Envisage: Developing SLA-aware Deployed Services with Formal Methods

Elvira Albert, Frank De Boer, Reiner Hähnle, Einar Broch Johnsen, Cosimo Laneve

Envisage: Developing SLA-aware Deployed Services with Formal Methods*

Elvira Albert[1], Frank de Boer[2], Reiner Hähnle[3], Einar Broch Johnsen[4], and Cosimo Laneve[5]

[1] Complutense University of Madrid, Spain, elvira@fdi.ucm.es
[2] CWI Amsterdam, The Netherlands, f.s.de.boer@cwi.nl
[3] TU Darmstadt, Germany, haehnle@cs.tu-darmstadt.de
[4] University of Oslo, Norway, einarj@ifi.uio.no
[5] University of Bologna, Italy – INRIA FOCUS, cosimo.laneve@unibo.it

Insufficient scalability and bad resource management of software services can easily eat up any potential savings from cloud deployment. Failed service-level agreements (SLAs) cause penalties for the provider, while oversized SLAs waste resources on the customer's side. 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 [6]; this cost estimation does not even consider the *impact cost* due to, for example, delayed time to market, lost revenue, lost customers, and bad public relations. The Envisage project aims at shifting deployment decisions from the end of the software engineering process to become an integral part of software design [2].

Deployment on the cloud gives software designers far reaching control over the resource parameters of the execution environment, such as the number and kind of processors, the amount of memory and storage capacity, and the bandwidth. In this context, designers can also control their software's trade-offs between the incurred cost and the delivered quality-of-service. SLA-aware services, which are *designed for scalability*, can even change these parameters dynamically, at runtime, to meet their service contracts. Envisage permits to design and validate these services by connecting executable models to formal service contracts and an API that is an abstraction of the cloud environment, see Fig. 1. This approach enables new kinds of analysis:

- **Simulation ("Early modeling"):** The formally defined modeling language ABS [10] realizes a separation of concerns between the *cost* of execution and the *capacity* of dynamically provisioned cloud resources [11]. Models are executable; a simulation tool supports rapid prototyping and visualization.
- **Formal methods ("Early analysis"):** as ABS was designed for analysis, it enables a range of tool-supported formal techniques, including behavioral types for deadlock analysis and SLA compliance [8], worst-case cost analysis [1], deductive verification [7], and automated test-case generation [4].
- **Monitoring ("Late analysis"):** ABS supports code generation backends [5] that preserve upper bounds on cost and permit performance monitoring of the provisioned cloud resources after deployment [13].

* Supported by the EU project FP7-610582 *Envisage: Engineering Virtualized Services* (http://www.envisage-project.eu).

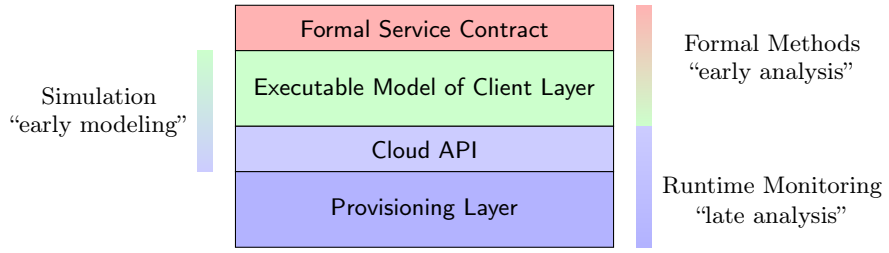

Fig. 1. Making services SLA-aware by means of formal methods, from [9].

The modeling approach and analyses developed in Envisage have been successfully applied in an industrial context to SDL Fredhopper's eCommerce Optimization [3] and Apache Hadoop YARN [12].

References

1. E. Albert, P. Arenas, A. Flores-Montoya, S. Genaim, M. Gómez-Zamalloa, E. Martin-Martin, G. Puebla, and G. Román-Díez. SACO: Static Analyzer for Concurrent Objects. In *TACAS*, LNCS 8413, pages 562–567. Springer, 2014.
2. E. Albert, F. de Boer, R. Hähnle, E. B. Johnsen, and C. Laneve. Engineering virtualized services. In *2nd Nordic Symposium on Cloud Computing & Internet Technologies (NordiCloud'13)*, pages 59–63. ACM Press, 2013.
3. E. Albert, F. S. de Boer, R. Hähnle, E. B. Johnsen, R. Schlatte, S. L. Tapia Tarifa, and P. Y. H. Wong. Formal modeling of resource management for cloud architectures: An industrial case study using Real-Time ABS. *Journal of Service-Oriented Computing and Applications*, 8(4):323–339, 2014.
4. E. Albert, M. Gómez-Zamalloa, and M. Isabel. SYCO: A systematic testing tool for concurrent objects. In *Compiler Construction (CC'16)*. ACM, 2016.
5. N. Bezirgiannis and F. de Boer. ABS: A high-level modeling language for cloud-aware programming. In *SOFSEM*, pages 433–444. Springer, 2016.
6. B. W. Boehm and P. N. Papaccio. Understanding and controlling software costs. *IEEE Trans. SW Eng.*, 14(10):1462–1477, 1988.
7. C. C. Din, R. Bubel, and R. Hähnle. KeY-ABS: A deductive verification tool for the concurrent modelling language ABS. In *CADE*, LNCS 9195. Springer, 2015.
8. E. Giachino, C. Laneve, and M. Lienhardt. A Framework for Deadlock Detection in ABS. *Software and Systems Modeling*, 2016. To Appear.
9. R. Hähnle and E. B. Johnsen. Designing resource-aware cloud applications. *IEEE Computer*, 48(6):72–75, 2015.
10. E. B. Johnsen, R. Hähnle, J. Schäfer, R. Schlatte, and M. Steffen. ABS: A core language for abstract behavioral specification. In *FMCO*, LNCS 6957, pages 142–164. Springer, 2011.
11. E. B. Johnsen, R. Schlatte, and S. L. Tapia Tarifa. Integrating deployment architectures and resource consumption in timed object-oriented models. *Journal of Logical and Algebraic Methods in Programming*, 84(1):67–91, 2015.
12. J.-C. Lin, I. C. Yu, E. B. Johnsen, and M.-C. Lee. ABS-YARN: A formal framework for modeling Hadoop YARN clusters. In *FASE*, LNCS 9633. Springer, 2016.
13. B. Nobakht, S. de Gouw, and F. S. de Boer. Formal verification of service level agreements through distributed monitoring. In *ESOCC*, LNCS 9306, pages 125–140. Springer, 2015.