Engineering Virtualized Services *

Elvira Albert
Complutense University of Madrid, Spain
elvira@fdi.ucm.es

Frank de Boer
CWI Amsterdam, The Netherlands
f.s.de.boer@cwi.nl

Reiner Hähnle
TU Darmstadt, Germany
haehnle@cs.tu-darmstadt.de

Einar Broch Johnsen
University of Oslo, Norway
einarj@ifi.uio.no

Cosimo Laneve
University of Bologna, Italy
laneve@cs.unibo.it

ABSTRACT

To foster the industrial adoption of virtualized services, it is necessary to address two important problems: (1) the efficient analysis, dynamic composition and deployment of services with qualitative and quantitative service levels and (2) the dynamic control of resources such as storage and processing capacities according to the internal policies of the services. The position supported in this paper is to overcome these problems by leveraging *service-level agreements* into software models and resource management into early phases of service design.

1. INTRODUCTION

Cloud computing is an execution environment with elastic resource provisioning, several stakeholders, and a metered service at multiple granularities for a specified level of quality of service (QoS) [10]. A host of cloud computing presents a number of services to client applications, including infrastructure and platform functionalities and software services for virtualizing the deployment of resources. This virtualization provides an elastic amount of resources to application-level services, thus making it possible to, for example, allocate a changing processing capacity to a service depending on demand. We say that application-level services are *virtualized* if they can adapt to the elasticity of cloud computing.

For virtualized services, resource provisioning is regulated by a legal contract between the service owner and the provider of the virtualized environment, called a *service-level agreement* (SLA). However, these legal texts are by their very nature not integrated in the software artifacts. Current modeling and analysis techniques make it extremely difficult for the software developer to realistically predict the resource requirements of the targeted service at an early design stage. Languages and tools for software development lack high-level support to systematically analyze performance under varying resource assumptions and to express and compare different resource management policies. Variations in end-user scenarios, value-added services, and dynamic service composition further complicate the picture by extending the functionalities of an application-level service at the expense of potentially changing its cost profile.

In traditional engineering processes for services, both the deployment and the SLA regulating the deployment are add-ons to the software development process. The appropriate deployment and SLA compliance are determined a posteriori, *after* the design of the service's program logic. Virtualization allows deployment and resource provisioning to be *internalized* as part of the program's logic, enabling services to dynamically scale to accommodate client traffic.

For software development methods to be effective in the engineering of virtualized services, it is our position that

1. SLAs should be part of a *design by contract methodology* for virtualized service engineering and
2. virtualized resources should be managed by explicit language primitives since the early phases of service design.

These are key concepts that (i) enable the composition of virtualized services with respect to their quality and (ii) allow software developers to address the challenges posed by virtualization for the software-as-a-service abstraction already at early stages of development. Services for virtualized environments require descriptions of resource-dependent and resource-aware behaviors that are based on abstract yet detailed executable models. This helps to optimize the usage of runtime resources, as well as to decrease development costs and shorten time to market for service developers.

Our position calls for a model-based analysis of quantitative (non-functional) aspects of SLAs, rather than qualitative aspects of SLAs such as security. A major implication of our position is to enable a coherent tool-based analysis of *models of SLA-aware application-level services* in the context of different *deployment scenarios*. This means that models should

1. capture scalable services through their support for resource awareness and resource management, and
2. be analyzed by applying techniques that are based on scalable methods.

*This position paper is written in the context of the EU project FP7-610582 ENVISAGE: Engineering Virtualized Services (`http://www.envisage-project.eu`).

Preprint. To appear in *Proc. 2nd Nordic Symposium on Cloud Computing & Internet Technologies (NordiCloud 2013)*. ACM DL, 2013.

In the sequel, we detail our position on model-based analysis of SLAs in a design by contract methodology, and discuss its consequences for a research agenda in formal methods.

2. DESIGN BY CONTRACT

The term *design by contract* was coined by Bertrand Meyer referring to the contractual obligations that arise when invoking methods in the object-oriented programming language Eiffel [23]: only if a caller can ensure that certain behavioral conditions hold before the method is activated (the precondition), it is assured that the method results in a specified state when it completes (the postcondition). Design by contract enables software to be organized as encapsulated services with interfaces specifying the contract between the service and its clients. Clients can "program to interfaces"; they need not know how the service is implemented.

Our position necessitates a design by contract methodology for SLA-aware virtualized services, which is to be integrated in industrial software development processes. By investigating the contractual obligations that are present in a PaaS environment, specific abstractions can be identified that are suitable for collaboration between platform provider, service provider, and service client. We build on work on interface automata [9], typestates [29], user-defined (i.e., application-level) scheduling policies [3,12], and process contracts [22] to define *service contracts*, a novel model of behavioral interfaces that specify the QoS and the resource usage in different deployment scenarios. The service contracts will be embedded in a modeling language for services and will be amenable to formal analysis.

Our position also implies the need to provide a range of tools for the analysis of models, based on formal techniques, that ensure conformance to application-level services. These tools enable the application of various analyses on the executable models already during the early design phase of the targeted service. This allows the developer to improve resource usage and QoS in deployment scenarios spanning from virtualized services for mobile users to resource provisioning in data centers. Regarding the analysis of the models, the design by contract approach ensures scalability of the analyses by compositionality: the encapsulated modules need only be checked with respect to their service contracts. To support a coherent and consistent suite of tools, both the modeling language and the service contracts will have a formal semantics. We finally remark that the description of different quantitative aspects in the service contracts drives both the *horizontal verification* of developed services, i.e. between service contracts themselves, and the *vertical verification*, i.e. between service contracts and the actual cost that can be reliably and automatically estimated for the models.

3. MODELING VIRTUALIZED SERVICES

General-purpose modeling languages exploit *abstraction* to reduce complexity [20]: descriptions primarily focus on the functional behavior and the logical composition of software. Industry-strength object-oriented programming and modeling languages, however, support different concurrency and interaction paradigms. The most prominent are multithreading and concurrent objects, using interaction mechanisms such as method calls, message passing, and shared resources. Researchers, including the authors of this paper, have developed a number of techniques to enable the compositional development of modular systems and the flexible reuse of components. However these techniques still overlook how a software's deployment influences its behavior. This is highly problematic for modern software targeting, for example, cloud computing and reflective middleware, where virtualization technologies allow an application to *modify resources of its deployment scenario during execution* [4].

To fully exploit the potential of virtualization, it is important to make services both scalable and cost efficient by leveraging deployment decisions to the software design. A major challenge in software engineering today is to find a tradeoff between the two conflicting requirements of abstraction and deployment control in the application design phase. In fact, the introduction of low-level deployment in a high-level modeling language is potentially disruptive in software engineering, but it is unavoidable due to the new scenario that is delineated by cloud computing. It is worth noting that in software design, no general, systematic means exists today to model and analyze software in the context of a set of available virtualized resources, nor to analyze redistribution of virtualized resources in terms of load balancing or reflective operations. To the best of our knowledge, no current research directly addresses these challenges raised by virtualization, and in particular, the modeling of quantitative virtualized resources as data inside the software itself, which is a primary property of virtualized resources.

Our starting point is a separation of concerns between the application model, which requires resources, and the deployment scenario, which reflects the virtualized computing environment and elastically provides resources. This allows the developer to analyze the performance and scalability of a service for many different deployment scenarios already at the modeling level. For example, the model of an application may be analyzed with respect to deployments on virtual machines that may vary in a number of features: the amount of allocated computing or memory resources, the choice of application-level scheduling policies for client requests, and the distribution of computation over different virtual machines with fixed bandwidth constraints. Automated resource analysis [1] can be used to determine the most appropriate choice of SLA for the application, and to validate that the abstract system model complies with the SLA.

Models of virtualized systems in this context need to be SLA-aware: the modeling language will include primitives to express resource modeling and to support the virtualization of resources at an appropriate abstraction level. This way, the modeling language can express cloud computing software, such as SaaS business applications or PaaS abstractions, and feature an interface through which the application-level services can inspect and manipulate the virtualized resources of the platform. We see this interface in relation with standardization efforts in virtualization and cloud provisioning. The abstraction level of the modeling language also allows virtualized systems to be mapped to different deployment scenarios which describe the underlying virtualized architecture and to express dynamic load balancing policies depending on both the SLA and the current deployment of the service.

Executable models that describe precisely the control and data flow of the target service are a necessity for the analysis of the resource needs in different settings. Such executable models also allow code generation from the modeling language to different mainstream implementation languages, such as JAVA, SCALA, or ERLANG. Concrete starting points for such models are *abstract behavioral specification* languages, such as ABS [13]. The ABS language targets distributed systems based on object-oriented concepts, thus it may be easily used by software engineers, and service-level contracts can be naturally integrated into the object-oriented interfaces. These *service contracts* follow and extend the design by contract methodology, and include both behavioral interfaces and QoS descriptions.

As a proof of concept several models that include deployment scenarios with parametric resources have been created in ABS [14, 15]. In these models application-level exchange of virtualized CPU resources is used to model and compare load balancing strategies between servers. Recently ABS has been applied to model dynamic resource management on the cloud [6, 16]. These case studies show that our proposed formal approach compares favorably to custom simulation tools and that it scales to industrial problems. However, the proof of concept does not yet permit ABS models to be parametrized with resource policies in terms of SLAs, nor does it extend formal analysis to varying resource models and dynamic deployment.

4. FORMAL LANGUAGES FOR SLAS

The formalization of SLAs is a prerequisite for developing formal analysis methods that check whether a service conforms to an SLA. For this reason, a number of formal SLA specifications have been developed [2, 18, 30]. They all define SLAs in terms of XML schemata. The problem of all these notations is their lack of a formal semantics: they are all mark-up languages that rely on an implicit (hence inherently ambiguous) understanding of the various concepts represented. Another problem is that, for virtualized systems, SLAs will require continuous re-assessment, for the duration of the SLA, to cope with changing enterprise conditions. It is not clear how this continuous reassessment is addressed in the above proposals.

Specification languages for SLA are currently being integrated with semantic annotations, e.g., SAWSDL [19] for service descriptions and SWAPS [25] for WS-Agreement. Another relevant example is SLAng [21], an object-oriented language with a precise formal interpretation in terms of service infrastructure and behaviors. Similarly, Okika formalizes BPEL in a rewriting logic framework [24]. These efforts, however, have limited expressive power and the extension to elastic resources of virtualized systems has not been investigated. A different, more abstract SLA formalism is CC-pi [8], a combination of concurrent constraint programming and pi-calculus formalisms, which models computational processes for specifying and negotiating QoS requirements, and supports reasoning about resource allocation. CC-pi only checks for consistency and does not address issues as optimization of business values or contractual norms—topics that are addressed in detail in [27] and in RBSLA [26].

Today, client-level SLAs do not allow the service's potential resource usage to be determined or adapted when unforeseen changes to resources occur. This is because user-level SLAs are not explicitly related to actual performance metrics and configuration parameters of the services. As a result the recent EU FP7 project SLA@SOI [28], which is being continued in the Future Internet PPP project FI-WARE [11], proposes an informal stepwise mapping between higher-level SLAs, such as those specified by clients, and lower-level SLAs and capabilities. This stepwise mapping is one of the prerequisites to support automated inference of resource usage from user-level SLAs.

Our position implies to push this line of research further and provide a modeling approach which *incorporates SLA requirements at the application-level* to ensure the QoS expectations of clients. This modeling approach will build on and consolidate the existing work to develop a practical, integrated SLA formalism for virtualized systems. It can be realized by investigating the contractual obligations that are present in, for example, a PaaS environment and provide specific abstractions that are suitable for the collaboration between platform provider, service provider, and client. The outcome of this consolidation effort includes a concrete SLA modeling language with a formal semantics and the formal description of (at least basic) enterprise level processes for SLA design and update. Another outcome is a contracts language that is embedded in an abstract behavioral modeling language such as ABS and thus amenable to formal analysis.

5. TOOLS FOR VIRTUALIZED SYSTEMS

Based on the formal semantics of an executable modeling language with integrated SLA, we envisage the development of a range of techniques for model-based analysis.

Monitoring and Service Contracts

Our position calls for techniques that address the difficulties of traditional monitoring tools [17], including fragmented visibility into the application stack, the lack of user-focused SLAs, and the absence of a budget perspective. The corresponding framework will provide a user-focused model with both a budget and cost perspective. Monitoring models must fill the gap between the negotiations with the client about SLAs, the service contract, and the deployment model.

Code Generation

Code generation for models of virtualized software should be instantiated to a deployment model. This will require developing new techniques, since models are rather high-level. However, automatic code generation will still be feasible, because models are executable and because of the constraints imposed by the deployment model. To prove correctness of the generated code, our position foresees the need for novel symbolic execution mechanisms that enable automated verification. Additionally, information about (asymptotic) resource consumption, computed by resource analysis of the high-level models, can be embedded into the profiles of the generated code. The correctness of such embedded information can be checked against the generated code to prove its validity, i.e., that the generated code preserves the resource consumption inferred from the model.

Resource Analysis

The cost analysis framework of virtualized services should be powerful enough to derive the deployment configuration and the interactions among services, and to automatically infer the overall cost from the cost of each service. The fact that this analysis will be developed at the level of the abstract models, which combine resource modeling with deployment modeling, allows these analyses to go beyond traditional cost models. For example, cost models for data size with bandwidth restrictions on communication can be developed for the underlying deployment scenarios. An important ambition (that stems from our overall position and that goes beyond the current existing technology) is to develop a resource analysis framework for determining whether certain resource usages are possible, given the service contracts of the component services.

Verification

We envisage an automatic deductive verification tool to ensure that a distributed, concurrent model respects a service contract. The properties stated in SLAs go beyond wellformedness of call sequences. For example, they involve limits imposed on storage space. Consequently, a first-order program calculus for the abstract modeling language is required. To achieve full automation, appropriate abstractions for service contracts need to be identified, following techniques suggested in [7], together with specialized proof search strategies and decision procedures. In case of a failed verification attempt, the developer should be supported by feedback on the kind of property that has been violated and under which condition. The developer also needs a concise rule book for modeling practices that help automation. In cases where automatic verification is still impossible, hybrid techniques can be considered, as discussed below.

Test Case Generation

Symbolic execution is the central part of most glassbox test case generation tools, which typically obtain the test cases from the branches of the symbolic execution tree. For virtualized services, the symbolic execution mechanisms should be integrated within a test case generation tool to produce test cases for the high-level models in a fully automatic way. The main challenge will be on handling distribution aspects of the services and the variety of deployment configurations within symbolic execution. Since information on resource management is explicitly available in our models it is possible to generate test cases that are aware of resource usage. A main research problem to be solved is *guidance* of the test case generator towards specific behaviors of the model by means of appropriate heuristics.

6. CONCLUSIONS

This position paper advocates a software engineering approach to virtualized services where (1) SLAs are part of a *design by contract methodology* and (2) virtualized resources are managed by explicit language primitives since the early phases of service design. This involves the extension of descriptions of virtualized services to encompass resource-dependent and resource-aware behaviors based on abstract yet detailed executable models. This new software engineering approach will render application-specific resource management policies to become fully integrated with the program logic of a service and analyzable already at an early stage in the development of the service. This in turn leads to a better exploitation of runtime resources, as well as to lower development costs and shorter time to market for service developers.

The scale of the potential economical benefits inherent to our proposal can be illustrated by the well-known cost increase to fix defects in later development phases [5]. IBM Systems Sciences Institute estimates that a defect which costs one unit to fix in design, costs 15 units to fix in testing (system/acceptance) and 100 units or more to fix in production (see Figure 1, left), and this cost estimation does not even consider the *impact cost* due to, e.g., delayed time to market, lost revenue, lost customers, bad public relations, etc. Now, these ratios are for *static infrastructure*. Considering the high additional complexity of resource management for virtualized services, it is reasonable to expect even more significant differences; Figure 1 (right) conservatively suggests ratios for virtualized software in *dynamic infrastructures*. The modeling and analysis approach proposed in this paper aims at detecting deployment errors such as the impossibility to meet an SLA, already *in the design phase*. The associated savings potential clearly justifies any additional cost that might be incurred from formalisation of SLAs.

The research agenda proposed in this position paper forms the basis of a new EU FP7 project called ENVISAGE that includes (1) a behavioral specification language for describing resource-aware models and deployment choices; (2) a simulator with visualization facilities; and (3) tool support for automated resource analysis, validation of SLAs, code generation, and runtime monitoring of SLAs for deployed services. As argued above, such a methodology and associated tools will allow services to be delivered in a more effective, efficient, and reliable manner than today, accelerating the development cycle and lowering the operational costs for innovative networked services that make use of cloud computing.

7. REFERENCES

[1] E. Albert, P. Arenas, S. Genaim, G. Puebla, and D. Zanardini. Cost Analysis of Java Bytecode, In *16th Eur. Symp. on Programming (ESOP)*, *LNCS* 4421, pages 157–172. Springer, 2007.

[2] D. Battré, F. M. T. Brazier, K. P. Clark, M. A. Oey, A. Papaspyrou, O. Wäldrich, P. Wieder, and W. Ziegler. A proposal for WS-agreement negotiation. In *11th IEEE/ACM Intl. Conf. on Grid Computing*, pages 233–241. IEEE CS Press, 2010.

[3] J. Bjørk, F. S. de Boer, E. B. Johnsen, R. Schlatte, and S. L. Tapia Tarifa. User-defined schedulers for real-time concurrent objects. *Innovations in Systems and Software Engineering*, 9(1):29–43, 2013.

[4] G. S. Blair, G. Coulson, P. Robin, and M. Papathomas. An architecture for next generation middleware. In *IFIP Intl. Conf. on Distributed Systems Platforms and Open Distributed Processing (Middleware'98)*, pages 191–206. Springer, 1998.

[5] B. W. Boehm and P. N. Papaccio. Understanding and controlling software costs. *IEEE Transactions on Software Engineering*, 14(10):1462–1477, 1988.

[6] F. S. de Boer, R. Hähnle, E. B. Johnsen, R. Schlatte,

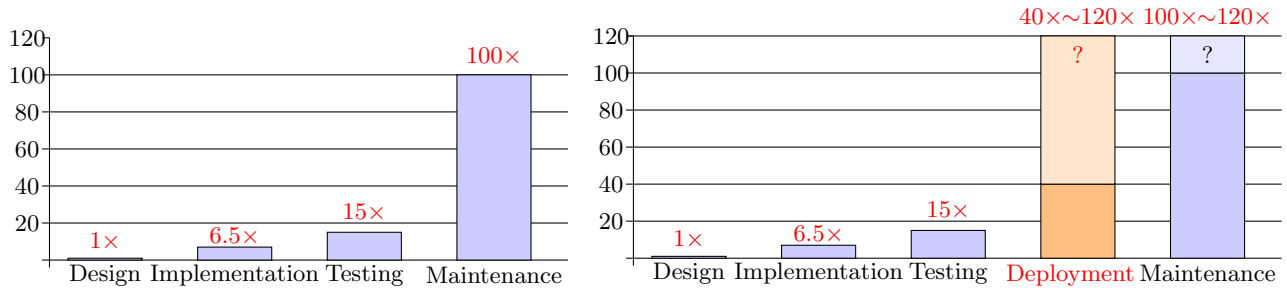

Figure 1: Relative costs to fix software defects for static infrastructure (*left*, source: IBM Systems Sciences Institute) and their extension to virtualized systems with dynamic infrastructure (*right*). The columns indicate the phase/stage of the software development at which the defect is found and fixed.

and P. Y. H. Wong. Formal modeling of resource management for cloud architectures: An industrial case study. In *Eur. Conf. on Service-Oriented and Cloud Computing (ESOCC)*, *LNCS* 7592, pages 91–106. Springer, 2012.

[7] R. Bubel, R. Hähnle, and B. Weiss. Abstract interpretation of symbolic execution with explicit state updates. In *6th Intl. Symp. on Formal Methods for Components and Objects (FMCO)*. Springer, 2009.

[8] M. G. Buscemi and U. Montanari. QoS negotiation in service composition. *J. Log. Algebr. Program.*, 80(1):13–24, 2011.

[9] L. De Alfaro and T. A. Henzinger. Interface automata. In *8th Eur. Software Engineering Conf. & 9th ACM SIGSOFT Intl. Symp. on Foundations of software engineering*, ESEC/FSE-9, pages 109–120. ACM Press, 2001.

[10] European Commission Expert Group Report. The future of cloud computing: Opportunities for European cloud computing beyond 2010, 2010.

[11] FI-WARE. Web: `www.fi-ppp.eu/projects/fi-ware`.

[12] M. M. Jaghoori, F. S. de Boer, T. Chothia, and M. Sirjani. Schedulability of asynchronous real-time concurrent objects. *Journal of Logic and Algebraic Programming*, 78(5):402–416, 2009.

[13] E. B. Johnsen, R. Hähnle, J. Schäfer, R. Schlatte, and M. Steffen. ABS: A core language for abstract behavioral specification. In *9th Intl. Symp. on Formal Methods for Components and Objects (FMCO)*, *LNCS* 6957, pages 142–164. Springer, 2011.

[14] E. B. Johnsen, O. Owe, R. Schlatte, and S. L. Tapia Tarifa. Dynamic resource reallocation between deployment components. In *Intl. Conf. on Formal Engineering Methods (ICFEM)*, *LNCS* 6447, pages 646–661. Springer, 2010.

[15] E. B. Johnsen, O. Owe, R. Schlatte, and S. L. Tapia Tarifa. Validating timed models of deployment components with parametric concurrency. In *Intl. Conf. on Formal Verification of Object-Oriented Software (FoVeOOS)*, *LNCS* 6528, pages 46–60. Springer, 2011.

[16] E. B. Johnsen, R. Schlatte, and S. L. Tapia Tarifa. Modeling resource-aware virtualized applications for the cloud in Real-Time ABS. In *Intl. Conf. on Formal Engineering Methods (ICFEM)*, *LNCS* 7635, pages 71–86. Springer, 2012.

[17] D. Jones. *The Definitive Guide to Monitoring the Data Center, Virtual Environments, and the Cloud.* Realtime publishers, 2010.

[18] A. Keller and H. Ludwig. The WSLA framework: Specifying and monitoring service level agreements for web services. *Journal of Network and Systems Management*, 11:57–81, 2003.

[19] J. Kopecký, T. Vitvar, C. Bournez, and J. Farrell. SAWSDL: Semantic annotations for WSDL and XML schema. *IEEE Internet Computing*, 11:60–67, 2007.

[20] J. Kramer. Is abstraction the key to computing? *Communications of the ACM*, 50(4):36–42, 2007.

[21] D. D. Lamanna, J. Skene, and W. Emmerich. SLAng: A language for defining service level agreements. *Future Trends of Distributed Computing Systems, IEEE Intl. Workshop*, page 100, 2003.

[22] C. Laneve and L. Padovani. The must preorder revisited. In *18th Intl. Conf. on Concurrency Theory*, *LNCS* 4703, pages 212–225. Springer, 2007.

[23] B. Meyer. Design by contract: The Eiffel method. In *TOOLS (26)*, page 446. IEEE CS Press, 1998.

[24] J. C. Okika. *Analysis and Verification of Service Contracts.* PhD thesis, Aalborg University, 2010.

[25] N. Oldham and K. Verma. Semantic WS-agreement partner selection. In *In 15th Intl. WWW Conf.*, pages 697–706. ACM Press, 2006.

[26] A. Paschke. RBSLA a declarative rule-based service level agreement language based on RuleML. In *Intl. Conf. on Computational Intelligence for Modelling, Control and Automation and Intl. Conf. on Intelligent Agents, Web Technologies and Internet Commerce*, pages 308–314. IEEE CS Press, 2005.

[27] J. P. Sauvé, F. Marques, A. Moura, M. C. Sampaio, J. Jornada, and E. Radziuk. SLA design from a business perspective. In *16th IFIP/IEEE Intl. Workshop on Distributed Systems: Operations and Management (DSOM)*, *LNCS* 3775. Springer, 2005.

[28] SLA@SOI. Web: `http://sla-at-soi.eu`.

[29] R. E. Strom and S. Yemini. Typestate: A programming language concept for enhancing software reliability. *IEEE Transactions on Software Engineering*, 12(1):157–171, 1986.

[30] V. Tosic, B. Pagurek, and K. Patel. WSOL - a language for the formal specification of classes of service for web services. In *Intl. Conf. on Web Services (ICWS)*, pages 375–381. CSREA Press, 2003.