Radboud Repository

Radboud University Nijmegen

PDF hosted at the Radboud Repository of the Radboud University Nijmegen

The following full text is a publisher's version.

For additional information about this publication click this link.
http://hdl.handle.net/2066/204533

Please be advised that this information was generated on 2021-10-20 and may be subject to change.

Article 25fa pilot End User Agreement

This publication is distributed under the terms of Article 25fa of the Dutch Copyright Act (Auteurswet) with explicit consent by the author. Dutch law entitles the maker of a short scientific work funded either wholly or partially by Dutch public funds to make that work publicly available for no consideration following a reasonable period of time after the work was first published, provided that clear reference is made to the source of the first publication of the work.

This publication is distributed under The Association of Universities in the Netherlands (VSNU) 'Article 25fa implementation' pilot project. In this pilot research outputs of researchers employed by Dutch Universities that comply with the legal requirements of Article 25fa of the Dutch Copyright Act are distributed online and free of cost or other barriers in institutional repositories. Research outputs are distributed six months after their first online publication in the original published version and with proper attribution to the source of the original publication.

You are permitted to download and use the publication for personal purposes. Please note that you are not allowed to share this article on other platforms, but can link to it. All rights remain with the author(s) and/or copyrights owner(s) of this work. Any use of the publication or parts of it other than authorised under this licence or copyright law is prohibited. Neither Radboud University nor the authors of this publication are liable for any damage resulting from your (re)use of this publication.

If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please contact the Library through email: copyright@ubn.ru.nl, or send a letter to:

University Library
Radboud University
Copyright Information Point
PO Box 9100
6500 HA Nijmegen

You will be contacted as soon as possible.

Improved Architectures/Deployments with Elmo

Arjan Lamers[1,2], Marko van Eekelen[2,3], and Sung-Shik Jongmans[2,4](✉)

[1] First8 B.V., Nijmegen, The Netherlands
[2] Department of Computer Science, Open University of the Netherlands, Heerlen, The Netherlands
ssj@ou.nl
[3] Institute for Computing and Information Sciences, Radboud University Nijmegen, Nijmegen, The Netherlands
[4] Department of Computing, Imperial College London, London, UK

Abstract. Manually reasoning about candidate refactorings to alleviate bottlenecks in service-oriented systems is hard, even when using high-level architecture/deployment models. Nevertheless, it is common practice in industry. Elmo is a decision support tool that helps service-oriented architects and deployment engineers to analyze and refactor architectural and deployment bottlenecks in service-oriented systems.

Keywords: Services · Refactoring · Architecture · Deployment

1 Highlights

Elmo is a decision support tool that helps service-oriented architects and deployment engineers to analyze and refactor architectural and deployment bottlenecks in service-oriented systems. We first summarize the highlights:

– *Elmo uses a light-weight, high-level,* ***whiteboard-style notation*** *to model architectures/deployments (Fig. 1).* This is important, as business executives often base their decisions on such "whiteboard models": the ability to explain consequences of architecture/deployment refactorings in a simple notation that is naturally understandable to executives is vital in industry.
– *Elmo* ***automatically analyzes potential bottlenecks*** *in architecture and deployment models.* The analysis performed by Elmo is based on the formal semantics of Elmo's whiteboard-style notation [8,15]. The user can subsequently manipulate the models (by selecting refactorings from a list) to improve the models and eliminate the bottleneck. In addition to this:
– *Elmo* ***automatically infers refactorings*** *of architectures/deployments, guaranteed to achieve a given goal.* Using graph exploration algorithms, Elmo is capable of exploring the space of possible refactorings (up to a given depth), and if multiple chains of refactorings achieve the same goal, Elmo automatically compares the other pros and cons of these chains. However:

© Springer Nature Switzerland AG 2019
X. Liu et al. (Eds.): ICSOC 2018 Workshops, LNCS 11434, pp. 419–424, 2019.
https://doi.org/10.1007/978-3-030-17642-6_36

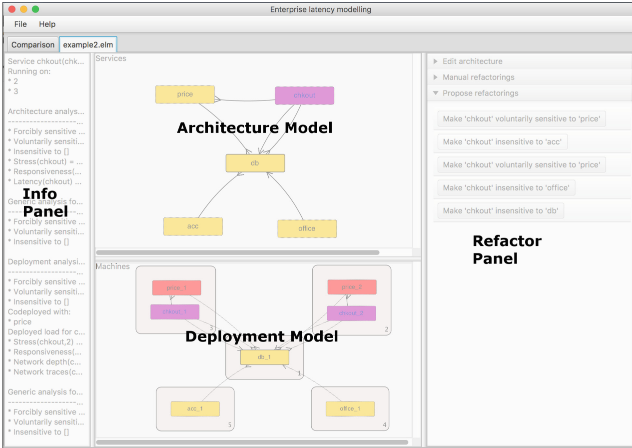

Fig. 1. Elmo (blurred: simplified e-commerce system)

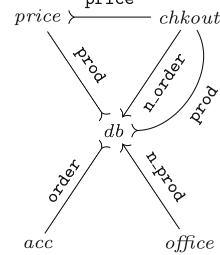

Fig. 2. Architecture model of a simplified e-commerce system (cf. Fig. 1)

– *Elmo* ***does not carry out*** *refactorings.* Elmo is geared toward providing decision support: its sole purpose is to help architects and deployment engineers (1) to find the best solution for a refactoring problem, and (2) perhaps more importantly, to convince executives that this solution should indeed be implemented. Actual implementation of refactorings is beyond Elmo's scope.
– *Elmo* ***works purely qualitatively*** *instead of quantitatively.* Once performance issues start to manifest, Elmo avoids a need for brittle and expensive additional performance measurements (profiling, monitoring, simulations, etc.), relying exclusively on dependency analyses between services.
– *Elmo is being developed and* ***evaluated with industry*** *partner First8.* This already led to promising initial results. First8 is a software company that specializes in custom business-critical systems; refactoring service-oriented systems to improve performance is an important activity of First8.

2 Motivation

In current industrial practice, architects and deployment engineers need to *manually* reason about candidate refactorings to alleviate bottlenecks in service-oriented systems. Even when using high-level architecture/deployment models, this has three major issues. *First*, reasoning about refactorings is an intellectually demanding activity, and often requires architects to make simplifying assumptions and/or apply additional abstractions. This leads to imprecise refactoring proposals, which may be more costly, more risky, and less effective than necessary. *Second*, as refactoring proposals are based on experience and best-practices, architects can easily overlook less-intuitive refactorings that may very well be

most-effective for a given system. *Third*, predicting how different refactorings will interact with each other is hard (e.g., an improvement in the architecture may be canceled out by an improvement in the deployment, due to resource sharing). In a large-scale project, First8 architects indeed struggled with these issues, which strongly motivated Elmo's development.

3 The Tool

Elmo is shown in Fig. 1, in the context of a simplified e-commerce system (see the demo for details [9]). Using the graphical interface, Elmo allows an architect to construct architecture/deployment models. As shown in Fig. 1, these models are lightweight, and they look similar to architectural whiteboard diagrams.

Formally [8,15], the models are graphs; Fig. 2 shows an example (i.e., an architecture model of the same simplified e-commerce system as in Fig. 1). Vertices represent services (architecture) or service instances (deployment), while edges represent service calls. We distinguish between *pushes* (denoted as $s_1 \longrightarrow s_2$, meaning that s_1 initiates an information flow from s_1 to s_2) and *pulls* (denoted as $s_1 \longrightarrow\!\!\prec s_2$, meaning that s_1 initiates an information flows from s_2 to s_1), as these impact performance differently. Calls are furthermore annotated with the type of information involved. One or more instances of services can be *deployed* on one or more *machines*.

Each time information is pushed to or pulled from a service s, work needs to be done and *stress* is imposed on s. The stress set of s ($\mathit{Stress}(s)$) is defined as the set of services that may impose stress directly or indirectly by being stressed themselves. Service instances sharing the same machine also share their stress. A related notion is *responsiveness* ($\mathit{Resp}(s)$), which indicates how quickly a service s is able to deliver its work. It consists both of the stress set of the service, as well as the stress of the services from which it needs to pull information. Based on stress and responsiveness, three qualitative sensitivity levels can be defined: service a is *insensitive* to service b if b never impacts a (formally: $b \notin \mathit{Resp}(a)$); a is *willingly sensitive* to b if b impacts a *only* on a's initiative (formally: $b \in \mathit{Resp}(a) \wedge b \notin \mathit{Stress}(a)$); a is *unwillingly sensitive* to b if b impacts a *also* on b's initiative (formally: $b \in \mathit{Stress}(a)$). For instance, if service s_1 pushes information to service s_2, the performance of s_2 is sensitive to s_1, *unwillingly* (i.e., s_2 has no control). In contrast, if s_1 pulls information from s_2, the performance of s_1 is sensitive to s_2, *willingly* (i.e., s_1 chooses to depend on s_2).

These models can be used to explore sensitivity characteristics of the system: an architect can select service (-instances) and Elmo evaluates and highlights its sensitivities. Possible refactorings are immediately shown with resulting effects; these refactorings are formally guaranteed to preserve the existing behavior [8]. An architect can then apply the suggested refactorings to the architecture/deployment model, to try out ideas and compare results of different strategies. Also, using memory-optimized graph exploration algorithms, Elmo can automatically suggest a chain of refactorings towards a given goal (e.g., "make service s_4 insensitive to service s_1").

Elmo is open-source [10] and implemented in JavaFX. The refactorings currently supported are: splitting and merging services, changing pushes to pulls and vice versa, changing deployments of services, and adding caches and/or queues.

4 Working with Elmo

The typical Elmo workflow looks as follows. First, architects and deployment engineers sit together to manually construct a high-level architecture/ deployment model of the system (initially on a whiteboard; subsequently in Elmo).[1]

A typical goal is improving performance of a service, or preparing for increased load on an endpoint provided by a service. Given such a goal, a natural starting point (the service-under-investigation) is often clear.

Depending on the aims of the architects and deployment engineers, they can subsequently select Elmo's "sensitivities perspective" (to improve a service's performance) or its "impact perspective" (to find potential scalability issues). In the former case, Elmo computes and shows to which services the service-under-investigation is sensitive; in the latter case, Elmo computes and shows which services will be impacted by an increased load on the service-under-investigation.

Based on Elmo's analyses, the architects and deployment engineers can subsequently pick services that, undesirably, are sensitive to the service-under-investigation (or vice versa), formulate a refactoring goal to alleviate these sensitivities, and let Elmo explore the search space. Finally, Elmo reports candidate refactorings, along with a number of metrics to compare them against each other. The architects and deployment engineers can evaluate these options, and apply the one of their choice to the model.

5 First Industrial Case Study

JoinData is a digital highway for farm-generated data, used nation-wide in the Netherlands. It allows for data exchange in the agricultural sector. For example, milking-robots on the farm, suppliers of animal feed, or laboratories that provide reports on soil can exchange information with accountancy firms, governmental organisations, or farm management systems. *EDI-Circle* is the messaging component of the JoinData platform and acts as a hub that receives, adapts and routes messages between parties. Due to expected growth, the scalability and performance of EDI-Circle need to improve. We compared (1) the manual analysis and proposed course of action by the lead architect of the EDI-Circle project with (2) Elmo's automated analysis.

The service-under-investigation was determined to be the *download* service s, the primary service provided by EDI-Circle. Elmo's analysis confirmed that s had

[1] Note that although this is currently a manual activity, we are planning to investigate integration of Elmo with existing tools to (semi-)automatically generate models of existing service-oriented systems (e.g., [11]).

many sensitivities. The proposed solution by the architect was "to change the whole system, since everything is connected", by trying to improve the throughput of *all* services involved, starting with the database. Elmo's analysis, however, showed that some services to which s was sensitive, logically should not affect s's performance. Elmo therefore proposed a much more localized chain of refactorings, *excluding the database*, specifically aimed to alleviate these unnecessary sensitivities. This solution was not proposed by the architect; to find it, Elmo explored a space of 1,143,227 possible scenarios. This is first evidence that Elmo can offer significant benefits.

6 Related Work and Novelty

– *Qualitative vs. quantitative.* Contrasting a large body of work on quantitative modeling (e.g., [2,4,12,14,21]), the models that First8 architects today use (and thus supported by Elmo) are *intentionally* qualitative. Although quantitative models can be more accurate, such models require load functions, detailed descriptions, or actual implementations; these are often demanding to obtain. Also, calculating the performance of the architecture/deployment might not be instantaneous, but may require a lengthy simulation. Finally, small changes in deployments or algorithms, can have a big impact on performance, rendering painfully obtained performance measurements obsolete. To our knowledge, no other tools exist that take a similar approach to ours to aid in reasoning about refactorings of service-oriented systems, their architectures, and their deployments. This places Elmo in a unique position.
– *Design refactorings vs. implementation refactorings.* There are also other architecture tools that aid in refactoring an existing architecture. They help in visualizing the architecture, detecting code smells like dependency cycles or validating architecture rules (e.g. [3,5,16,19,20]). These tools work at the implementation/code level and do not take actual deployment into account, nor can they evaluate performance sensitivities like Elmo does.
– *Monitoring.* Application Performance Monitoring (e.g. [1,6,7,17]) tools provide a quick insight in actual interaction between services and aid in detecting real problems. They can only do this when software is actually deployed, not during design. Whereas these tools can identify performance bottlenecks, they have only very limited support for finding solutions. Elmo, in contrast, can automatically compute series of refactorings to achieve a given goal.
– *Modeling techniques. UML Component Diagrams* enable architects to document dependencies between components/services. A key difference with our approach is that component diagrams do not distinguish between push and pull operations [13] (i.e., component diagrams model *dependencies* between components, but they do not model the *directions of information flows* that push and pull operations additionally convey); in our model, this is vital information to reason about sensitivities between services. To provide such information in UML, complementary behavioral diagrams (e.g., *UML Sequence Diagrams*) can be used, but then the level of detail becomes too low for our

purpose, while at the same time a maintenance burden emerges. Also, mixing different types of diagrams is cumbersome.
It may be interesting to investigate whether a new *UML Profile* can be used to *extend* component diagrams with Elmo-like features (although we note that the UML specification suggests to use profiles for component diagrams only to capture information about whether the interface of one component "is suitable for consumption by the depending component" [18]).

References

1. AppDynamics LLC: AppDynamics. https://www.appdynamics.com
2. Bertoli, M., Casale, G., Serazzi, G.: JMT: performance engineering tools for system modeling. SIGMETRICS Perform. Eval. Rev. **36**(4), 10–15 (2009)
3. Bischofberger, W., Kühl, J., Löffler, S.: Sotograph – a pragmatic approach to source code architecture conformance checking. In: Oquendo, F., Warboys, B.C., Morrison, R. (eds.) EWSA 2004. LNCS, vol. 3047, pp. 1–9. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24769-2_1
4. Brebner, P.: Real-world performance modelling of enterprise service oriented architectures: delivering business value with complexity and constraints (abstracts only). SIGMETRICS Perform. Eval. Rev. **39**(3), 12 (2011)
5. Caracciolo, A., Lungu, M.F., Nierstrasz, O.: A unified approach to architecture conformance checking. In: WICSA, pp. 41–50. IEEE Computer Society (2015)
6. Datadog: Datadog. https://www.datadoghq.com
7. DynaTrace LLC: DynaTrace. https://www.dynatrace.com
8. van Eekelen, M., Jongmans, S.S., Lamers, A.: Non-quantitative modeling of service-oriented architectures, refactorings, and performance. Tech. Rep. TR-OU-INF-2017-02, Open University of The Netherlands (2017)
9. Elmo Team: Elmo video. https://youtu.be/Oi9kxqh_GBs
10. Elmo Team: Elmo website. http://www.open.ou.nl/ssj/elmo
11. Granchelli, G., Cardarelli, M., Francesco, P.D., Malavolta, I., Iovino, L., Salle, A.D.: Microart: a software architecture recovery tool for maintaining microservice-based systems. In: ICSA Workshops, pp. 298–302. IEEE (2017)
12. Juan Ferrer, A., et al.: OPTIMIS: a holistic approach to cloud service provisioning. Future Gener. Comp. Syst. **28**(1), 66–77 (2012)
13. Kobryn, C.: Modeling components and frameworks with UML. Commun. ACM **43**(10), 31–38 (2000)
14. Kounev, S.: Performance modeling and evaluation of distributed component-based systems using queueing petri nets. IEEE Trans. Softw. Eng. **32**(7), 486–502 (2006)
15. Lamers, A., van Eekelen, M.: A lightweight method for analysing performance dependencies between services. In: Celesti, A., Leitner, P. (eds.) ESOCC Workshops 2015. CCIS, vol. 567, pp. 93–110. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-33313-7_7
16. Lattix LDM2: Lattix Architect 10. http://lattix.com
17. New Relic: New Relic Inc. https://newrelic.com
18. Object Management Group: Unified Modeling Language 2.5.1 (2017). https://www.omg.org/spec/UML/2.5.1/
19. SonarSource: SonarQube. https://www.sonarqube.org
20. Structure101: Structure101 Studio. https://structure101.com
21. Zhu, L., Liu, Y., Bui, N.B., Gorton, I.: Revel8or: model driven capacity planning tool suite. In: ICSE, pp. 797–800. IEEE Computer Society (2007)