider them for our work.

3.2 Constrained Domains

The approach assumes that all services for which SLAs should eventually be made available fall into a functional domain and express this domain through a description document. Typically, semantic web service languages such as the Web Ontology Language for Services (OWL-S) and the Web Service Modelling Language (WSML) are used to create these documents. The Unified Service Description Language (USDL) also offers vocabulary to specify the service domain. Domains can be specialised subdomains of one or more generalised domains.

The ontology set shown in Figure 2 shows how generic base ontologies and domain-specific ontologies for cloud computing services can be combined to yield expressive semantic descriptions. Higher-level concepts such as cloud storage capacity or compute cloud burst mode are clearly domain-specific. For these concepts, an appropriate modelling support can be visually given by loading appropriate widgets depending on the editor implementation. Whenever no such widget exists, a generic representation down to the basic data types present in all ontology languages is always possible as a fallback. Nine base ontologies and numerous domain ontologies are contained in the WSMO4IoS catalogue which focuses especially on non-functional properties of services from various classes and domains (Spillner, 2012). Likewise, similar catalogues exist for common cloud services as in IaaS, PaaS and SaaS (Rady, 2012). For instance, if the provider intends to model an SLA template for a cloud storage service, the specific properties listed in Table 2 will be queried and turned into SLOs beyond generic business, network and service QoS properties. This avoids the specification of SLOs which are not of relevance to the particular kind of service.

Table 2: SLOs suitable for a cloud storage service

Property	Representation	Source Domain
Sign-up privacy	Slider (%)	Online Account
Capacity	Slider (Unit)	Storage
Redundancy	Slider (%)	Storage
Backup	Option	Storage
Encryption	Option	Storage

3.3 Transformations

In order to reduce the engineering effort and the potential for inconsistencies to appear, the service level objective terms in SLAs and SLA templates should be linked to the corresponding service description documents. There are two options: A link on the syntactic/interface level to mark which web service oper-

Table 1: Example for an SLA with objectives (SLOs), preconditions (QCs) and weighting (BVLs)

Service Level Objective (SLO)			Qualifying Condition (QC)			Business Value (BVL)
NFP	Predicate	Metric	NFP	Predicate	Metric	Weight
Domain-independent (Generic)						
Availability	$\geq$	99.9%				5
ResponseTime	$<$	40ms				3
Throughput	$\geq$	100Mbps	NetworkBandwidth	$\geq$	100 MBps	1
Domain-specific: Cloud Storage						
Capacity	$>$	9999GB				1
Redundancy	$\geq$	200.00%				1

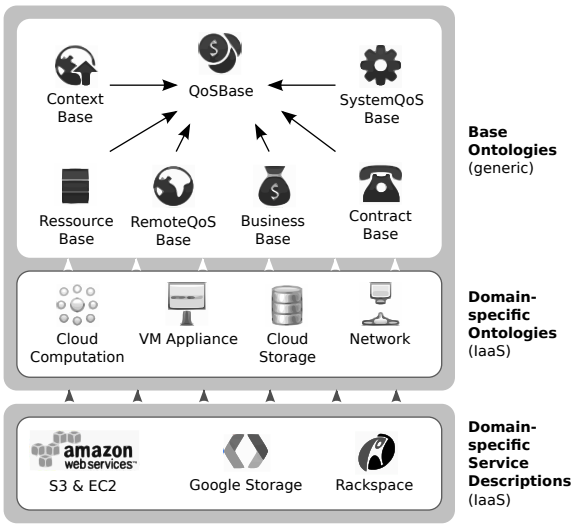

Figure 2: WSMO4IoS service models for cloud computing infrastructure services

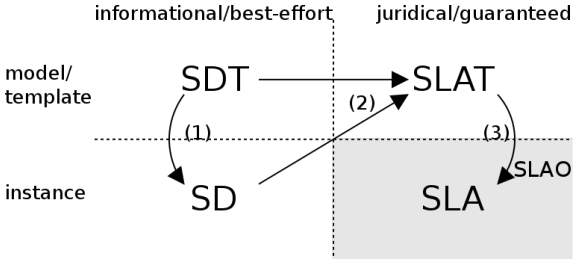

Figure 3: Transformations between models and instances (vertical) and between informational and juridical descriptive service artefacts (horizontal)

ations or resources will be protected, to be defined in a Web Service Description Language (WSDL) or Web Application Description Language (WADL) document, and a link on the semantic/conceptual level to mark which service concepts, characteristics and properties will be protected, to be defined by USDL, WSML or OWL-S as previously outlined. Our transformation work focuses on non-functional property entries in service descriptions with WSML semantics.

Figure 3 puts all five SLA-related descriptive service artefacts and their three transformation options into perspective. The approach assumes that in order to transform a service description into an SLA template (step 2), a rich service description (SD) exists which may have been derived from a description template (SDT) in the form of base and domain ontologies beforehand (step 1). Subsequently, the template can be negotiated into an offer and eventually into a valid SLA while maintaining the links as annotations (step 3).

According to the SLA model, each SLO is defined by a variable with a required value and, if dynamically added by a service runtime, a current value as meta-data. The target format of the variables can follow either an fully contained non-functional property model or a schema-driven one with external expressions. The Java Expression Language (JEXL) is a language to define logical statements with comparison operators over the variables which are themselves defined according to a model of non-functional properties such as the one defined by the Job Submission Description Language (JSDL). JEXL expressions are rather complex and become hard to maintain when several of them are used in an SLA document. The fully contained model defines all values within each of the SLO terms, without using WS-Agreement's `Location` element. It is used by, among others, the Value-Types extension to WS-Agreement (WSAG+VT). Specifically, WSAG+VT allows for fixed values, ranges and options with appropriate comparison operators such as `less-than` and `out-of-range`. Figure 4 shows how in this context the WSAG+VT extension compares to JEXL expressions. Our approach assumes an extended WSAG+VT with ontology grounding to maintain the link to the original service description. The grounding XML elements extend the SLO variable's `@Metric` attribute. A downside of this decision is that the full conformance to the WS-Agreement specification is lost, although this is more a theoretic concern than a practical one and traded off with much better support for extensibility without having to modify schemas. In both cases, WS-Agreement `GuaranteeTerms` are used to hold the objectives

while `ServiceProperties` hold the variables.

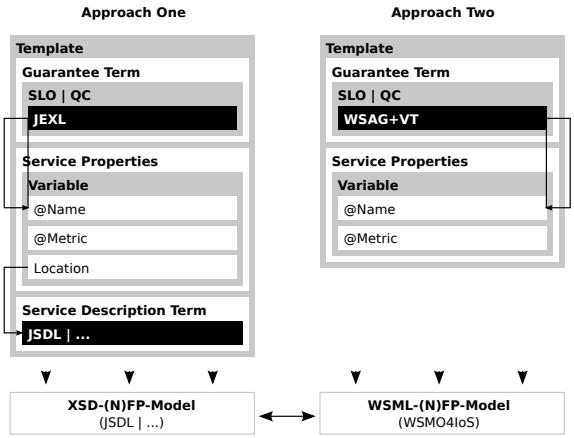

Figure 4: Comparison of JEXL and WSAG+VT

Figure 5 contains a rough outline of the transformation rules between a service description expressed in WSML and the target SLA template expressed in WS-Agreement. WSML interfaces are referenced from within SLA documents by WS-Addressing statements. Monitoring endpoints can similarly be defined. Non-functional properties are converted from WSML NFPs into WSAG+VT SLOs while pricing specifications are considered for the definition of penalties. For them, a custom WSAG+Business extension has been defined based on the WSMO4IoS and USDL pricing models which include currency and price region concepts. Finally, further metadata is converted, such as the provider-side participant address from the WSMO4IoS contract model properties to the corresponding WS-Agreement element `AgreementResponder`, while the `AgreementInitiator` is filled into negotiated SLAs at runtime. The transformation quality depends on the presence of specialised service level meta-properties in the service description. One kind of meta-property is the declaration of whether an NFP can be guaranteed or not, with the estimated guarantee levels. Without the meta-properties, a fallback transformation takes place. Generally, these meta-properties cause more service description modelling overhead but have additional advantages, for instance filtering out services without certain guarantees already in the service selection phase before attempts to negotiate a contract.

3.4 Integration

An SLA editor is subject to integration with both other service modelling tools and with service registries in open markets. The integration is primarily document-based, where the documents are descriptive

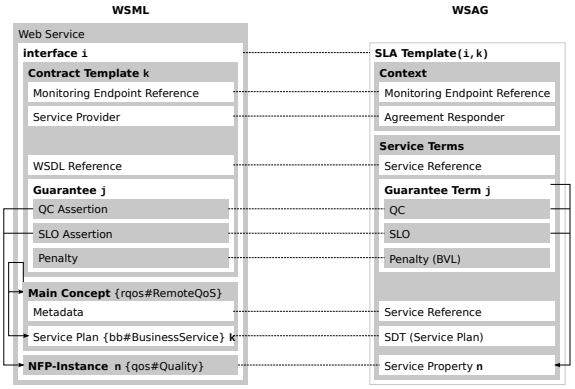

Figure 5: Transformation rules between WSML service descriptions and WS-Agreement SLA templates

service artefacts consumed or produced by the editor. The following combinations are useful:

- Import of service description document from tool: The SLA editor reads the service descriptions, transforms it into a stub SLA and lets the user perform the modelling process.

- Import of service description document from market: Similarly, a service description is loaded and transformed after presenting the user a list of published services from a specified registry.

- Import of SLA document from tool: The source of such SLA documents is supposed to be network-transparent, which allows for access to both local files and files served over HTTP in distributed tooling environments.

- Import of SLA document from market: The user selects a registry endpoint and gets a list of public or owned SLA documents. Each imported document is treated as a modified version of the published one in order to control the resubmission.

- Export of SLA document to tool: This is useful if the SLA document should not be published on its own but rather as part of a tradeable service package. An editor would spawn the packaging tool with the document path as its argument.

- Export of SLA document to market: This export uploads the document to a service registry in an authenticated session. It needs to be associated to an existing service entry. The upload can optionally lead to the rewriting of existing service descriptions, for instance to update location pointers to SLA templates.

4 SLA ENGINEERING IN PRACTICE

We have validated the proposed SLA engineering approach by extending the former WSAG Editor with domain constraints, document transformations and tool and market integration points, and the subsequent usage and user testing of the editor on a cloud storage scenario service as an example of an IaaS offering.

The SLA editor which incorporates the domain-constrained and transformation-based SLA engineering approach has been implemented as a standalone application based on the Eclipse Rich Client Platform (RCP) Java library collection with supporting libraries for the service artefact types, for instance WSMO4J and WSAG4J. The editor uses parametrised XSL transformations to convert WSML artefacts into SLA documents. During this process, WSMO4J serialises the WSML artefact into its XML representation because typically WSML is modelled in human-readable compact syntax. Each WSML interface leads to the generation of a service level. As a secondary option, SLA documents can be generated from WSDL files. In this case, the service level information will not be present.

The editor currently contains visual support for the domain-independent modelling of services, participants, value types (for WSAG+VT objectives) and artefact references. Further visual support is given for the domain-specific modelling of cloud resource service properties. All widgets are dynamically loadable and could hence be loaded on demand from a dedicated engineering registry which is bound to the SDTs on a service marketplace.

The initiation of an SLA document through a graphical wizard is highlighted in Figure 6. Within the wizard, a source service description can be selected and a transformation can be applied to it. Once a preliminary SLA document has been created, the SLOs can be freely manipulated. Figure 7 shows a screenshot of the crucial editor component which is responsible for the expression of objectives. The editor integrates well into engineering environments through command-line parameters and document passing as files. It also retrieves from and publishes into the ConQo service registry. The editor menu with access to the integration options is contained in Figure 8.

For the user testing, we have developed a custom cloud storage service called EasyStorage with a WSDL and a WSML description. Both the WSAG Editor and the EasyStorage service have been installed into SPACEflight, a completely integrated research and education environment with many preconfigured platforms, tools and scenarios for experiencing advances in service sciences and cloud computing (OSSPRI, 2012). SPACEflight is accessible as a native installation on a desktop or notebook computer, as a virtual machine running in full-screen mode, or as a remote desktop instance running in the cloud. This way, the participants could concentrate on the editor usage instead of being side-tracked by installation or configuration procedures. We have then defined a number of modelling tasks which together with a set of evaluation questions were included in a survey for user testing. The test methodology includes a comparison with a previous testing round in which the former WSAG Editor (version $v1$) without any of the characteristics introduced in this work had

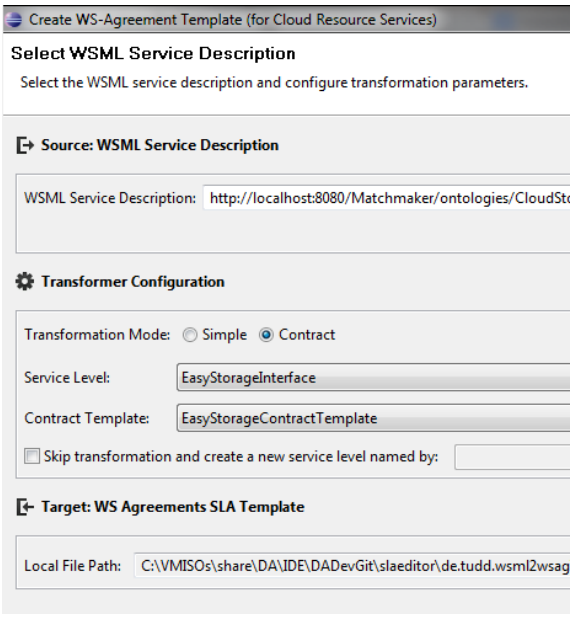

Figure 6: Wizard to generate new SLA documents

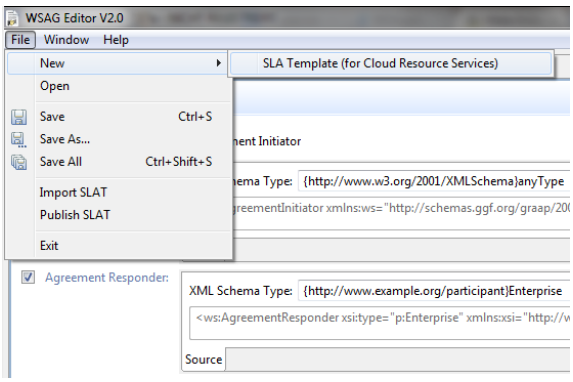

Figure 8: Menu of the editor to invoke the wizard to load and save SLATs and to create new SLATs or transform existing service descriptions into SLATs

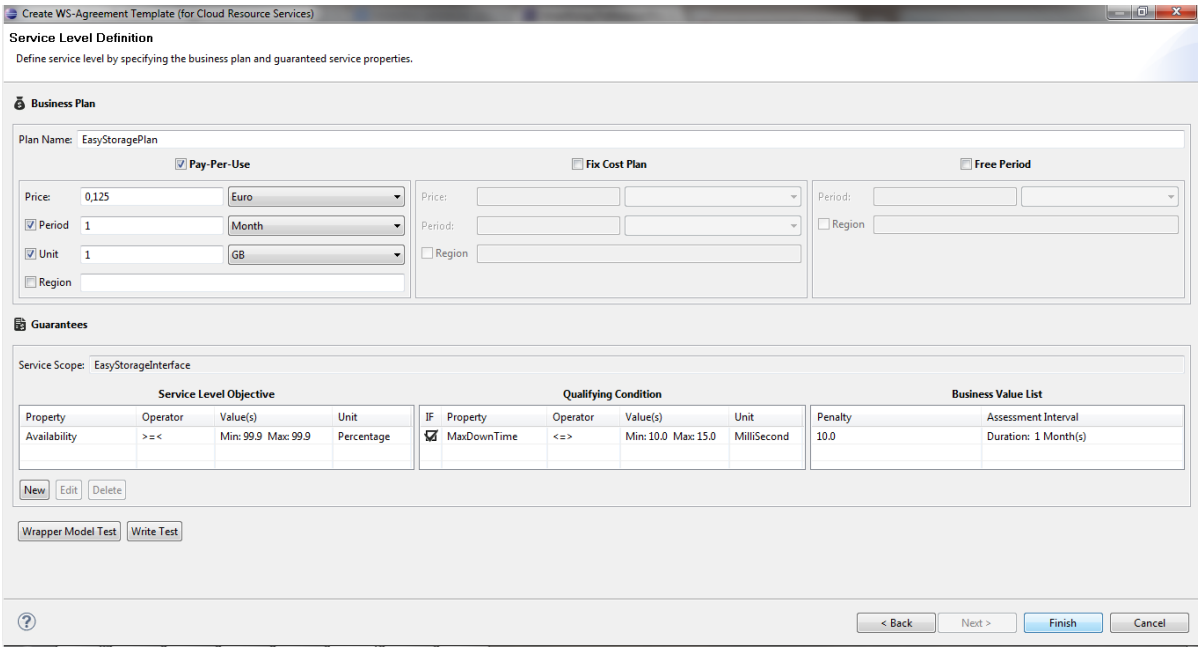

Figure 7: Editor component for business and technical SLOs

been evaluated. The participants were assisted with the new editor's (version *v*2) integrated help system and an SLA expert.

The first modelling task required the modelling of four SLOs, three of them with QCs. The second task required service innovation by taking imaginary feedback into account to adapt the SLA templates to more appropriate ones. The third task added the modelling of an existing commercial cloud storage service. Over these three tasks, seven subtasks were defined and measured in terms of how much time was spent on them. In addition, 16 general usability questions were asked. Table 3 outlines the central questionnaire performance metrics, including both the time spent on each task (T) and number of help inquiries with the expert present during the evaluation (H). Most tasks were not available with the editor version *v*1.

Table 3: Performance metrics in the evaluation questionnaire

Task	Tv1	Hv1	Tv1	Hv2
Create/Skip transform	-	-		
Create/Simple transform	-	-		
Create/Contract transform	-	-		
Upgrade SLA level	-	-		
Add level, w/ WSML	-	-		
Add level, w/o WSML	-	-		
Custom SLAT				

The evaluation was conducted with six participants. On average, the participants needed 28.17 minutes with 2.67 help inquiries for the first three tasks, 15.33 minutes with 1.5 inquiries for the second group of three, and 10.00 with 1.33 inquiries for the last task. The highest complexity has been measured with the modelling of compensation rules in case of SLA violations, which hints at a potential for improved presentation and interaction regarding the compensation strategies. The results are visualised in Figure 9.

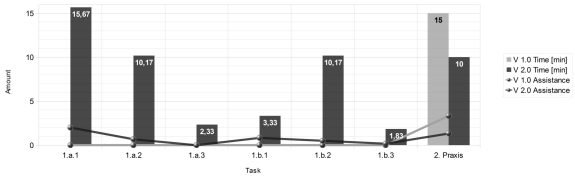

Figure 9: Performance comparison of the editing process between versions *v*1 and *v*2 of the editor

In addition to the performance, the efficiency of filling out all SLAT fields was measured. Without the transformation support, the users needed on average 15 minutes at the first time and 10.17 minutes for consecutive modelling sessions. The simple transformation of the SLAT from the service description with partially pre-filled fields required just 10.17 minutes for the first run and 7.33 for the consecutive ones. Finally, the full transformation based on the WSMO4IoS `ContractBase` ontology required 2.33 minutes for the first and 1.83 for consecutive sessions. This means that a speed-up factor of 1.38 can be achieved for simple transformations without any additional modelling effort, and a factor of 5.55 when the service descriptions are already enriched with ap-

propriate contract statements. The factors are outstanding in Figure 10.

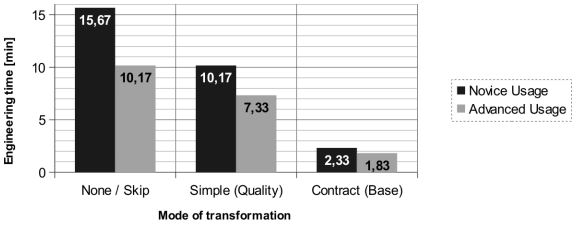

Figure 10: Efficiency comparison between the transformation-less and transformation-supported editing processes

Due to the nature of performing a user testing evaluation, all results are subjective and provide little empirical evidence of the improvements from using our approach. The initial feedback from the user testing is however highly positive. Participants are able to better understand the purpose of the SLA contents and can better express their guarantees compared to the earlier version of the editor. Hence, we assume that the approach is a valuable contribution to the field of service engineering and service management.

5 CONCLUSION

The presented SLA engineering approach brings significant improvements for service engineers and providers in service and cloud computing ecosystems. It helps providers to more rapidly achieve the desired expression of which service levels can be guaranteed. The three main distinct characteristics over previous modelling approaches are (1) domain-constrained service level objective modelling to acknowledge the differences in domain knowledge, (2) consistency-preserving transformation from service descriptions to SLA documents to avoid data curation problems, and (3) publishing and re-engineering integration with service marketplaces and other engineering tools. The WSAG editor, which has been extended by the authors of this work to evaluate the engineering approach, has been made freely available for download [1] under the auspices of the Open Source Service Platform Research Initiative to spur its adoption and ensure its long-term availability.

For the future, we see further potential improvements to the approach and the tool support for it. First, it shall be possible to model multiple SLA levels in one document with flexible combinations of WS-Agreement `All` and `ExactlyOne` block semantics. Second, constraints should be bundling so that multiple non-functional properties can be covered by a single constraint, as SLO or QC assertions. Third, the modelling of linear and more complex penalty functions should be supported. Finally, the widgets for visually supported domain-specific modelling should be dynamically retrieved from an engineering marketplace.

[1] WSAG editor website: `http://serviceplatform.org/wiki/Service_Engineering`

ACKNOWLEDGEMENTS

This work has received funding under project number 080949277 by means of the European Regional Development Fund (ERDF), the European Social Fund (ESF) and the German Free State of Saxony.

REFERENCES

Alhamad, M., Dillon, T., and Chang, E. (2010). Conceptual SLA framework for Cloud Computing. In *4th IEEE International Conference on Digital Ecosystems and Technologies (DEST)*, pages 606–610. Dubai, United Arab Emirates.

Andrieux, A., Czajkowski, K., Dan, A., Keahey, K., Ludwig, H., Nakata, T., Pruyne, J., Rofrano, J., Tuecke, S., and Xu, M. (2007). Web Services Agreement Specification (WS-Agreement). Grid Resource Allocation Agreement Protocol (GRAAP) WG of the Open Grid Forum, available online: `www.ogf.org/documents/GFD.107.pdf`.

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., and und Matei Zaharia, I. S. (2010). A view of cloud computing. *Communications of the ACM*, 53(4):50–58.

Baresi, L., Nitto, E. D., and Ghezzi, C. (2006). Toward Open-World Software: Issues and Challenges. *IEEE Computer*, 39(10):36–43.

Bertolino, A., Bianculli, D., Angelis, G. D., Frantzen, L., Kiss, Z. G., Ghezzi, C., Polini, A., Raimondi, F., Sabetta, A., Carughi, G. T., and Wolf, A. (2007). PLASTIC Test Framework: Prototype Implementation. PLASTIC Deliverable D4.2.

Boissel-Dallier, N., Lorré, J.-P., and Benaben, F. (2009). Management Tool for Semantic Annotations in WSDL. In *Proceedings of the Confederated International Workshops and Posters on On the Move to Meaningful Internet Systems: Beyond SAWSDL (OTM)*, pages 898–906. Vilamoura, Portugal.

Butler, J., Lambea, J., Nolan, M., Theilmann, W., Torelli, F., Yahyapour, R., Chiasera, A., and Pistore, M. (2011). SLAs Empowering Services in the Future Internet. In *The Future Internet – Future Internet Assembly 2011: Achievements and Technological Promises*, number 6656 in Lecture Notes in Computer Science (LNCS), pages 327–338. Budapest, Hungary.

D'Andria, F., Martrat, J., Kirkham, T., Naqvi, S., Gallop, J., and Arenas, A. (2007). The evolving use of Service Level Agreements and the influence of Trust within the support and development of Grids to enable a next generation of business models. In *Proceedings of Service Oriented Computing: A Look at the Inside (SOC@Inside'07) at ICSOC*.

Frankova, G., Malfatti, D., and Aiello, M. (2006). Semantics and Extensions of WS-Agreement. *Journal of Software*, 1(1):23–31.

Gillam, L., Li, B., and O'Loughlin, J. (2012). Adding Cloud Performance to Service Level Agreements. In *Proceedings of the 2nd International Conference on Cloud Computing and Services Science (CLOSER)*, pages 621–630. Porto, Portugal.

Gjermundrød, H., Dikaiakos, M. D., Stümpert, M., Wolniewicz, P., and Kornmayer, H. (2008). g-Eclipse - an integrated framework to access and maintain Grid resources. In *Proceedings of the 9th IEEE/ACM International Conference on Grid Computing (Grid)*, pages 57–64. Tsukuba, Japan.

Jouve, W., Lancia, J., Consel, C., and Pu, C. (2006). A Multimedia-Specific Approach to WS-Agreement. In *Proceedings of the European Conference on Web Services (ECOWS)*, pages 44–52. Zürich, Switzerland.

Kearney, K. T., Torelli, F., and Kotsokalis, C. (2010). SLA$\star$: An Abstract Syntax for Service Level Agreements. In *Proceedings of the 11th IEEE/ACM International Conference on Grid Computing (GRID)*, pages 217–224. Brussels, Belgium.

Keller, A. and Ludwig, H. (2002). The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services. Technical Report RC22456, IBM Research.

Kotsokalis, C., Rueda, J. L., Gomez, S. G., and Chimeno, A. E. (2011). Penalty Management in the SLA@SOI Project. In Wieder, P., Butler, J. M., and und Ramin Yahyapour, W. T., editors, *Service Level Agreements for Cloud Computing*, pages 105–121. Springer New York.

Lamanna, D., Skeene, J., and Emmerich, W. (2003). Specification Language for Service Level Agreements (SLAng). TAPAS Deliverable D2.

Mendes, C. and da Silva, M. M. (2012). DEMO-Based Service Level Agreements. In *Third International Conference on Exploring Service Sciences (IESS)*, volume 103 of *Lecture Notes in Business Information Processing (LNBIP)*, pages 227–242. Geneva, Switzerland.

Mitchell, B. and McKee, P. (2006). SLAs: A Key Commercial Tool. In Cunningham, P. and Cunningham, M., editors, *Innovation and the Knowledge Economy: Issues, Applications, Case Studies*. IOS Press, Amsterdam, Netherlands.

Müller, C., Durán, A., Resinas, M., Ruiz–Cortés, A., and Martín–Díaz, O. (2010). Experiences from building a WS–Agreement document analyzer tool (Including use cases in WS–Agreement and WSAG4People) v1.0. Technical Report ISA-10-TR-03, University of Seville.

OSSPRI (2012). SPACEflight - Live Demonstrator for Internet of Services and Cloud Computing Technologies. Software appliance, available online: `http://serviceplatform.org`.

Paschke, A. (2005). RBSLA - A declarative Rule-based Service Level Agreement Language based on RuleML. In *International Conference on Intelligent Agents, Web Technology and Internet Commerce (IAWTIC)*, volume 2, pages 308–314. Vienna, Austria.

Rady, M. (2012). Parameters for Service Level Agreements Generation in Cloud Computing – A Client-Centric Vision. In *Advances in Conceptual Modelling: Third International Workshop on Conceptual Modelling of Services (CMS)*, volume 7518 of *Lecture Notes in Computer Science (LNCS)*, pages 13–22. Florence, Italy.

Scheithauer, G., Voigt, K., Winkler, M., Bicer, V., and Strunk, A. (2011). Integrated Service Engineering Workbench: Service Engineering for Digital Ecosystems. *International Journal of Internet and Enterprise Management*, 9(5):392–413.

Spillner, J. (2012). WSMO4IoS – Ontology concept collection for the Internet of Services. Specification, available online: `http://serviceplatform.org/spec/wsmo4ios/`.

Spillner, J., Kümpel, A., Uhlig, S., Braun, I., and Schill, A. (2011). An Integrated Provisioning Toolchain for the Internet of Services. In *Proceedings of the 10th IADIS International Conference WWW/Internet*, pages 566–570. Rio de Janeiro, Brazil.

Stantchev, V. and Schröpfer, C. (2009). Negotiating and Enforcing QoS and SLAs in Grid and Cloud Computing. In *Proceedings of the 4th International Conference on Advances in Grid and Pervasive Computing (GPC)*, number 5529 in Lecture Notes in Computer Science (LNCS), pages 25–35. Geneva, Switzerland.

Wu, L. and Buyya, R. (2010). Service Level Agreement (SLA) in Utility Computing Systems. Technical Report CLOUDS-TR-2010-5, Cloud Computing and Distributed Systems Laboratory, The University of Melbourne, Australia.