RC25681 (WAT1810-052) October 23, 2018
Computer Science

IBM Research Report

Papers From the 12th Advanced Summer School on Service-Oriented Computing (SummerSOC'18)

Johanna Barzen[1], Rania Khalaf[2], Frank Leymann[1], Bernhard Mitschang[1]
Editors

[1]University of Stuttgart
Universitätsstraße 38
70569 Stuttgart, Germany

[2]IBM Research Division, Cambridge
75 Binney Street
Cambridge, MA 02142 USA

The 12th Advanced Summer School on Service-Oriented Computing

June 24 - June 29
2018
Hersonissos, Crete, Greece

The 12th Advanced Summer School on Service Oriented Computing (SummerSOC'18) continued a successful series of summer schools that started in 2007, regularly attracting world-class experts in Service Oriented Computing to present state-of-the-art research during a week-long program organized in several thematic tracks: patterns and IoT, Quantum Computing, formal methods for SOC, computing in the clouds, data science, e-Health and emerging topics. The advanced summer school is regularly attended by top researchers from academia and industry as well as by PhD and graduate students.

During the different sessions at SummerSOC renowned researchers gave invited tutorials on subjects from the themes mentioned above. The afternoon sessions were also dedicated to original research contributions in these areas: these contributions have been submitted in advance as papers that had been peer-reviewed. Accepted papers were presented during SummerSOC and during the poster session. Furthermore, PhD students had been invited based on prior submitted and reviewed extended abstracts to present the progress on their theses and to discuss it during poster sessions. Some of these posters have been invited to be extended as a full paper, which are included in this technical report.

Also, this year the "Christos Nikolaou Memorial Ph.D. Award" to honor Prof. Christos Nikolaou's career-long contributions in university education and research was awarded for the third time. The winner of the Christos Nikolaou Memorial Ph.D. Award is Jacopo Soldani, who presents the core ideas of his awarded thesis in this technical report too. The award is not only an honor and distinction but is associated with 2000€ for the awardee, sponsored by StartTech Ventures.

Johanna Barzen, Rania Khalaf, Frank Leymann, Bernhard Mitschang
- Editors -

Content

On modelling, analysing and reusing multi-component applications

Jacopo Soldani

Department of Computer Science, University of Pisa, Pisa, Italy
soldani@di.unipi.it

Abstract. How to deploy and flexibly manage complex composite applications across heterogeneous cloud platforms is one of the main concerns in enterprise IT. There is a need for vendor-agnostic models allowing to specify the structure and management of composite cloud applications, and of techniques for verifying their design. Furthermore, the availability of techniques for reusing cloud applications would free developers from the need of designing/developing multiple times recurring application components. This paper provides an overview of our research contributions on (i) modelling and analysing the structure and management of composite cloud applications, and on (ii) fostering their reuse.

1 Introduction

Cloud computing has revolutionised IT, by offering a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable resources [25]. Applications and resources have no more to be bought and managed on premise, but they can be simply requested (and paid) when the corresponding functionality is actually needed [18].

Reaping the benefits of cloud computing is however not easy. Cloud applications typically integrate various and heterogeneous components, whose deployment, configuration, enactment, and termination must be properly coordinated [21]. This must be done by carefully taking into account all the dependencies occurring among the application components. As the number of components grows, or the need to reconfigure them becomes more frequent, application management becomes more and more time-consuming and error-prone [2].

Furthermore, current cloud technologies suffer from a lack of standardisation, with similar resources offered in a different manner by different providers [28]. As a result, application operators wishing to deploy and manage the same application on different cloud platforms (by fulfilling its individual requirements) might be required to re-design and re-configure most of the middleware and infrastructure layers, even from scratch. This requires deep technical expertise, and it results in error-prone development processes which significantly increase the costs for operating and maintaining composite cloud applications [31].

> *How to deploy and flexibly manage complex composite applications over heterogeneous cloud platforms is hence a serious challenge in today's enterprise IT [24].*

As highlighted by Binz et al. [2] and by Di Cosmo et al. [20], the above challenge results in two major issues, namely (i_1) how to automate the deployment and management of composite cloud applications and (i_2) how to support their vendor-agnostic design. Focusing on such issues, this paper provides an overview of our contributions trying to advance the state of the art on what follows:

– *Modelling and analysing composite cloud applications.* By relying on topology graphs [3] for describing the structure of composite cloud applications, we have proposed a compositional modelling that permits specifying the management behaviour of application components, by also taking into account that faults eventually occur while managing complex applications [19]. We have also proposed techniques for checking and planning the management of a composite cloud application.
– *Reusing composite cloud applications.* We have defined techniques for matching and adapting existing applications. Developers can describe the structure and desired behaviour of the components they need (e.g., a web server, or a DBMS), and our techniques can then be exploited to concretise their implementation.

The rest of this paper is organised as follows. Sect. 2 illustrates our contributions on modelling and analysing composite cloud applications, while Sect. 3 recaps our contributions on fostering their reuse. Finally, Sect. 4 draws some concluding remarks and illustrates some directions for future work.

2 Modelling and analysing cloud applications

A convenient way to represent complex composite applications (such as those deployed in cloud platforms) is a topology graph [3], whose nodes represent the application components, and whose arcs represent the dependencies among such components. More precisely, each topology node can be associated with the requirements of a component, the operations to manage it, and the capabilities it features. Inter-node dependencies associate the requirements of a node with capabilities featured by other nodes.

The *Topology and Orchestration Specification for Cloud Applications* (TOSCA [26]) meets this intuition, by providing a standardised modelling language for representing the topology of a cloud application. It also permits coordinating the application management, by defining (workflow) plans orchestrating the management operations offered by each component[1].

Unfortunately, in its current version, TOSCA does not permit specifying the behaviour of a cloud application's management operations. More precisely, it is not possible to describe the order in which the operations of a component must be invoked, nor how those operations depend on the requirements or how they affect the capabilities of that component (and hence the requirements of other components they are connected to). This implies that the verification of

[1] A self-contained introduction to TOSCA can be found in [16].

whether a management plan is valid can only be performed manually, with a time-consuming and error-prone process[2].

In [6, 11], we propose an extension of TOSCA that permits specifying the behaviour of the management operations of the components forming an application. We indeed show how their management behaviour can be modelled by *management protocols*, specified as finite state machines whose states and transitions are associated with conditions defining the consistency of the states of a component and constraining the executability of its management operations. Such conditions are defined on the requirements of a component, and each requirement of a component has to be fulfilled by a capability of another component.

We also illustrate how to derive the management behaviour of a cloud application by composing the protocols of its components, and how this permits automating various analyses concerning the management of a cloud application, like determining whether management are valid, which are their effects, or which plans permit reaching certain application configurations.

To deal with the potential occurrence of faults (which have to be considered when managing complex cloud applications [19]), we have then further extended management protocols. In [7, 9], we propose *fault-aware management protocols*, which permit modelling how nodes behave when faults occur, and we illustrate how to analyse and automate the management of composite cloud applications in a fault-resilient manner.

Notice that, even if the components of an application are described by fault-aware management protocols, the actual behaviour of such components may differ from their described behaviour (e.g., because of non-deterministic bugs [23]). In [9], we also show how unexpected behaviour can be naturally modelled by automatically completing fault-aware management protocols, and how this permits analysing the (worst-possible) effects of misbehaving components on the rest of a cloud application. We also propose a way to recover composite cloud applications that are stuck because a fault was not properly handled, or because of a misbehaving component.

Discussion. Fault-aware management protocols can play a foundational role for modelling and analysing the management of composite cloud applications. Indeed, to the best of our knowledge, they constitute the first compositional approach that permits modelling and analysing the stateful management behaviour of the components forming an application, by also taking into account that faults possibly occur while managing complex composite applications over heterogeneous clouds [30].

The feasibility of approaches based on fault-aware management protocols have been illustrated in [9], where we present the Barrel, a web-based application that permits editing and analysing fault-aware management protocols in composite cloud applications. In [9], we also illustrate the usefulness of such approaches, by showing how fault-aware management protocols (and Barrel as

[2] The same does not hold for the validation of the structure of TOSCA-based applications, for which approaches have been proposed in [12, 17].

well) were exploited in a concrete case study and in a controlled experiment. Both the case study and the controlled experiment highlight that the modelling and analysis techniques based on fault-aware management protocols can be fruitfully exploited not only at design time (to validate management plans, and to determine their effects), but also at run time (to automatically determine the management plans that permit reaching a desired application configuration, or which restore such configuration after the actual configuration has changed — e.g., because of faults or misbehaving components).

On the other hand, a full-fledged approach for modelling and analysing composite cloud applications requires to solve also other problems that we have not tackled yet. Below we discuss three of them, namely faults due to the "true concurrent" execution of management operations, applications whose topology is dynamic, and including cost and QoS in the modelling of applications.

Faults due to "true concurrency". The current version of fault-aware management protocols (as per [7, 9]) focuses on ensuring the consistency of a composite cloud application based on an interleaving semantics. This means that fault-aware management protocols do not support the analysis of the true concurrent execution of management operations.

Consider for instance the concurrent reconfiguration of two components, with one component requiring a capability of the other to successfully complete its reconfiguration. The latter may however stop providing the desired capability during the execution of its reconfiguration operation, even if such capability is provided right before and after executing such operation. While this may result in one of the two reconfiguration operations failing, an interleaving semantics (checking consistency only on states) would not be able to detect such failure.

In other words, faults can happen both after and *during* the concurrent execution of management operations [19]. It would be worthy investigating whether the latter case might generate problems in real-world scenarios, and (if this is the case) how to properly adapt fault-aware management protocols to cope with the "true concurrent" management of the components forming an application.

Dynamic topologies. Fault-aware management protocols can be easily adapted to cope with applications whose topology is dynamic. Indeed, to deal with applications whose components may dynamically (dis)appear, such components should be added to the application topology, and the binding function relating requirements and capabilities should be updated accordingly. This would be useful, for instance, to cope with the horizontal scaling of an application's components, as it would permit adding or removing replicas of a component to the application topology whenever such component has to be scaled out or scaled in.

Adapting our modelling and analysis techniques to cope with dynamic topologies would also be beneficial for exploiting them to manage other kinds of applications, which are characterised by a high churn of nodes (e.g., microservices-based applications, or fog applications).

Modelling cost and QoS. Fault-aware management protocols do not take into account costs nor QoS, since our focus so far has been on automatically coordinating the management of the components forming a composite cloud application (by also taking into account that faults potentially occur while managing complex applications). Cost and QoS are however important factors for cloud applications [1], and fault-aware management protocols should hence be extended to take into account also such factors. For instance, we should permit modelling how much does it cost (in terms of money or time) to reside in a certain state or to perform a certain operation. This would permit devising analysis techniques to determine the cost (in terms of money or time) to maintain/drive an application in/to a given configuration, or to determine the cheapest or fastest management plans that permit changing the actual configuration of an application.

3 Fostering the reuse of cloud applications

To ease the design of a cloud application (e.g., a web application), we may wish to implement some of its parts by reusing already existing, validated solutions (e.g., the server stack needed to run a web application). More generally, the reuse of (fragments of) existing cloud applications can ease and speed-up the development of new applications.

In [14, 15], we illustrate how to reuse existing, TOSCA-based cloud applications. We first formally define how to *exact* match an application with respect to a desired application component, so as to reuse the whole application to implement such component.

We then define three other types of matching (*plug-in*, *renaming-based*, and *white-box*), each permitting to identify larger sets of applications that can be adapted so as to exactly match a desired component. We also illustrate a methodology for adapting matched applications to exactly match the target component.

Notice that, by reusing a cloud application in its entirety, we might deploy unnecessary software (i.e., software that is not needed to concretely implement the desired application component).

To tackle this issue, in [31] we further extend our matchmaking and adaptation approach by introducing and assessing ToscaMart (*TOSCA-based Method for Adapting and Reusing application Topologies*). ToscaMart is a method that permits reusing only the fragment of an application topology that is necessary for implementing a desired application component.

Notice that all notions of matching mentioned above are purely syntactic and do not take into account the behaviour of management operations (i.e., they do not check whether the behaviour of the operations of an available application is compatible with that of the operations of a desired application component).

We overcome this limitation in [4, 5] by exploiting the behaviour information in management protocols. Namely, we define when a desired management protocol can be *simulated* [29] by an available one, and we exploit such notion of simulation to extend the conditions constraining exact and plug-in matching. We then relax the notion of simulation into that of f-simulation (which permits simulating a desired operation with a sequence of available operations), and we exploit f-simulation to further relax plug-in matching. We also describe a coinductive procedure to compute the function f determining an f-simulation among two management protocols (if any), and how matched applications can be adapted so as to be employed in place of desired components.

Discussion The matching techniques discussed above can constitute a fruitful support to foster the reuse of composite cloud applications, and to speed-up their design and development. Developers can indeed describe only the application components that are specific to their solutions (e.g., those they implemented), along with abstract descriptions of the management infrastructures such components need to run. Such abstract descriptions could then be concretely implemented by matching, adapting, and reusing (fragments of) already existing solutions.

The feasibility and potential of our notions of matching have been tested by running a proof-of-concept implementation of the syntactic matching over a plastic repository of TOSCA applications (in [15, 31]). The effectiveness has also been discussed by formally proving termination and soundness of ToscaMart (in [31]), and by assessing the behaviour-aware matching (by formally proving termination, soundness, and completeness of the coinductive algorithm determining a f-simulation between two protocols — in [5]).

It is worth noting that, in general, most existing approaches to the reuse of cloud applications support a development from-scratch of applications, and do not account for the possibility of adapting existing third-party applications. As we discussed in [30], to the best of our knowledge, the syntactic matching approaches presented in [14, 15, 31] advance the state-of-the-art as they are the first approaches that permit reusing (fragments of) existing cloud applications, by relying on TOSCA as the reference standard for cloud interoperability, and to support an easy reuse of third-party services. The behaviour-aware matching in [4, 5] is also advancing the state-of-the-art on the reuse of composite cloud applications. Indeed, it constitutes the first approach for reusing composite applications that takes into account both functional and extra-functional features of their components, and which relies on the widely accepted idea of exploiting behaviour models to match operations and on behaviour simulation to abstract from non-relevant operation mismatches.

On the other hand, in order to select (fragments of) composite cloud applications that can be effectively reused to implement abstractly specified components, some problems have still to be investigated and addressed. Below we discuss three of them, namely the full integration of the proposed techniques, the assessment of the substitutability assumption made by TOSCA, and the inclusion of cost and QoS in our matching approaches.

Full integration. The behaviour-aware matching proposed in [4, 5] permits reusing applications only in their entirety. To permit reusing only the fragments of such applications that are actually necessary to implement a desired component, the current implementation of ToscaMart should be integrated with the behaviour-aware matching that we presented in [4, 5].

Furthermore, the behaviour-aware matching proposed in [4, 5] does not take into account faults, since the notions of simulation are defined on "plain" management protocols (viz., those defined in [6]). The notions of simulation and of behaviour-aware matching should be hence extended to permit comparing the fault-aware management protocol of (a fragment of) an available application with that of a desired component.

Substitutability assumption. Our notions of matching are based on the substitutability assumption made by TOSCA, which states that a component can be made concrete by substituting it with a composite application, provided that the latter exposes the same features as the former on its boundaries [27]. The truthfulness of such an assumption should be tested on repositories of real-word TOSCA applications, which unfortunately are not available at the moment.

Cost-aware and QoS-aware matching. Our notions of matching do not take into account costs nor QoS. This is because our focus so far has been on enacting the reuse of composite cloud applications by matching their syntactic signature and their management behaviour with respect to those of a desired component.

Cost and QoS are however important factors for cloud applications [1, 22]. Our notions of matching should hence be extended to take into account also the cost or QoS of composite cloud applications, as this would permit selecting, among the matched applications, those leading to lower costs or to better time performances, for instance.

4 Conclusions

Management protocols can play a foundational role for modelling and analysing the management of composite cloud applications. The feasibility of approaches based on management protocols has been illustrated with Barrel, while their potential has been shown by exploiting them to analyse and automate the deployment and management of a concrete case study (and with a controlled experiment — in [9]).

The matching and adaptation techniques we have proposed can also contribute to support a vendor-agnostic design of composite cloud applications, and to automate the deployment and management of composite applications. The feasibility and potential of our notions of matching have been tested, and their effectiveness has been formally assessed (see Sect. 3).

At the same time, a full-fledged support for modelling, analysing, and reusing composite cloud applications requires also solutions to problems that we have not yet been tackled (as discussed in Sects. 2 and 3). Some of the corresponding directions for future work are listed below.

Management protocols-based orchestration. An interesting direction for future work is the development of an orchestrator for composite cloud applications based upon management protocols. The orchestrator should input a composite application and its desired configuration, and it should ensure that such application configuration is reached and maintained. The orchestrator should exploit management protocols to determine the management plan leading the application to the desired configuration (from the initial situation where no application component is installed), and it should execute such management plan. Then, whenever a fault occurs and changes the actual application configuration (or whenever the desired configuration is changed), the orchestrator should automatically determine a management plan to restore the desired application configuration.

A first step towards the availability of such an orchestrator has been provided in [13], where we present an engine (called TosKer) for orchestrating the management of composite applications based on TOSCA and Docker. TosKer exploits management protocols to permit customising the management behaviour of the components forming an application, and to ensure that components are actually managed accordingly to their specified behaviour. It however still lacks a support for automatically reaching and maintaining the desired configuration of an application (from determining the management plan leading to the desired configuration, to reconfiguring it when its actual configuration is different from the desired one). The extension of TosKer for providing such support is part of our future work.

Faults due to "true concurrency". As we observed in Sect. 2, there might be cases where a requirement is assumed by a component, and another component executes a management operation during which the capability satisfying such requirement is not maintained (even if it is available before and after executing it). We plan to investigate whether this might generate problems in real-world scenarios, and how to properly adapt the composition rules defining the fault-aware management behaviour of a composite application. Preliminary results in this direction are illustrated in [10].

Dynamic topologies. To deal with applications whose components may dynamically (dis)appear, it should be enough to add such components to the application topology, and to update the binding function relating requirements and capabilities. Some results in this direction are contained in [8], where we show how to deal with topologies where the dependencies among a fixed set of components can dynamically change. Still, there is work to do on how to deal with topologies whose set of components can dynamically change. This is in the scope of our immediate future work.

Modelling cost and QoS. The modelling and analyses techniques based upon fault-aware management protocols do not take into account costs and QoS. The extension of such techniques to include cost and QoS properties is in the scope of our future work.

Full integration of the proposed matching techniques. Our behaviour-aware matching approach does not take into account faults, and it permits reusing applications only in their entirety (see Sect. 3). We plan to extend our behaviour-aware matching approach to permit comparing fault-aware management protocols (hence including faults in the comparison), and to integrate it with ToscaMart (to permit reusing only the fragments of application that are actually necessary for implementing a desired component).

Substitutability assumption. The truthfulness of the substitutability assumption made by TOSCA (which states that a component can be made concrete by substituting it with a composite application, provided that the latter exposes the same features as the former on its boundaries [27]) should be tested on repositories of real-word TOSCA applications, which unfortunately are not available at the moment. The assessment of such assumption is hence left for future work.

Cost-aware and QoS-aware matching. Our notions of matching do not take into account costs or QoS. The extension of such notions to include QoS and costs in the selection of to-be-reused components are in the scope of our future work.

Acknowledgments. Special thanks to Antonio Brogi, who made me discover the world of scientific research. He taught me what scientific research is, how to carry it out, and especially how to always enjoy doing research. Without his guidance and persistent help my Ph.D. thesis would not have been possible.

Thanks are also due to Andrea Canciani, Antonio Cisternino, Claus Pahl, Filippo Bonchi, Frank Leymann, Marco Danelutto and Pengwei Wang. Andrea, Filippo, and Pengwei actively cooperated with me, and their help was precious to reach some of the results in my Ph.D. thesis. Antonio and Marco were instead exposed several times to the progresses of my research work, and they were always giving me valuable feedback, especially concerning the research path to follow. Frank and Claus kindly hosted me as a visiting researcher, helping me in growing as a researcher.

Finally, thanks are due the members of the committee of SummerSoC, who granted me the honour of receiving the Christos Nikolau Memorial Ph.D. Award (and who have given me the possibility of writing this paper disseminating the results of the research I have been doing so far).

References

1. Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., Zaharia, M.: A view of cloud computing. Communications of the ACM **53**(4), 50–58 (2010). https://doi.org/10.1145/1721654.1721672
2. Binz, T., Breitenbücher, U., Kopp, O., Leymann, F.: TOSCA: Portable Automated Deployment and Management of Cloud Applications, pp. 527–549. Springer, New York, NY (2014). https://doi.org/10.1007/978-1-4614-7535-4_22
3. Binz, T., Fehling, C., Leymann, F., Nowak, A., Schumm, D.: Formalizing the Cloud through Enterprise Topology Graphs. In: Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on. pp. 742–749. IEEE Computer Society (2012). https://doi.org/10.1109/CLOUD.2012.143
4. Bonchi, F., Brogi, A., Canciani, A., Soldani, J.: Behaviour-aware matching of cloud applications. In: Proceedings of the 10th International Symposium on Theoretical Aspects of Software Engineering, TASE 2016. pp. 117–124. IEEE (2016). https://doi.org/10.1109/TASE.2016.32

5. Bonchi, F., Brogi, A., Canciani, A., Soldani, J.: Simulation-based matching of cloud applications. Science of Computer Programming **162**, 110 – 131 (2018). https://doi.org/10.1016/j.scico.2017.06.001, special Issue on TASE 2016
6. Brogi, A., Canciani, A., Soldani, J.: Modelling and analysing cloud application management. In: Dustdar, S., Leymann, F., Villari, M. (eds.) Service-Oriented and Cloud Computing: 4th European Conference, ESOCC 2015, Taormina, Italy, September 15-17, 2015, Proceedings. pp. 19–33. Springer (2015). https://doi.org/10.1007/978-3-319-24072-5_2
7. Brogi, A., Canciani, A., Soldani, J.: Fault-aware application management protocols. In: Aiello, M., Johnsen, B.E., Dustdar, S., Georgievski, I. (eds.) Service-Oriented and Cloud Computing: 5th IFIP WG 2.14 European Conference, ESOCC 2016, Vienna, Austria, September 5-7, 2016, Proceedings. pp. 219–234. Springer (2016). https://doi.org/10.1007/978-3-319-44482-6_14
8. Brogi, A., Canciani, A., Soldani, J.: Modelling the dynamic reconfiguration of application topologies, faults included. In: Jacquet, J.M., Massink, M. (eds.) Coordination Models and Languages. LNCS, vol. 10319, pp. 178–196. Springer International Publishing (2017)
9. Brogi, A., Canciani, A., Soldani, J.: Fault-aware management protocols for multi-component applications. Journal of Systems and Software **139**, 189 – 210 (2018). https://doi.org/10.1016/j.jss.2018.02.005
10. Brogi, A., Canciani, A., Soldani, J.: True concurrent management of multi-component applications. In: Kritikos, K., Plebani, P., De Paoli, F. (eds.) Service-Oriented and Cloud Computing, ESOCC 2018, Proceedings. Springer (2018)
11. Brogi, A., Canciani, A., Soldani, J., Wang, P.: A Petri net-based approach to model and analyze the management of cloud applications. In: Koutny, M., Desel, J., Kleijn, J. (eds.) Transactions on Petri Nets and Other Models of Concurrency XI. pp. 28–48. LNCS Transactions on Petri Nets and Other Models of Concurrency, Springer Berlin Heidelberg (2016). https://doi.org/10.1007/978-3-662-53401-4_2
12. Brogi, A., Di Tommaso, A., Soldani, J.: Sommelier: A tool for validating tosca application topologies. In: Pires, L.F., Hammoudi, S., Selic, B. (eds.) Model-Driven Engineering and Software Development. pp. 1–22. Springer International Publishing (2018)
13. Brogi, A., Rinaldi, L., Soldani, J.: Tosker: A synergy between tosca and docker for orchestrating multi-component applications. Software: Practice and Experience . https://doi.org/10.1002/spe.2625, *[In press]*
14. Brogi, A., Soldani, J.: Matching cloud services with TOSCA. In: Canal, C., Villari, M. (eds.) Advances in Service-Oriented and Cloud Computing: Workshops of ESOCC 2013, Málaga, Spain, September 11-13, 2013, Revised Selected Papers. pp. 218–232. Springer (2013). https://doi.org/10.1007/978-3-642-45364-9_18
15. Brogi, A., Soldani, J.: Finding available services in TOSCA-compliant clouds. Science of Computer Programming **115–116**, 177–198 (2016). https://doi.org/10.1016/j.scico.2015.09.004
16. Brogi, A., Soldani, J., Wang, P.: TOSCA in a nutshell: Promises and perspectives. In: Villari, M., Zimmermann, W., Lau, K.K. (eds.) Service-Oriented and Cloud Computing: 3rd European Conference, ESOCC 2014, Manchester, UK, September 2-4, 2014. Proceedings. pp. 171–186. Springer (2014). https://doi.org/10.1007/978-3-662-44879-3_13
17. Brogi, A., Tommaso, A.D., Soldani, J.: Validating tosca application topologies. In: Proceedings of the 5th International Conference on Model-Driven Engineering and Software Development - Volume 1: MODELSWARD,. pp. 667–678. SciTePress (2017). https://doi.org/10.5220/0006244006670678

18. Buyya, R., Yeo, C.S., Venugopal, S., Broberg, J., Brandic, I.: Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. Future Generation Computer Systems **25**(6), 599 – 616 (2009). https://doi.org/10.1016/j.future.2008.12.001
19. Cook, R.I.: How complex systems fail. Cognitive Technologies Laboratory, University of Chicago (1998), URL: http://web.mit.edu/2.75/resources/random/How Complex Systems Fail.pdf
20. Di Cosmo, R., Mauro, J., Zacchiroli, S., Zavattaro, G.: Aeolus: A component model for the cloud. Information and Computation **239**(0), 100 – 121 (2014). https://doi.org/10.1016/j.ic.2014.11.002
21. Fehling, C., Leymann, F., Retter, R., Schupeck, W., Arbitter, P.: Cloud Computing Patterns: Fundamentals to Design, Build, and Manage Cloud Applications. Springer (2014). https://doi.org/10.1007/978-3-7091-1568-8
22. Gartner: Gartner identifies the top 10 strategic technologies for 2011. analysts examine latest industry trends during gartner symposium/itxpo, October 17-21, in Orlando. Press Release (2010)
23. Gray, J.: Why do computers stop and what can be done about it? In: Proceedings of the 5th symposium on Reliability in Distributed Software and Database Systems. pp. 3–12. Los Angeles, CA, USA (1986)
24. Leymann, F.: Cloud computing. it — Information Technology, Methoden und innovative Anwendungen der Informatik und Informationstechnik **53**(4), 163–164 (2011). https://doi.org/10.1524/itit.2011.9070
25. Mell, P.M., Grance, T.: SP 800-145. The NIST Definition of Cloud Computing. Tech. rep., Gaithersburg, MD, United States (2011)
26. OASIS: Topology and Orchestration Specification for Cloud Applications. URL: http://docs.oasis-open.org/tosca/TOSCA/v1.0/TOSCA-v1.0.pdf (2013)
27. OASIS: Topology and Orchestration Specification for Cloud Applications (TOSCA) Primer. URL: http://docs.oasis-open.org/tosca/tosca-primer/v1.0/tosca-primer-v1.0.pdf (2013)
28. Petcu, D., Macariu, G., Panica, S., Crciun, C.: Portable cloud applications - from theory to practice. Future Generation Computer Systems **29**(6), 1417 – 1430 (2013). https://doi.org/10.1016/j.future.2012.01.009
29. Sangiorgi, D.: Introduction to Bisimulation and Coinduction. Cambridge University Press, New York, NY, USA (2011)
30. Soldani, J.: Modelling, analysing and reusing composite cloud applications. Ph.D. thesis, University of Pisa (2017), https://etd.adm.unipi.it/theses/available/etd-03242017-175216/unrestricted/thesis.pdf
31. Soldani, J., Binz, T., Breitenbcher, U., Leymann, F., Brogi, A.: ToscaMart: A method for adapting and reusing cloud applications. Journal of Systems and Software **113**, 395–406 (2016). https://doi.org/10.1016/j.jss.2015.12.025

The SmartOrchestra Platform: A Configurable Smart Service Platform for IoT Systems

Andreas Liebing[1], Lutz Ashauer[1], Uwe Breitenbücher[2], Thomas Günther[3], Michael Hahn[2], Kálmán Képes[2], Oliver Kopp[2], Frank Leymann[2], Bernhard Mitschang[2], Ana C. Franco da Silva[2], and Ronald Steinke[3]

[1] StoneOne AG, Berlin, Germany
andreas.liebing@stoneone.de
[2] University of Stuttgart, Stuttgart, Germany
[firstname.lastname]@informatik.uni-stuttgart.de
[3] Fraunhofer FOKUS, Berlin, Germany
[firstname.lastname]@fokus.fraunhofer.de

Abstract. The Internet of Things is growing rapidly while still missing a universal operating and management platform for multiple diverse use cases. Such a platform should provide all necessary functionalities and the underlying infrastructure for the setup, execution and composition of Smart Services. The concept of Smart Services enables the connection and integration of cyber-physical systems (CPS) and technologies (i.e., sensors and actuators) with business-related applications and services. Therefore, the *SmartOrchestra Platform* provides an open and standards-based service platform for the utilization of public administrative and business-related Smart Services. It combines the features of an operating platform, a marketplace, a broker, and a notary for a cloud-based operation of Smart Services. Thus, users of cyber-physical systems are free to choose their control applications, no matter what device they are using (e.g., smartphone, tablet or personal computer) and they also become independent of the manufacturers' software. This will enable new business opportunities for different stakeholders in the market and allows flexibly composing Smart Services.

Keywords: SmartOrchestra Platform, Smart Services, Cyber-Physical Systems, Internet of Things

1 Introduction

The Internet of Things (IoT) paradigm has received great attention in the last years leading to a vast amount of heterogeneous IoT middleware, protocols and devices (e.g., sensors, actuators or gateways). Related IoT applications, for example, implementing the processing of sensor data in order to control an actuator, therefore need to be bound to certain concrete technologies, hardware devices, and protocols. This makes it a challenge to enable the interoperability and composition of different IoT applications, for example, to compose them to solve problems on another scale (e.g., from automating houses over streets to cities). Another challenge is the distributed nature of IoT environments and the large number of devices, which makes it infeasible to deploy and manage IoT applications together with their required software and middleware components manually. Therefore, a universal operating and management platform for multiple diverse IoT use cases is needed which enables interoperability and automated deployment for IoT applications through Smart Services. The concept of Smart Services enables the connection and integration of cyber-physical systems (CPS) and technologies (i.e., sensors and actuators) with business-related applications and services. Such a platform should provide all necessary functionalities and the underlying infrastructure for the setup, execution and composition of Smart Services. Within this work, we introduce the ***SmartOrchestra Platform***, which provides an open and standards-based service platform for the utilization of public administrative and business-related Smart Services. Therefore, it combines the features of an operating platform, a marketplace, a broker, and a notary for a cloud-based operation of Smart Services. Thus, users of cyber-physical systems are free to choose their control applications, no matter what device they are using (e.g., smartphone, tablet, or personal computer) and they also become independent of the manufacturers' software. This will enable and provide new business opportunities for different stakeholders in the market and allows flexibly utilizing and composing Smart Services.

The remainder of the paper starts with an introduction of the SmartOrchestra Platform and its major building blocks in Sect. 2. This is followed by a description how the introduced components of the platform work together to provide a universal operating and management platform for multiple diverse IoT use cases in Sect. 3. Finally, a summary of the paper is given in Sect. 4.

2 The SmartOrchestra Platform

The SmartOrchestra Platform enables a uniform service description as well as a secure and safe internet-based composition and integration of heterogeneous cyber-physical systems and services based on standardized cloud and orchestration technologies. The platform serves both, a transparent catalog to evaluate suitable services from a widespread ecosystem as well as an operational platform and interface between control devices and sensor units with their respective applications. In this way, the platform will be an open, secure, and standardized Smart Service Platform.

The conceptual design of the SmartOrchestra Platform and its major building blocks in combination with provisioning workflows and IoT devices is depicted in Fig. 1. The main entry point to the platform is the Marketplace, which allows users to evaluate, run and compose existing services from the Service Catalog as well as provide and market new services. To deploy and configure Smart Services OpenTOSCA [2] is used as Provisioning Engine. The Provisioning Engine is responsible for the automated deployment of a Smart Service and the configuration of its underlying infrastructure. For example, this can comprise the installation of related software services in the Cloud providing the business logic of a Smart Service (e.g., data filtering, processing, or aggregation [13]), the configuration of IoT devices and gateways, and installing required software on them.

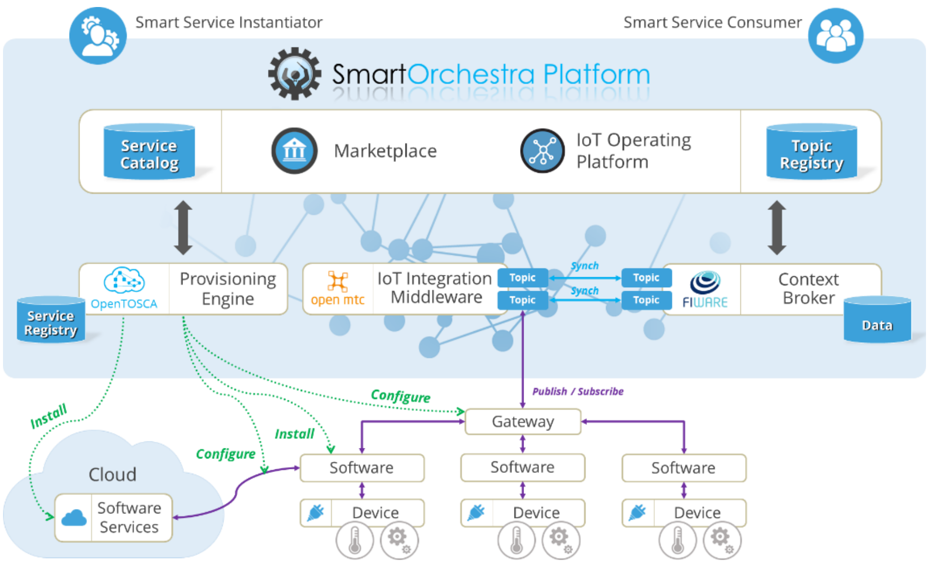


Fig. 1. SmartOrchstra Platform Architecture

For the integration of IoT devices into Smart Services, the SmartOrchestra Platform uses OpenMTC [6, 15] as IoT Integration Middleware [16, 17]. OpenMTC provides required protocols and adapters to integrate and mediate between the heterogeneous devices, sensors, and actuators within the SmartOrchestra Platform. Therefore, it comes with an embedded service layer that enables communication between devices through a Publish/Subscribe model. Furthermore, the platform provides the FIWARE Orion Context Broker [9] as Context Broker. While OpenMTC is responsible for enabling the communication between devices, sensors and actuators through corresponding generic interfaces, the FIWARE Orion Context Broker in combination with the FIWARE Short Time Historic [10] is used as a midterm data repository to enable later analysis of data provided by IoT devices. Therefore, OpenMTC synchronizes all published data to respective entities at the FIWARE broker as depicted by the synch arrows in Fig. 1.

In the following, each of the building blocks of the platform is described in more detail. Furthermore, the interplay of the components is outlined in order to give an idea how the overall SmartOrchestra Platform operates.

2.1 Marketplace and IoT Operating Platform

The open and secure SmartOrchestra marketplace brings together and combines intelligent private, industrial, and municipal Smart Services. This results in new innovative services that make data available for use. The marketplace allows for browsing, choosing, and configuring of Smart Services. Data of each running service can also be accumulated and visualized in the marketplace via customer specific dashboards including widgets and configurable taxonomies for structured presentation. Smart Services and/or data channels can be combined and orchestrated by rules and actions. The parameters of devices or services can be changed during runtime of the service. This turns the marketplace into an IoT Operating Platform.

2.2 Provisioning Engine: OpenTOSCA

The SmartOrchestra Platform enables the deployment of Smart Services by employing OpenTOSCA [2, 12] as Provisioning Engine. OpenTOSCA is an open-source ecosystem for the modeling, provisioning, and management of cloud applications based on the OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA) standard [1, 3, 21] and supporting concepts for CPS and IoT [19, 22–24]. In TOSCA, the structure of an application, e.g., a Smart Service, is described using topology models. These models are represented as graphs containing typed nodes and directed typed edges. Nodes, called *node templates*, represent the software components of a Smart Service, while edges, called *relationship templates*, describe the relationships among the components, e.g., dependencies and connections.

TOSCA offers two approaches to application provisioning: (i) a declarative approach and (ii) an imperative approach [4, 5]. In the declarative approach, only the topology model has to be provided to a provisioning engine, which implicitly knows how to set up the application components. More precisely, it is sufficient to describe what needs to be provisioned, and not how this needs to be done. In contrast, the imperative approach relies on explicitly describing how an application has to be hosted. To realize this, a so called Build Plan has to be provided that describes, which steps have to be executed to set up the components of the topology.

The OpenTOSCA ecosystem is composed of following: (i) the graphical TOSCA modeling tool Eclipse Winery [20] and (ii) the TOSCA runtime environment OpenTOSCA container. Once the topology of a Smart Service is modeled using Eclipse Winery, it can be optionally checked against a collection of compliance rules [7] and exported as a Cloud Service Archive (CSAR), which can be deployed into the OpenTOSCA container to provision and instantiate the Smart Service. The provisioning of Smart Services can be secured by the specification of non-functional requirements through policies [18].

OpenTOSCA allows for an easy integration with other systems through its provided APIs, which offer the main functionalities to automatically provision applications using the OpenTOSCA container, and afterwards, retrieve information about the instantiated applications. The Marketplace, which is the user's entry point to the SmartOrchestra Platform, is integrated with OpenTOSCA through the provided OpenTOSCA API. In this way, the Marketplace can, for example, configure and automatically provision Smart Services, and retrieve information about all available Smart Services.

2.3 IoT Integration Middleware: OpenMTC

The OpenMTC platform [6, 15] is an open-source implementation of the oneM2M standard[1], which intends to support machine-to-machine (M2M) communication for applications in a simplified way. Furthermore, OpenMTC is as well available as a Generic Enabler in the FIWARE Catalogue [14].

OpenMTC consists of a gateway and backend component that provides a REST API and uses the CRUD principle for managing resources. Through protocol adapters, OpenMTC is able to interact with various devices of different technologies. Thus, information from heterogeneous data sources will be unified in a harmonized data model, so that applications can easily access the data without the need to know the device specific technologies. Furthermore, data can be already preprocessed close to the source before they are send to other endpoints.

OpenMTC has a generic request/response model that can be mapped on different transport protocols, e.g., HTTP or MQTT. The provided functionality includes registration of applications, discovery of resources, subscription to new data, simplified addressing of resources, scheduled communication, and more [11].

2.4 Context Broker: FIWARE Orion Context Broker

The Orion Context Broker [9] is the main component of the FIWARE platform [8]. Through its REST API, the Broker allows the registration of *context elements*, which can be updated by *context producers*. Furthermore, *context consumers* can either query these context elements or subscribe to them to get notifications when the context elements are updated [11]. The Orion Context Broker can be configured through the marketplace to automatically work together with another FIWARE Generic Enabler, the FIWARE Short Time Historic [10], which is used for midterm storage of data.

3 Interplay of the SmartOrchestra Platform Components

In this section, we describe the interaction with the platform and the interplay of the platform components. Consequently, the interplay is described based on the different roles that interact with the SmartOrchestra Platform (cf. Fig. 1): Smart Service Instantiators and Smart Service Consumers.

The role of a *Smart Service Instantiator* is to provision different Smart Services from within the *Service Catalog* using the provided interfaces given by the IoT Operating Platform. These services can then be used by the Smart Service Consumers for their own applications. The Smart Service Instantiator is responsible for providing relevant data, such as credentials and endpoints, to enable access to the *IoT Devices* for the Provisioning Engine and therefore to enable the installation of different software components. To realize the integration of the IoT Devices into the platform, these software components, such as *adapters* and *gateways*, are responsible for binding the used hardware to the IoT Integration Middleware by sending the relevant data from the sensors

[1] http://www.onem2m.org/

and enabling the invocation of operations on actuators. As stated in the previous section, the IoT Integration Middleware is used as the first layer to integrate heterogeneous IoT hardware, while the Context Broker component is used as a second layer to store data from the sensors for midterm data analysis within the SmartOrchestra Platform.

The synchronization of the data across the IoT Integration Middleware and the Context Broker is based on a bi-directional exchange between respective entities managed by each of the components. These entities are automatically created when the provisioning of the software components is finished and therefore require no additional effort from the Smart Service Instantiator. After the provisioning and as soon as OpenMTC and the FIWARE broker receive data from sensors and actuators, a *Smart Service Consumer* is able to subscribe on the available data or issue commands through corresponding topics provided by the *Topic Registry* of the platform. Based on that, Smart Service Consumers are able to integrate Smart Services operated and managed through the SmartOrchestra Platform into their business applications.

4 Summary

This work presented the SmartOrchestra Platform as an enabler for interoperability, automated deployment and composition of IoT applications through Smart Services. Thereby, the different components of the platform build on well-established standards such as TOSCA for application provisioning and management or oneM2M for machine-to-machine communication and IoT. The result is an open and standards-based platform for the utilization of public administrative and business-related Smart Services by combining the features of an operating platform, a marketplace, a broker, and a notary for a cloud-based operation of Smart Services.

Acknowledgments This work is funded by the BMWi project SmartOrchestra (01MD16001F).

References

1. Bergmayr, A., Breitenbücher, U., Ferry, N., Rossini, A., Solberg, A., Wimmer, M., et al.: A Systematic Review of Cloud Modeling Languages. ACM Computing Surveys. 51, 1, (2018).
2. Binz, T., Breitenbücher, U., Haupt, F., Kopp, O., Leymann, F., Nowak, A., et al.: OpenTOSCA – A Runtime for TOSCA-based Cloud Applications. In: ICSOC. (2013).
3. Binz, T., Breiter, G., Leymann, F., Spatzier, T.: Portable Cloud Services Using TOSCA. IEEE Internet Computing. 16, 03, (2012).
4. Breitenbücher, U., Binz, T., Képes, K., Kopp, O., Leymann, F., Wettinger, J.: Combining Declarative and Imperative Cloud Application Provisioning based on TOSCA. In: IC2E. IEEE (2014).

5. Breitenbücher, U., Képes, K., Leymann, F., Wurster, M.: Declarative vs. Imperative: How to Model the Automated Deployment of IoT Applications? In: SummerSOC. IBM Research Report (2017).
6. Corici, M., Coskun, H., Elmangoush, A., Kurniawan, A., Mao, T., Magedanz, T., et al.: OpenMTC: Prototyping Machine Type communication in carrier grade operator networks. In: IEEE Globecom Workshops. (2012).
7. Fischer, M.P., Breitenbücher, U., Képes, K., Leymann, F.: Towards an Approach for Automatically Checking Compliance Rules in Deployment Models. In: SECURWARE. Xpert Publishing Services (2017).
8. FIWARE: FIWARE Catalogue, https://www.fiware.org/developers/catalogue/.
9. FIWARE: Orion Context Broker, https://www.github.com/telefonicaid/fiware-orion.
10. FIWARE: Short Time Historic (STH) – Comet, https://github.com/telefonicaid/fiware-sth-comet.
11. Franco da Silva, A.C., Breitenbücher, U., Hirmer, P., Képes, K., Kopp, O., Frank Leymann, et al.: Internet of Things Out of the Box: Using TOSCA for Automating the Deployment of IoT Environments. In: CLOSER. (2017).
12. Franco da Silva, A.C., Breitenbücher, U., Képes, K., Kopp, O., Leymann, F.: OpenTOSCA for IoT: Automating the Deployment of IoT Applications based on the Mosquitto Message Broker. In: IoT. ACM (2016).
13. Franco da Silva, A.C., Hirmer, P., Breitenbücher, U., Kopp, O., Mitschang, B.: Customization and provisioning of complex event processing using TOSCA. Computer Science - Research and Development. (2017).
14. Fraunhofer FOKUS: OpenMTC Generic Enabler, https://catalogue.fiware.org/enablers/openmtc.
15. Fraunhofer FOKUS: OpenMTC Platform Architecture, http://www.openmtc.org/index.html#MainFeatures.
16. Guth, J., Breitenbücher, U., Falkenthal, M., Fremantle, P., Kopp, O., Leymann, F., et al.: A Detailed Analysis of IoT Platform Architectures: Concepts, Similarities, and Differences. Presented at the (2018).
17. Guth, J., Breitenbücher, U., Falkenthal, M., Leymann, F., Reinfurt, L.: Comparison of IoT Platform Architectures: A Field Study based on a Reference Architecture. In: CIoT. IEEE (2016).
18. Képes, K., Breitenbücher, U., Fischer, M.P., Leymann, F., Zimmermann, M.: Policy-Aware Provisioning Plan Generation for TOSCA-based Applications. In: SECURWARE. Xpert Publishing Services (2017).
19. Képes, K., Breitenbücher, U., Leymann, F.: Integrating IoT Devices Based on Automatically Generated Scale-Out Plans. In: SOCA. IEEE (2018).
20. Kopp, O., Binz, T., Breitenbücher, U., Leymann, F.: Winery – A Modeling Tool for TOSCA-based Cloud Applications. In: ICSOC. (2013).
21. OASIS: Topology and Orchestration Specification for Cloud Applications (TOSCA) Version 1.0. (2013).
22. Saatkamp, K., Breitenbücher, U., Leymann, F., Wurster, M.: Generic Driver Injection for Automated IoT Application Deployments. In: iiWAS. ACM (2017).

23. Franco da Silva, A.C., Hirmer, P., Breitenbücher, U., Kopp, O., Mitschang, B.: TDLIoT: A Topic Description Language for the Internet of Things. In: ICWE. Springer Berlin Heidelberg (2018).
24. Zimmermann, M., Breitenbücher, U., Leymann, F.: A TOSCA-based Programming Model for Interacting Components of Automatically Deployed Cloud and IoT Applications. In: ICEIS. SciTePress (2017).

All links were last followed on July 18, 2018.

Distributed Access Control for the Internet of Things[⋆]

Tobias Straub[1,2] and Ulf Schreier[1]

[1] Furtwangen University {tobias.straub,ulf.schreier}@hs-furtwangen.de
[2] University of Stuttgart tobias.straub@ipvs.uni-stuttgart.de

Abstract. The progressive miniaturization of computer-based devices and the ability to monitor an environment or perform demanding autonomous tasks revolutionize the way we live, learn and work today and in the future. Cyber-physical systems with the ability to communicate in various types form the Internet of Things (IoT). IoT is a paradigm which is already today used as a backbone technology in different application domains, for example, in Industry 4.0 or Smart Home. In order to prevent unauthorized access to resources on devices, the permissions of the requesting agent have to be checked. Centralized access control is one strategy for this task, but it prevents direct communication from device to device as desired in IoT. This paper describes the today's IoT environments by example, shows up the problems establishing access control in such environments and outlines a solution. It introduces a distributed access control approach that works close to IoT devices with often unused computational power and that enables direct communication from device to device as desired in IoT.

Keywords: Access control · Internet of Things · Security.

1 Introduction

Monitoring an environment and performing tasks in such an environment needs close interaction between actors [14]. Interaction, in turn, has various types of actors and often requires different forms of communication. For example, device to device, device to and from an application or device to and from distributed storage. Furthermore, communication is often not restricted to local networks. It can also happen via heterogeneous networks [1,2]. Nevertheless, most computer-based devices today cannot communicate with each other or only in an inadequate way. The insufficient fashion to communicate is why the idea of the Internet of Things (IoT) gains more and more interest over the last years. IoT can be considered as a radical evolution of the Internet. From providing human interconnections into a network of interconnected devices [6,2].

⋆ This research was partially funded by the Ministry of Science of Baden-Württemberg, Germany, for the Doctoral Program "Services Computing" (http://www.services-computing.de/?lang=en).

There are two general options available to implement access control in IoT environments which are described in the problem statement of this paper. Both of these centralize the access control system. This centralization means all access requests must go through a centralized access control instance. Direct communication from device to device as desired in IoT is not possible. This paper describes the drawbacks of centralizing access control to secure IoT environments and outlines a solution.

2 Fundamentals

The Internet of Things has several fundamental aspects [7,14]: Sensing through sensors in the embedded device, heterogeneous access to connect with different networks and devices, application and services such as security and privacy.

2.1 Internet of Things by example

For 2025, the installed base of IoT devices is forecast to grow to more than 75 billion worldwide [13]. However, already today exists several application domains which use the IoT as a backbone technology. For example, Industry 4.0, Smart City, Smart Home or the Ambient Assisted Living which is supported by Smart Home.

With 21%, a significant amount of IoT-solution vendors focus on the application domain of Smart Home to enhance the lifestyle of individuals by providing new values in fashions, which have not possible before [16]. IoT as backbone technology reimagines how to engage with homeowners, guests, and services which are provided by or through Things.

In many cases, Smart Home environments perform simple tasks, like lowering or upping shutters. However, far more complicated scenarios are possible which need a tight integration of a variety of devices. For example, the sensors of a smart power socket in a smart home environment might detect there is no voltage applied to the socket. The power socket then might communicate with other power sockets in the surrounding environment and figure out, that there is no voltage applied to the sockets in the kitchen. The Smart Home can now connect with the digital appointment system of an electrician to set up a next day service. It might connect to the calendar of the homeowners and perceive, that all homeowners are at work at the time an appointment with the electrician is available. Due to the urgency, the appointment will still be booked. Concurrently, the smart home issues a one-time front door access token to make it possible for the electrician to open the front door and deactivate the security system with his cell phone within the appointed time by Near-field communication. Simultaneously, the smart home informs the homeowners to go out for dinner today evening because the stove is not available, due to the lack of electricity. It might also appoint a breakfast for the next day in the favorite restaurant of the homeowners and set up the alarm clock to ring 30 minutes earlier than usual. Furthermore, the smart home sends the address of the restaurant to the cars navigation system. On the way, the homeowners can monitor the progress of the electrician through the security camera in the kitchen.

2.2 Access Control

Today, there exists a wide range of channels by which a user wants to access IT landscapes. For example, by single sign-on with a social network account, or by an application on mobile devices or in a cloud. An access control system must discover and check all kind of access requests. Further, security requirements for IT landscapes specified by companies and private households must be observed, too. Access control systems must grant or deny access to a requested resource by evaluating the defined authorization and access rules. A resource is thereby an entity that contains the information the user wants to access, for example, a file, a database or a program. Access control as concept and systems as the implementation of the concept seeks thereby to prevent activity that could lead to a breach of security.

Several variations of access control are available [4,3,5]. One frequently used variation is Role-based Access Control (RBAC). This variation reduces the complexity of security administration by generalizing subjects under a role property. In access control, a subject is a user or a program which is executing on behalf of the user accessing resources through specific actions. In RBAC a subject is replaced by one or more roles. Each user is assigned to one or more roles. Each role, in turn, is assigned with privileges that are permitted to all users in that role. Every user in a particular role can then perform the same set of rights on the system [5].

Access control will be problematic if security administrators have a demand for fine-grained rules. With RBAC this quickly leads to a role explosion if roles are created to fit each use case. Attribute-based Access Control (ABAC) [11] offers a more flexible and multi-dimensional approach to access control. This approach is realized by enforcing security decisions based on attributes accessible within the application. An attribute could be user data, information about the action the user is performing, properties of the resource the user is manipulating or retrieving or it could be an environment variable.

3 Problem Statement

In IoT environments, there are two general options to implement access control. The first option is to implement it on the same network of the IoT environment. Access control is centralized and all the access requests go through the centralized access control instance before the request reaches the requested resource. Direct communication from device to device as desired in IoT is not possible in this case. The centralized authorization component always acts as a middleman for every request. Fig. 1 shows this approach. On the left side of the figure, a user or a program which is executing on behalf of the user wants to access one of the devices on the right side of the figure. In this example, there is a smart door lock, a door light, and a door security camera are pictured. Access control is implemented as a central component of the requesting subject and the devices.

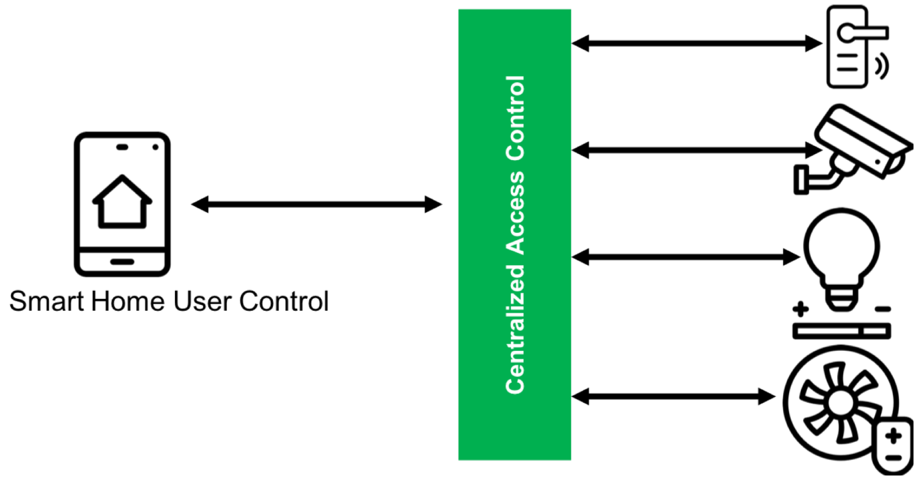


Fig. 1. The centralized access control instance acts as a middleman for every access request to IoT devices and prevents a direct communication from device to device as desired in IoT.

The second practice is to implement access control on a network on top of the actual IoT system. For example, the IoT system is managed by a cloud application administrated by the provider of the smart home environment. However, even with this practice, there is a centralized authorization instance that all requests must pass (Fig. 1), and that has many disadvantages.

First of all, in a centralized access control system, the complete information needed for the evaluation of access rules must be stored centrally. While this was still manageable with traditional access control variations, it is a problem when working with ABAC as an access control mechanism, because the number of attributes used in a condition is not limited. The amount of always available information in the centralized instance is thus dependent on local attributes used for the evaluation.

Another disadvantage is that the process of approving or denying the request must be done on the central authorization instance. This causes two problems. First, it is a single point of failure. If the centralized access control instance fails, resource access is not possible and hence all resources are effectively unavailable. The second problem is performance. The central instance might become a bottleneck when vast amounts of users, devices, and overall attributes must be supported.

One more disadvantage, especially in larger organizations, is the need for a particular trusted user like an administrator to modify access privileges for a resource. In most cases, especially in application domains like smart homes, it makes more sense to let the owner of the resource decide about access privileges.

All the mentioned drawbacks of the centralized access control are a problem for scalability. But scalability is a crucial factor for IoT.

4 Solution Approach

The mentioned drawbacks of centralized authorization in IoT environments make it necessary to look for alternative solutions. In most of today's Internet of Things, environments consist of a massive amount of devices with high computation power available. The idea is to use this often unused and therefore available computational power for a distributed access control instead of centralized access control.

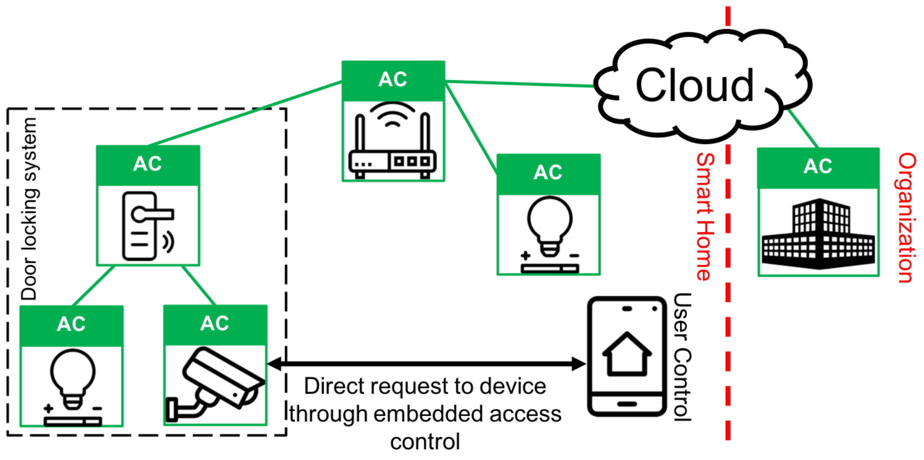


Fig. 2. An exemplary representation of a distributed access control in the application domain of a smart home that allows direct communication from device to device as desired in IoT.

Fig. 2 shows an example of distributed access control in the application domain of a smart home. The devices in the figure are arranged hierarchically and access control logic and execution are located at the device, pictured by green frames with the inscription "AC" around the devices. In addition to the smart home environment, the figure also shows the connectivity to an organization connected by the cloud. The figure shows the enabling of device to device communication through embedded access control.

Instead of a central access control instance, access control is distributed to IoT devices. The responsibility of access control is placed as close as possible to the devices. The closeness of the access control to the device enables high computational IoT devices that can self-evaluate the access rules that are specified for the device.

Relocating of access control closer to the devices enables the possibility of direct communication between devices and prevents a single point of failure. It also reduces the transfer of information which is needed to evaluate access rules because the information is usually presented directly on the side of the resource at the time of access control. Thus, this approach is not only reducing the data transfer but also improves the performance. Not only because the information itself is close to the resource, but also due to the distribution only those requests are checked by an access control component which are required to reach the corresponding resource.

The hierarchical structure as shown in Fig. 2 also allows inheriting policies which are determined for devices which are higher in the hierarchy to devices which are lower in the hierarchy. For example, by setting up a rule to prevent access to a particular device for a specific person within a smart home environment. The rule can, in this case, be inherited according to the device below in the hierarchy. Another example for inheritance is a compound of different devices, for example, a door locking system with three components: the smart door lock, the door light, and a door security camera. In most cases, it will be useful, that access rules are applied to all of the components instead of only a single component. Thus, a user who has general access to the door locking system can also access the door light and the door security camera. Such an access control hierarchy is therefore not restricted to the local IoT environment. It is also possible to inherit access control rules from organizations through the cloud. For example, if a maintenance worker of the door locking system wants to carry out maintenance. In this case, a global rule can be set on the side of the maintenance organization that is inherited through the cloud to the appropriate devices.

If the self-evaluation of access rules is not possible, because a device has not enough computational power, outsourcing the self-evaluation to devices on a higher level within the hierarchy is possible. This has the advantage that the evaluation is still close to the device instead of a centralized instance within the IoT environment or on a centralized instance on the cloud with worse runtime performance.

According to the problem statement and solution, there are various research objectives to work on in the future.

Research question 1: One objective is the effective distribution of access control components to the devices. Such access control components can be for example access control rules, the access control engine or just parts of the access control engine. In Internet of Things, there are many devices with different requirements. Taking account of these heterogeneous requirements is essential. It will not be possible for all devices to self-evaluate complex access decisions. However, other devices, in turn, have enough computing power to evaluate such complex policies. The efficient distribution of policies across network boundaries plays a significant role. For example, the definition of policies at an external access control provider in the cloud and the use of these global defined policies on devices within the smart home environment. Just as necessary as the distribution of rules is the retrieval of the required rules to evaluate the right policies for the right user at the right time. So a discovery mechanism to find the decentralized policies is required. Distribution and corresponding mechanisms also cause new problems and questions. For example, to keep track of the distribution of rules to ensure their management.

Research question 2: Another research objective is the consideration of existing mechanisms for identity and authorization. In the centralized world, there are massive amounts of such mechanisms available. For example, OAuth [9], User-Managed Access [8] or Attribute-based access control. In this context, it should be considered which mechanisms are relevant and how they need to be extended. Particular focus should be put on attribute-based access control and RestACL. RestACL [12] is an efficient access control language for RESTful services. The language follows the ideas of attribute-based access control and utilizes the concepts of REST to enable quick identification of policies that have to be evaluated to find an access decision.

Research question 3: Additionally to the mentioned research objectives, another research objective is the identification of criteria for satisfactory performance for distribution and discovery of policies. In the context of RestACL is the set of possible evaluable rules limited by the size of device RAM. To persist evaluable rules that exceed the size of device RAM it is required to use a database. Especially in this case is a satisfactory performance necessary to enable efficient access to persisted access rules.

5 Related Work

Recently, there have been various approaches to access control with different objectives to address the problems of centralization and decentralization.

The work described in "Outsourcing Access Control for a Dynamic Access Configuration of IoT Services" [15] explains an approach to authorizing resources within distributed systems. Furthermore, it describes how current solutions, developed for conventional devices like OAuth can be simplified and applied to the individual requirements for IoT devices.

Another objective to address problems of centralization in an IoT environment explained in "Distributed Capability-based Access Control for the Internet of Things" [10]. The work describes a distributed scenario with end-to-end access control validation. The presented solution allows the deployment of such scenarios like the mentioned without the intervention of any intermediate entity.

IoT using Internet protocols and paradigms, for example REST, to enable collective monitoring and executing tasks in an environment to interact with the physical world [6,2]. The work "RestACL: An Access Control Language for RESTful Services" [12] is, therefore, a promising approach. The paper describes with RestACL an access control language for RESTful services based on ABAC that enables quick identification of policies that have to be evaluated to find an access decision.

6 Conclusion

In this paper, we presented the disadvantages of traditional centralized access control in the context of IoT and proposed a solution for a distributed access control. The solution outlines the enabling of the direct device to device communication as in IoT desired by using unused computational power on IoT devices. The exemplary scenario in this work demonstrates how we will live, learn and work in future underuse of IoT as a backbone technology. The answering of the research questions, defined in this paper to evaluate the outlined approach, will be the central part in the doctoral thesis of Tobias Straub.

References

1. Bello, O., Zeadally, S.: Communication Issues in the Internet of Things (IoT). In: Next-Generation Wireless Technologies, pp. 189–219. Springer, London (2013), http://link.springer.com/10.1007/978-1-4471-5164-7_10
2. Bello, O., Zeadally, S.: Intelligent Device-to-Device Communication in the Internet of Things. IEEE Systems Journal **10**(3), 1172–1182 (2016), http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6725683
3. Benantar, M.: Discretionary-Access Control and the Access-Matrix Model. In: Access Control Systems, pp. 147–167. Kluwer Academic Publishers, Boston (2006), http://link.springer.com/10.1007/0-387-27716-1_5
4. Benantar, M.: Mandatory-Access-Control Model. In: Access Control Systems, pp. 129–146. Kluwer Academic Publishers, Boston (2006), http://link.springer.com/10.1007/0-387-27716-1_4
5. Benantar, M.: Role-Based Access Control. In: Access Control Systems, pp. 190–251. Kluwer Academic Publishers, Boston (2006), http://link.springer.com/10.1007/0-387-27716-1_8
6. Gubbi, J., Buyya, R., Marusic, S., Palaniswami, M.: Internet of Things (IoT): A vision, architectural elements, and future directions. Future Generation Computer Systems **29**(7), 1645–1660 (2013), https://www.sciencedirect.com/science/article/pii/S0167739X13000241
7. Gubbi, J., Buyya, R., Marusic, S., Palaniswami, M.: Internet of Things (IoT): A vision, architectural elements, and future directions. Future Generation Computer Systems **29**(7), 1645–1660 (2013), https://www.sciencedirect.com/science/article/pii/S0167739X13000241
8. Hardjono, T., Maler, E., Machulak, M., Catalano, D.: User-Managed Access (UMA) Profile of OAuth 2.0 (2015), https://docs.kantarainitiative.org/uma/rec-uma-core.html
9. Hardt, D.: The OAuth 2.0 Authorization Framework (2012), https://tools.ietf.org/html/rfc6749
10. Hernández-Ramos, J.L., Jara, A.J., Marín, L., Skarmeta, A.F.: Distributed Capability-based Access Control for the Internet of Things. In: 5th International Workshop on Managing Insider Security Threats (MIST 2013) (2013), https://pdfs.semanticscholar.org/972f/d21ba4a8049d43e370305ae0e2fc3b378e55.pdf
11. Hu, V.C., Kuhn, D.R., Ferraiolo, D.F.: Attribute-Based Access Control. Computer **48**(2), 85–88 (2015), http://ieeexplore.ieee.org/document/7042715/

12. Hüffmeyer, M., Schreier, U.: RestACL: An Access Control Language for RESTful Services. In: Proceedings of the 2016 ACM International Workshop on Attribute Based Access Control - ABAC '16. pp. 58–67. ACM Press, New York, New York, USA (2016), http://dl.acm.org/citation.cfm?doid=2875491.2875494
13. IHS: Internet of Things (IoT) connected devices installed base worldwide from 2015 to 2025 (in billions). Tech. rep. (2016), https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/
14. Miorandi, D., Sicari, S., De Pellegrini, F., Chlamtac, I.: Internet of things: Vision, applications and research challenges. Ad Hoc Networks **10**(7), 1497–1516 (2012), https://www.sciencedirect.com/science/article/pii/S1570870512000674
15. Montesano, P., Hueffmeyer, M., Schreier, U.: Outsourcing Access Control for a Dynamic Access Configuration of IoT Services. In: Proceedings of the 2nd International Conference on Internet of Things, Big Data and Security - Volume 1: IoTBDS. pp. 59–69 (2018), http://www.scitepress.org/DigitalLibrary/PublicationsDetail.aspx?ID=tLpMh45IM8Y=&t=1
16. Williams, Z.D.: IoT Platform Company List 2017 (2017), https://iot-analytics.com/iot-platforms-company-list-2017-update/

Towards Deployable Research Object Archives Based on TOSCA

Michael Zimmermann[1], Uwe Breitenbücher[1], Jasmin Guth[1], Sibylle Hermann[2], Frank Leymann[1], and Karoline Saatkamp[1]

[1] Institute of Architecture of Application Systems, University of Stuttgart
Universitätsstraße 38, 70569 Stuttgart, Germany
{firstname.lastname}@iaas.uni-stuttgart.de
[2] University Library of Stuttgart
Holzgartenstraße 16, 70174 Stuttgart, Germany
sibylle.hermann@ub.uni-stuttgart.de

Abstract. In science, reproducibility means that a scientific experiment can be repeated by another scientist with the same result. This is of particular importance to verify the results as well as to show the usefulness and reusability for further research. However, the exclusive publication of the research results in a scientific journal is usually not sufficient. In addition to research results, also research data as well as research software need to be published and made public available in order to enable researcher to gain new insights and thus advance research. However, the reproducibility and reusability of research data and research software typically is hindered by several barriers. Therefore, this work intends to first provide an overview of the current situation and issues in this particular topic and furthermore sketch our vision of standards-based Research Object Archives containing scientific publications, software, data, metadata and licenses in order to tackle the existing problems.

Keywords: Research Object, Reusability, Reproducibility, Deployment Model, TOSCA.

1 Introduction and Background

New findings in all research areas are based on the fact that existing knowledge and results of scientific studies and experiments can be shared, verified, and reused. This is of particular importance to verify the results, and thus verify the statements of the experiment as well as to show the usefulness and reusability for further research. Typically, only the scientific findings are published in scientific journals. The rapidly growing collective knowledge is dependent on software processing the raw data, which is an essential part of all scientific work today. Therefore, in order to understand and reproduce research results, not only the publication of the results and the research data, but also the associated research software must be made publicly available [2,28]. However, research software does not only have to be published, but must also be easily executable in order to enable researcher to reuse the software or reproduce the research results. But the heterogeneity in used infrastructures and technologies as well as other different technical requirements for the provisioning and operation of scientific applications requires expert IT knowledge about, for example, deployment technologies in order to be able to use the developed research software. Thus, some standards-based approach to automate the provisioning, the management, as well as the execution of research software and scientific experiments is required.

Regarding only the pure management of research software and research data, there is already some work available. That not only the research results and publications, but also related research software and research data is required in order to make research software usable and research results comprehensible has already been recognized and discussed by different authors [2,4,12,28]. Moreover, in various research domains, there are already some first isolated solutions, which are mainly concerned with the provisioning of research software and the connection with the generated data, for example, in the area of high-performance computing [26] or neuroscience [20]. There are also first approaches to model and store research results together with algorithms, data, methods, workflows and metadata [3–5,15]. Hunter [15] proposes so-called *Scientific Publication Packages*, which focus on the description of artifacts and the relationships between them. Bechhofer et al. [3,4] and Belhajjame et al. [5] define so-called *Research Objects*. Both approaches focus primarily on linking data with associated artifacts, such as algorithms or used methods, as well as the representation of the experiment workflow. Research Objects also have a defined life-cycle and are versionable. This way, different states, such as the progress of a study, can be directly mapped within the research object and changes to the object can be traced. However, both approaches do not address the self-contained packaging and automated provision of research software locally or in a cloud environment, the citability of research software and data, or the integration of license checks of the source code. With the constantly increasing importance, not only of research data and research software, but also the linking with the publications, new challenges for researcher arise: There is uncertainty regarding the licensing of software, there are no clear guidelines on how research software can be described and thus, made discoverable by metadata, and there are no procedures for the automated installation of research software to make it usable [1]. In addition, mechanisms are needed to ensure the sustainable storage and retrievability, the identifiability and thus the citability of different versions of research data, software or required components. Even though, the concepts of Scientific Publication Packages and Research Objects show the first approaches for dealing with research results, research data, and research software, a comprehensive and standards-based solution not only for storage, but also for finding, licensing, citation and provisioning of any research software has not yet been developed. Moreover, the aspects of automatically provisioning as well as managing applications is missing in these concepts. Thus, we want to use the findings from these concepts and further develop a standards-based packaging format enabling the automated provisioning and management of applications for the requirements of research software.

Regarding the automated deployment of cloud applications, in recent years, several technologies and standards were developed. This includes configuration management technologies such as Chef[1], Ansible[2], or Puppet[3], as well as container technologies such as Docker[4]. However, configuration management technologies and container technologies are based on specific artifacts, for example, installation scripts required to deploy the application. Furthermore, these artifacts and their formats differ greatly dependent of the used technologies [11]. Therefore, when multiple technologies need to be combined to deploy non-trivial applications, broad technical knowledge of the technologies and an integration process are required. But, as mentioned before, since information technology is being used in more and more scientific areas and not only in computer science, this technical knowledge cannot be assumed in order to run research software. Besides these technologies, there are also standards such as the Topology and Orchestration Specification for Cloud Applications (TOSCA) [8,22,23]. TOSCA is an OASIS standard that enables the definition of application deployments by topology models and management plans, which can be executed automatically by a TOSCA runtime, like e.g., the OpenTOSCA container [7]. A topology model describes the application components and their relations to each other. This includes application-specific components, such as PHP applications or databases, middleware, and infrastructure components, such as web servers or virtual machines. Thereby, application deployments can be described in a vendor-independent and portable manner. Thus, we want to use the TOSCA concepts in order to realize the provisioning and managing aspect of our Research Object Archives approach.

To sum up, there is no comprehensive approach available yet enabling the packaging of all required research artifacts as well as the corresponding publication, that is also supporting the automated provisioning and managing of the research software. Therefore, in this paper, we want to introduce our vision of a standards-based approach for packaging all scientific artifacts, such as research results, research data, research software as well as scientific publications together into one executable archive, thus, enabling the automated provisioning and management of scientific applications. To achieve that, we try to combine the packaging concepts of Research Objects with the deployment and managing concepts of the TOSCA standard in order to create automatically deployable Research Object Archives.

The remainder of this paper is structured as follows: We discuss and present the fundamentals, such as Research Objects and the OASIS standard Topology Orchestration and Specification for Cloud Applications (TOSCA) in detail in Section 2. We sketch our idea of Research Object Archives in Sect. 3 and discuss how it addresses the identified issues and how it enables the portability by an approach enabling the automated deployment and managing. In Sect. 4, works related to our approach are discussed. Finally, Sect. 5 concludes this paper.

[1] https://www.chef.io/chef
[2] https://www.ansible.com
[3] https://puppet.com
[4] https://www.docker.com

2 Basic Concepts

The idea behind Research Object Archives (ROARs) is to give the possibility to publish automatically deployable research software. For this purpose, the concept of Research Objects (ROs) will be implemented with the Topology Orchestration and Specification for Cloud Applications (TOSCA) standard. This section will introduce the basic concepts of ROs and TOSCA.

2.1 Research Objects

In general, reuse of a specific set of research results requires additional information. To meet this requirement, Bechhhofer et al. developed the concept of Research Objects. [3,4] They define the following set of principles to publish research results together with algorithms, data, methods, workflows, and metadata:

Reusable: ROs should be usable as a whole or in parts.

Repurposeable: The relationship of the contained parts should be described.

Repeatable: There should be enough information to repeat the research. Also important are sufficient privileges to access the data.

Reproducible: There should be enough information to validate the result.

Replayable: There should be enough information to understand what happens in a specific process.

Referenceable: A RO need a unique Identifier to get credit over citations for the research output.

To implement these principles, Bechhofer et al. propose the following features for ROs. References and resources are *Aggregated* in an RO. Such aggregation has to organize the content to resolve the resources dynamically. The ROs as object and the content of the RO have a separate *Identity*. Thus, the entire research object or parts of it can be unambiguously referenced. *Metadata* has two functions: To discover the RO and to describe the research for reuse. The RO as a research result must be described by machine-readable metadata to be found. Additionally, the content requires information about licensing, attribution, copyright or descriptions of provenance and the derivation of results. As research is a process with a *Lifecycle* that should be described. Different events will happen in a particular sequence and different actions can be performed at various stages. These changes over time, particularly adjustments like deleting or adding content to a RO, must be recorded with *Versioning*. The *Management* of ROs needs the possibility to operations like Creation, Retrieval, Update, and Deletion (CRUD). *Security* issues like access, authentication, ownership, and trust need to be addressed. The last feature Bechhofer et al. propose is the *Graceful Degradation of Understanding* to give the opportunity to use the Research object without understanding the whole research process.

The approach of Research Objects do not address a specific implementation of ROs. Neither the self-descriptive packaging and automated provision of research software locally or in a cloud nor the question of how to cite the research software and data or the integration of license checks of the source code is addressed by Research Objects.

2.2 Topology Orchestration and Specification for Cloud Applications

Since our approach is based on the TOSCA standard, we introduce the OASIS standard Topology Orchestration and Specification for Cloud Applications (TOSCA) in this subsection, in order to provide a comprehensive background. TOSCA enables to describe the automated deployment and management of applications in an interoperable and portable manner. We only give an overview of the fundamental TOSCA concepts, detailed information can be found in the TOSCA Specifications [23,24], the TOSCA Primer [24] and in Binz et al. [8]. A comparison of TOSCA with other Cloud Modeling languages can be found in Bergmayr et al. [6].

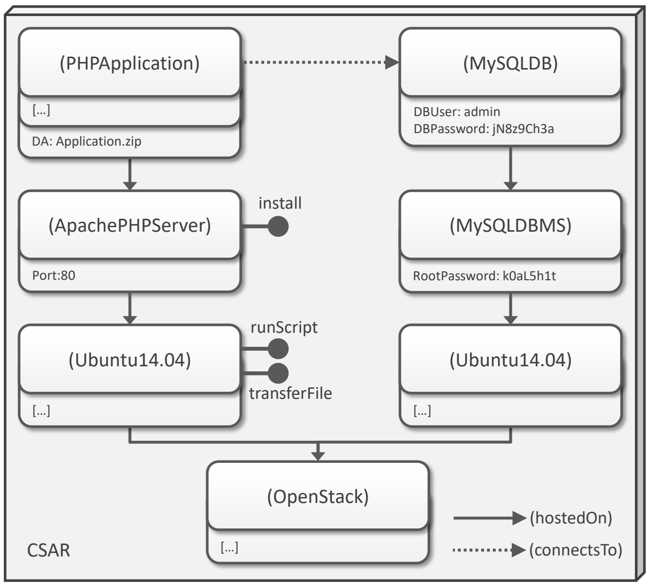


Fig. 1. Exemplary TOSCA Topology Template.

The TOSCA standards enables to describe required infrastructure resources, software components, as well as the structure of cloud or IoT applications. Furthermore, TOSCA enables to define the required operations for managing such applications. Thus, TOSCA enables the automated deployment and management of cloud and IoT applications. The structure of cloud applications are defined by so-called TOSCA *Topology Templates*. An exemplary TOSCA Topology Template, following the visual notation Vino4TOSCA [10] is illustrated in Fig. 1. Technically, a Topology Template is a graph consisting of nodes and directed edges. The nodes represent components of the application, for instance, a virtual machine or a database and are called *Node Templates*. The edges connecting the nodes specify the relations between Node Templates, for example, "hostedOn" or "connectsTo" and are called *Relationship Templates*. For reusability purposes, the TOSCA standard enables the specification of *Node Types* and *Relationship Types* defining the semantics of the Node Templates and Relationship Templates. Node

Types, for example, enable to define *Properties* as well as *Management Operations*. Properties are for example passwords, usernames, or the port of a web server and thus, enable the customization of the TOSCA Topology Templates. Management operations enable the management of the modeled components. For example, typically a software component node provides an “install” operation in order to install the component or a hypervisor node provides a “createVM” and “terminateVM” operation in order to create and terminate virtual machines.

Management operations are implemented by so-called *Implementation Artifacts (IAs)*. IAs can be implemented using any various technologies, e.g., as a WAR-file providing a WSDL-based SOAP Web Service, a configuration management technology, such as Ansible, Puppet, Chef, or just as a simple shell script. Besides IAs, TOSCA also defines so-called *Deployment Artifacts (DAs)* representing the artifacts implementing the business logic of the components of an application. For example, the DA of a Java application can be a WAR-file. In case of a PHP application, a DA would be a ZIP file containing all PHP files, images, and other required files. The creation and termination of instances of a modeled application can either be done *declaratively*, by deriving the actions that need to be executed directly from the Topology Template or *imperatively* with help of *Management Plans* [14]. A Management Plan is an executable workflow model specifying all tasks as well as the order in which these tasks need to be executed to achieve a certain management functionality, e.g., the provisioning of a new application instance or to scale out a component of a running application instance. TOSCA does not specify how such plans should be implemented, however, recommends using a workflow language such as the *Business Process Execution Language (BPEL)* [21] or the *Business Process Model and Notation (BPMN)* [25]. Furthermore, there is a BPMN extension called *BPMN4TOSCA* that is explicitly tailored for describing TOSCA-based management plans [17,19].

Additionally, the TOSCA standard also specifies a portable and self-contained packaging format, so-called *Cloud Service Archive (CSAR)*. CSARs enable to package Topology Templates, type definitions, Management Plans, IAs, DAs, and all other required files for automating the provisioning as well as the management of applications into one archive. Through the standardized meta-model and packaging format, these CSARs can be automatically processed and executed by standard-compliant TOSCA Runtime Environments, such as the OpenTOSCA Ecosystem[5]. Therefore, portability as well as interoperability can be ensured.

Overall, the OASIS standard TOSCA enables to automate the provisioning and management of complex applications as well as to specify and package all descriptions and required files in a portable format. Therefore, the standard provides a suitable basis for packaging and managing research software together with associated artifacts for an automated provisioning. Up to now, the identification via metadata as well as the linking with licenses, external data sources, and publication repositories is not considered in the standard. However, due to the extensibility of the specification, this standard can be extended accordingly.

[5] https://github.com/OpenTOSCA

3 Towards an Approach for the Automated Deployment of Research Objects

In this section, we introduce our approach of portable Research Object Archives (ROARs). Therefore, we first present the main concepts of ROARs and subsequently illustrate how they are supposed to be created and used. We further show, how they enable the automated deployment of the modeled application and how the proposed features presented in Sect. 2.1 are realized by ROARs.

3.1 Research Object Archive

In order to enable developing, packaging, publishing, and deploying research software efficiently, a self-contained and portable packaging format for research software is inevitable. Thus, the main goal objectives are to develop a packaging format that enables bundling all important information of research software, especially its technical dependencies, associated research data, descriptive metadata, used licenses, and a reference to the corresponding publication, as well as to enable the automated provisioning. The conceptual structure of a ROAR is depicted in Fig. 2. The ROAR format is based on Research Objects (ROs) (cf. Sect. 2.1) and the provisioning and management concepts of the TOSCA standard (cf. Sect. 2.2), which is explained in more detail in the following.

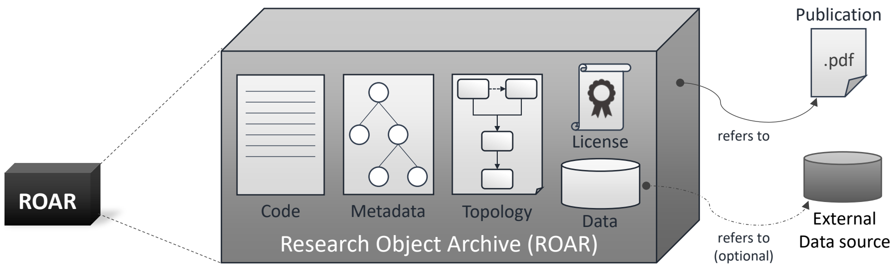
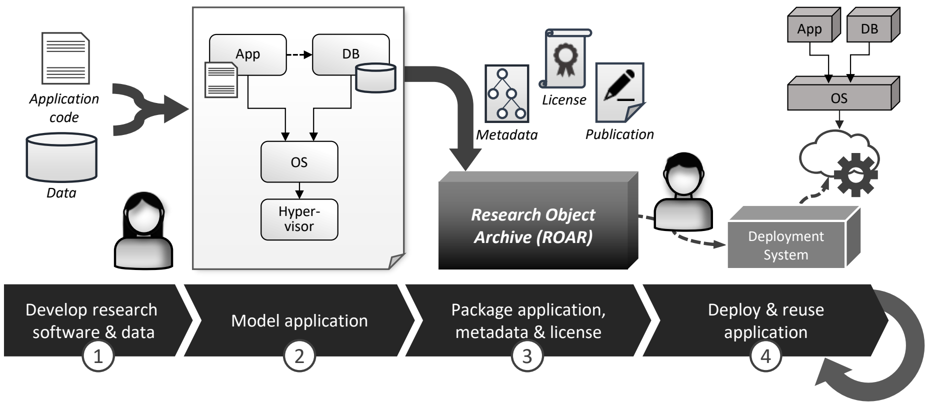


Fig. 2. Structure of a Research Object Archive (ROAR).

For ensuring the linking to the corresponding publication, a ROAR, for example in the case of an open access publication, should refer directly to the publication – typically a PDF file – and in case of house publications to the landing page of the article at the respective publisher. Furthermore, a ROAR should contain the used source code, descriptive metadata, license information, and the used database, which can also be located in an external repository. By supporting metadata, the ROAR can be found easily and thus, supporting the proposed feature *Metadata* in Sect. 2.1. Since database can be located in external repositories, ROARs need to support referencing external data sources and resolving them dynamically (cf. *Aggregation* in Sect. 2.1). Moreover, because data or software can change, ROARs need to support versioning (cf. *Versioning* in Sect. 2.1). Based on the metadata, a unique ID can be assigned to the ROAR, thus enabling citation (cf. *Identity* in Sect. 2.1). Of course, ROARs should be created, updated, and retrieved by researcher as well as stored in repositories or archives, thus supporting the proposed feature *Management* in Sect. 2.1. Furthermore, by not only adding the research software, dependencies, as well as required data, but also a TOSCA-based description how the application can be provisioned, a ROAR should enable the automated provisioning of the modeled and archived application and thus, simplify the repeatability of the research results (cf. *Graceful Degradation of Understanding* in Sect. 2.1). However, since not all necessary functionalities, such as referencing publications in a CSAR, are currently supported by the TOSCA standard, extensions of TOSCA and the CSAR packaging format must be developed and combined with the RO concept.

3.2 Research Object Portability and Deployment Approach

The reusability as well as the reproducibility are vital factors in science. Thus, ROARs need to be designed in a way supporting (i) the researcher creating such a ROAR as well as (ii) the researcher that wants to reuse a modeled application. In this section, we illustrate how ROARs are created and how they can be reused.



Application code
Data
App
DB
OS
Hyper-visor
Metadata
License
Publication
Research Object Archive (ROAR)
App
DB
OS
Deployment System
Develop research software & data
1
Model application
2
Package application, metadata & license
3
Deploy & reuse application
4

Fig. 3. Overview: Usage of Deployable Research Object Archives.

Fig. 3 shows the conceptual approach, the roles involved, and the general procedure for storing, testing, finding, and reusing research software as well as the associated data. When scientists want to publish their developed research software with associated data (see step 1), they first have to model the application's topology with all its dependencies as well as required data (see step 2) using the TOSCA modeling concepts (cf. Sect. 2.2). In our approach, it should also be possible to insert the data directly to the final ROAR as well as to only insert a reference to an external location where the data is stored. Thus, the topology model has to support both variants to define required data. Since, our concept is based on the TOSCA standard, of course available modeling tools supporting the TOSCA standard can be used for modeling the application, for example, the open-source tool *Winery*[6] [18]. After the modeling, the research software can be packaged with associated data, licenses, descriptive metadata, as well as associated publications, published, and persistently stored as a ROAR (see step 3). The selection of a suitable license for the research software can be supported by using license verification tools, such as Fossology[7] or Black Duck[8], which can automatically detect possible license violations regarding used libraries. The metadata added to the ROAR is used to describe and identify the research software and any data and publications it contains. Using this metadata, the ROAR can be found by other scientist in order to reuse the contained software or data (see step 4). Furthermore, a unique ID can be assigned to a ROAR for enabling the citation of it. Since a ROAR is a standards-based packaging format containing all the necessary components for enabling the automated provisioning (cf. Sect. 2.2), the modeled application can be automatically deployed using a standard-compliant deployment engine, such as the open-source runtime environment *OpenTOSCA Container*[9] [7]. Therefore, scientists do not require any expert IT knowledge at all and are able to use and recreate the research results as often as they want to.

4 Related Work

Packaging software, data, and publications together into one archive as well as enabling the provisioning of software have been in the focus of different research areas. Therefore, in this section, we complete our discussion about related work, which we already discussed partially in Sect. 1.

Weigel et al. [29] give a recommendation for *actionable collections* and a technical interface specification to enable client-server interaction. Their focus lies on the management of research collections. To enable and automate the provisioning of software in different environments, concepts of [16] and [30] have already been developed. Képes et al. [16] presents a template-based concept for provisioning software using a template that is suitable for a specific infrastructure. Zimmermann et al. [30] uses the infrastructure or database components available in an environment to connect the software and generate an overall model for provisioning. These concepts can be used to provision research software in the available environment and to connect it to existing data sources but did not cover the integrated publishing and description of software, data, workflow and licensing of the research results. Stodden et al. [27] provides a platform to store source code and data and run them directly in a cloud environment but did not provide licence checks and an integrated publishing format. For general storage and easy software execution, Boettiger [9] proposes Docker

[6] https://projects.eclipse.org/projects/soa.winery
[7] https://www.fossology.org/
[8] https://www.blackducksoftware.com/
[9] https://github.com/OpenTOSCA/container

containers to virtualize the exact system environment in which the original results were produced. The Software Heritage Archive [13] aims to collect all source code that is publicly available. The development history will be archived with an index in order to make the code referenceable and accessible. This approach is an important archive for software source code, however, is not intended to package all artifacts together in order to enable the execution of the software for reuse.

5 Conclusion

In this paper, we illustrated our approach of a comprehensive solution for packaging research software together with data, metadata, license information, and publications in a portable way, which enables reusability as well as reproducibility in science. Therefore, in this work we first presented an overview of the current issues and problems in this area. Furthermore, we presented the concepts of Research Objects and the TOSCA standard and sketched how both concepts can be combined in order to create our approach of Research Object Archives. Moreover, we depicted the composition and the ingredients of a ROAR and characterized the features it provides. We also illustrated how the portability of a ROAR is achieved and how the automated deployment is realized. In future work, we focus on the implementation and integration of the proposed concepts into the TOSCA modeling tool Winery and OpenTOSCA Container.

Acknowledgments This work was partially funded by the projects *SePiA.Pro* (01MD16013F) and *IC4F* (01MA17008G).

References

1. Almeida, D.A., Murphy, G.C., Wilson, G., Hoye, M.: Do Software Developers Understand Open Source Licenses? In: ICPC. pp. 1–11. IEEE (2017)
2. Barnes, N.: Publish your computer code: it is good enough. Nature News 467(7317), 753–753 (2010)
3. Bechhofer, S., Buchan, I., De Roure, D., Missier, P., Ainsworth, J., Bhagat, J., Couch, P., Cruickshank, D., Delderfield, M., Dunlop, I., Gamble, M., Michaelides, D., Owen, S., Newman, D., Sufi, S., Goble, C.: Why linked data is not enough for scientists. Future Generation Computer Systems 29(2), 599–611 (2013)
4. Bechhofer, S., De Roure, D., Gamble, M., Goble, C., Buchan, I.: Research Objects: Towards Exchange and Reuse of Digital Knowledge. In: FWCS. Nature Precedings (2010)
5. Belhajjame, K., Zhao, J., Garijo, D., Gamble, M., Hettne, K., Palma, R., Mina, E., Corcho, O., Gómez-Pérez, J.M., Bechhofer, S., Klyne, G., Goble, C.: Using a suite of ontologies for preserving workflow-centric research objects. Web Semantics: Science, Services and Agents on the World Wide Web 32, 16–42 (2015)
6. Bergmayr, A., Breitenbücher, U., Ferry, N., Rossini, A., Solberg, A., Wimmer, M., Kappel, G., Leymann, F.: A Systematic Review of Cloud Modeling Languages. ACM Computing Surveys (CSUR) 51(1), 22:1–22:38 (2018)

7. Binz, T., Breitenbücher, U., Haupt, F., Kopp, O., Leymann, F., Nowak, A., Wagner, S.: OpenTOSCA - A Runtime for TOSCA-based Cloud Applications. In: ICSOC. pp. 692–695. Springer (2013)
8. Binz, T., Breitenbücher, U., Kopp, O., Leymann, F.: TOSCA: Portable Automated Deployment and Management of Cloud Applications, pp. 527–549. Advanced Web Services, Springer (2014)
9. Boettiger, C.: An introduction to docker for reproducible research. ACM SIGOPS Operating Systems Review 49(1), 71–79 (2015)
10. Breitenbücher, U., Binz, T., Kopp, O., Leymann, F.: Vinothek - A Self-Service Portal for TOSCA. In: ZEUS. pp. 69–72. CEUR-WS.org (2014)
11. Breitenbücher, U., Binz, T., Kopp, O., Leymann, F., Wettinger, J.: Integrated Cloud Application Provisioning: Interconnecting Service-Centric and Script-Centric Management Technologies. In: CoopIS. pp. 130–148. Springer (2013)
12. Collberg, C., Proebsting, T., Warren, A.M.: Repeatability and benefaction in computer systems research. University of Arizona TR 14-4 (2015)
13. Cosmo, R.D., Zacchiroli, S.: Software Heritage: Why and How to Preserve Software Source Code. In: iPRES (2017)
14. Endres, C., Breitenbücher, U., Falkenthal, M., Kopp, O., Leymann, F., Wettinger, J.: Declarative vs. Imperative: Two Modeling Patterns for the Automated Deployment of Applications. In: Proceedings of the 9th International Conference on Pervasive Patterns and Applications (PATTERNS). pp. 22–27. Xpert Publishing Services (2017)
15. Hunter, J.: Scientific publication packages–A selective approach to the communication and archival of scientific output. International Journal of Digital Curation 1(1), 33–52 (2008)
16. Képes, K., Breitenbücher, U., Leymann, F.: The SePaDe System: Packaging Entire XaaS Layers for Automatically Deploying and Managing Applications. In: CLOSER. pp. 626–635. SciTePress (2017)
17. Kopp, O., Binz, T., Breitenbücher, U., Leymann, F.: BPMN4TOSCA: A Domain-Specific Language to Model Management Plans for Composite Applications. In: Proceedings of the 4th International Workshop on the Business Process Model and Notation (BPMN 2012). pp. 38–52. Springer (2012)
18. Kopp, O., Binz, T., Breitenbücher, U., Leymann, F.: Winery – A Modeling Tool for TOSCA-based Cloud Applications. In: ICSOC. pp. 700–704. Springer (2013)
19. Kopp, O., Binz, T., Breitenbücher, U., Leymann, F., Michelbach, T.: A Domain-Specific Modeling Tool to Model Management Plans for Composite Applications. In: Proceedings of the 7th Central European Workshop on Services and their Composition, ZEUS 2015. pp. 51–54. CEUR Workshop Proceedings (2015)
20. Nyström, P., Falck-Ytter, T., Gredebäck, G.: The TimeStudio Project: An open source scientific workflow system for the behavioral and brain sciences. Behavior Research Methods 48(2), 542–552 (2016)
21. OASIS: Web Services Business Process Execution Language (WS-BPEL) Version 2.0. Organization for the Advancement of Structured Information Standards (OASIS) (2007)
22. OASIS: Topology and Orchestration Specification for Cloud Applications (TOSCA) Primer Version 1.0. OASIS (2013)
23. OASIS: Topology and Orchestration Specification for Cloud Applications (TOSCA) Version 1.0. OASIS (2013)
24. OASIS: TOSCA Simple Profile in YAML Version 1.0. OASIS (2015)
25. OMG: Business Process Model and Notation (BPMN) Version 2.0. Object Management Group (OMG) (2011)

26. Pizzi, G., Cepellotti, A., Sabatini, R., Marzari, N., Kozinsky, B.: Aiida: automated interactive infrastructure and database for computational science. Computational Materials Science 111, 218–230 (2016)
27. Stodden, V., Hurlin, C., Pérignon, C.: Runmycode.org: A novel dissemination and collaboration platform for executing published computational results. In: eScience. pp. 1–8. IEEE (2012)
28. Stodden, V., McNutt, M., Bailey, D.H., Deelman, E., Gil, Y., Hanson, B., Heroux, M.A., Ioannidis, J.P., Taufer, M.: Enhancing reproducibility for computational methods. Science 354(6317), 1240–1241 (2016)
29. Weigel, T., Almas, B., Baumgardt, F., Zastrow, T., Schwardmann, U., Hellström, M., Quinteros, J., Fleischer, D.: Recommendation on Research Data Collections. Research Data Alliance (2017)
30. Zimmermann, M., Breitenbücher, U., Falkenthal, M., Leymann, F., Saatkamp, K.: Standards-based Function Shipping – How to use TOSCA for Shipping and Executing Data Analytics Software in Remote Manufacturing Environments. In: EDOC. pp. 50–60. IEEE (2017)

Application Scenarios for Automated Problem Detection in TOSCA Topologies by Formalized Patterns

Karoline Saatkamp[1], Uwe Breitenbücher[1], Oliver Kopp[2], and Frank Leymann[1]

[1] Institute of Architecture of Application Systems, University of Stuttgart
[2] Institute for Parallel and Distributed Systems, University of Stuttgart
Universitätsstraße 38, 70569 Stuttgart, Germany
`[lastname]@informatik.uni-stuttgart.de`

Abstract. There are several reasons why application components are redistributed to multiple environments: parts of the IT-infrastructure are outsourced or an application has to be deployed to different customers with different infrastructures. For an automated deployment, several technologies and standards already exist. With the TOSCA standard the deployment of an application can be modeled as topology representing the application's components and their relations. When such a topology-based deployment model is redistributed, problems can arise that were not present before, such as firewalls that prevent direct access to a component. Problems can be detected based on problem- and context-formalized patterns. In this paper, we present application scenarios for the pattern formalization approach to detect problems in restructured topology-based deployment models based on selected cloud computing patterns.

Keywords: Cloud Computing Patterns, Formalization, Prolog, TOSCA.

1 Introduction and Background

Over the last years, several technologies for the deployment and management of cloud applications, such as Docker[1], Kubernetes[2], or Cloud Foundry[3], have been developed. Besides these vendor-specific technologies, standards, such as the OASIS standard TOSCA (Topology and Orchestration Specification for Cloud Applications) [24], are published to describe the deployment and management of cloud applications in a vendor-independent manner. TOSCA enables the description of topology-based deployment models specifying an application's structure by its components and their relations [12]. Such topology-based deployment models are declarative deployment models that can be interpreted by a runtime that infers the deployment logic from the structural description [12]. Imperative models, on the other hand, define a process that specifies the deployment logic explicitly [12].

[1] https://www.docker.com/
[2] https://kubernetes.io/
[3] https://www.cloudfoundry.org/

TOSCA topologies are topology-based deployment models, describing the application's components and their relationships. Components can be application-specific components such as a PHP WebApp as well as middleware components such as a Tomcat or infrastructure components such as an OpenStack. A component can be, for example, a *PHP WebApp* that is *hosted on* an *Apache Web Server*, as depicted in **Fig. 1**. The *hostedOn* or *connectsTo* relations describe the relationships between the components. Due to several reasons, the components of an application might have to be redistributed to different environments: parts of the IT are outsourced or an application has to be deployed for different customers with different environmental conditions. Based on the *Split and Match* method introduced in a previous work [27], each application-specific component can be annotated with a target label that specifies the intended target location of the component. According to these labels, the topology-based deployment model is split and matched with the available infrastructure or platform service in the target location and this results in a restructured deployment model. In the example depicted in **Fig. 1**, two components that are formerly hosted on the same virtual machine can be redistributed, for example, to an AWS EC2 and an OpenStack.

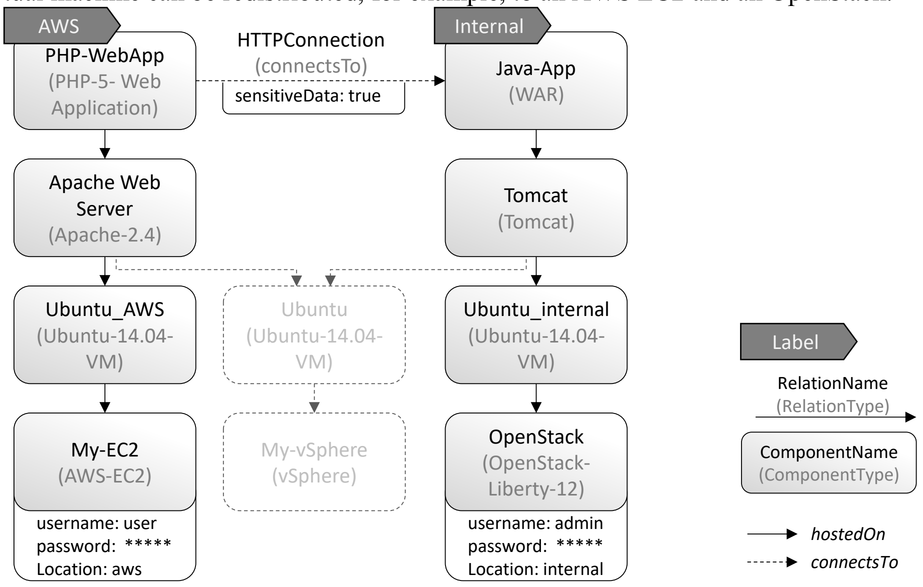


Fig. 1. Example of a restructured topology-based deployment model

In the initial topology in **Fig. 1** the *Apache Web Server* and the *Tomcat* are intended to be hosted on the same virtual machine *Ubuntu*, as shown by the dashed greyed out components. Based on the attached target labels *AWS* and *Internal,* the deployment model is split into two separate stacks. For this, the virtual machine is duplicated for each target location and the available infrastructure components from the target locations are selected and injected into the topology-based deployment model.

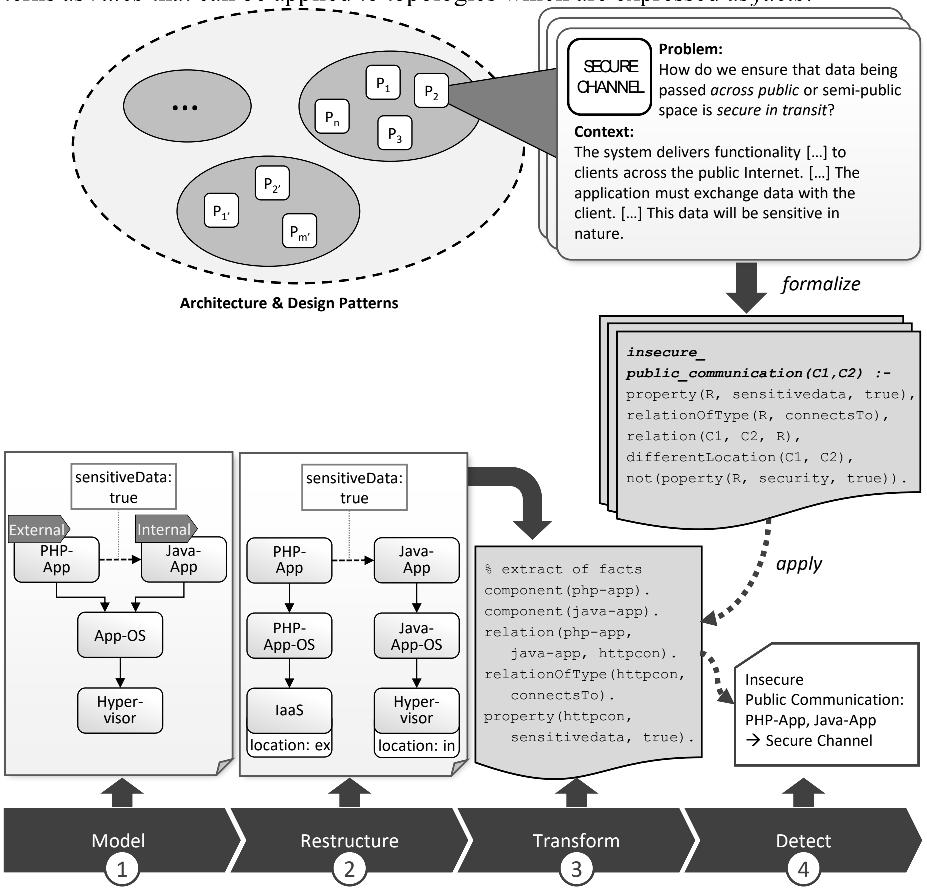
The restructuring of such deployment models can result in problems that have not existed before. In the example presented in **Fig. 1**, sensitive data that used to be exchanged within a single virtual machine is now exchanged over the internet, which can lead to security issues. Furthermore, communication restrictions or incompatibilities can occur. To detect such problems in an automated manner, we presented an approach to automatically detect problems in restructured deployment models based on formalizing architecture and design patterns [26]. This concept is based on existing design and architecture knowledge in the form of patterns that describe best-practice solutions for recurring problems in a certain context [1]. Such architecture and design patterns are discovered and described in several domains, for example, for general architecture solutions [9], application integration [17], security mechanisms [28], and for cloud computing [13]. Independent of the domain, a pattern describes the *problem* solved by this pattern, the *context* when the problem occurs, and the *solution* in a technology-independent manner. However, patterns are only captured as textual descriptions.

To enable an automated problem detection, Saatkamp et al. [26] presented a pattern formalization approach to formalize the problem and context description of a pattern. In previous work [26], the applicability of the formalization approach has been presented for two patterns: the *Secure Channel* pattern which is part of the security pattern language by Schumacher et al. [28] and the *Application Component Proxy* which is part of the cloud computing pattern language by Fehling et al. [13]. For a prototypical validation the TOSCA standard has been chosen, because it is a generic standard and independent from a certain technology or provider [4]. A TOSCA topology-based deployment model is described as *Topology Template*. The application's components are specified as *Node Templates* and their relations as *Relationship Templates*. The semantic of these elements is specified by their *Node Type* or *Relationship Type* respectively. In this paper, we extend the validation of the problem detection approach based on formalized patterns [26] by further patterns. The TOSCA-based prototype, presented in the previous paper is used as basis for the validation.

The remainder of this paper is structured as follows: Section 2 gives on overview of the problem detection approach while Section 3 describes the application scenarios. In Section 4 related work is discussed and Section 5 concludes this paper.

2 Problem Detection Approach Overview

We presented the approach to automatically detect problems in restructured deployment models based on formalizing architecture and design patterns a previous work [26]. This is based on the concept of applying architectural and design knowledge in terms of patterns to restructured topology-based deployment models. The textual description of such patterns is based on a pattern format. Even if pattern formats differ slightly between different pattern languages, the essential parts are the same [23,30]: (i) the *problem* section that describes the problem solved by the pattern, (ii) the *context* section that describes the context in which the problem arises, and (iii) the *solution* section that gives a technology-independent description of the best-practice solution. For the detection of problems occurring in restructured topology-based deployment models, the problem as well as context description are of main interest. However, for an automated approach each pattern has to be formalized in a machine-readable manner. Thus, the logic programming language Prolog is used to express the problem and context of patterns as *rules* that can be applied to topologies which are expressed as *facts*.



Problem:
How do we ensure that data being passed *across public* or semi-public space is *secure in transit*?

Context:
The system delivers functionality […] to clients across the public Internet. […] The application must exchange data with the client. […] This data will be sensitive in nature.

```
insecure_
public_communication(C1,C2) :-
property(R, sensitivedata, true),
relationOfType(R, connectsTo),
relation(C1, C2, R),
differentLocation(C1, C2),
not(poperty(R, security, true)).
```

```
% extract of facts
component(php-app).
component(java-app).
relation(php-app,
   java-app, httpcon).
relationOfType(httpcon,
   connectsTo).
property(httpcon,
   sensitivedata, true).
```

Insecure
Public Communication:
PHP-App, Java-App
→ Secure Channel

Fig. 2. Overview of the Problem Detection Approach Based on Formalized Patterns [26]

In **Fig. 2** an overview of the problem detection approach based on formalized patterns is depicted. At the top of the figure the different pattern languages are sketched. Each area represents a pattern language, for example, the enterprise integration patterns [17], the cloud computing patterns [13], or the security patterns [28] which also include the *Secure Channel* pattern. The Secure Channel pattern addresses the problem to ensure that data being passed across a public network is secure in transit. This textual description is formalized as a Prolog rule to enable an automated problem detection process. Even if the formalization has to be done manually, the resulting rule expressing the problem and context of the pattern can be reused for arbitrary topologies. The rule `insecure_public_communication(C1, C2)` queries a fact base for two components *C1* and *C2* that are connected by a relation R which has a key-value property *sensitivedata: true* attached. This relation R must be of type *connectsTo*. In addition, the two components C1 and C2 must have different locations, indicated by the key-value property *location*, and no security mechanisms are used for the connection between the components. This rule formalizes the problem and context of the Secure Channel pattern and can be applied to facts representing the structure of a topology.

The procedure to detect problems in a restructured topology-based deployment model, for example in a TOSCA topology, starts with modeling the application's structure (step 1). The topology is then annotated with target labels that indicate the desired distribution of the components to different target locations. In this example, the *PHP-App* shall be hosted in an external environment and the *Java-App* shall be located internally. In the restructuring step (step 2), the topology is split and matched using the Split and Match method by Saatkamp et al. [27]. In a third step, the graph-based description of the application's structure is transformed into Prolog facts. This can be automated based on transformation rules, as descripted in [26]. For each element in the topology a fact is created. An extract of the facts representing the exemplary topology is shown in **Fig. 2**. By applying all formalized patterns to the topology facts, problems can be detected in the restructured topology-based deployment model. As a result, the detected problems, the affected components, and the pattern addressing this problem are returned. For the transformation of TOSCA topologies to Prolog facts and the problem detection in such topologies the *Topology ProDec*[4] tool can be used [26].

3 Application Scenarios Based on Cloud Integration Patterns

The described approach in Section 2 can be applied to several pattern languages. In [26] the approach has been applied to the Secure Channel pattern from the security patterns [28] and to the Application Component Proxy from the cloud computing patterns [1]. In this paper, two additional application scenarios are presented based on two cloud integration patterns [1]: *Message Mover* and *Integration Provider*. In the following, the two patterns are described in more detail and the Prolog rules for formalizing these patterns are presented. They are also part of the patterns listed in the *Topology ProDec* tool and can be used for an automated problem detection in TOSCA topologies.

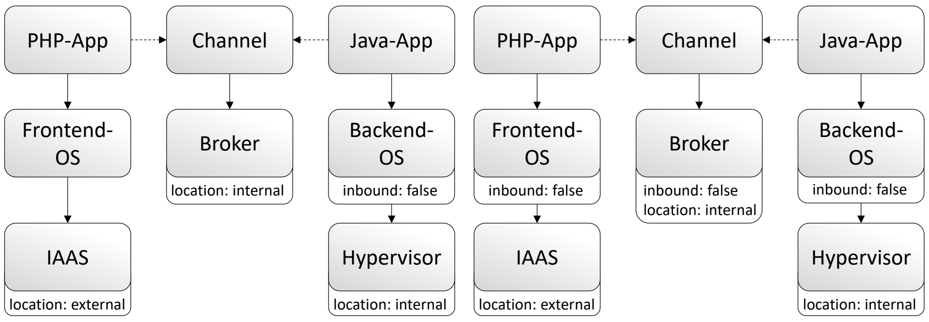


Fig. 3. Required Patterns: Message Mover (left) and Integration Provider (right)

[4] https://github.com/saatkamp/topology-prodec

3.1 Application Scenario: Message Mover

The Message Mover is a pattern of the cloud computing pattern language [1]. This pattern is just applicable to a message-based communication. Therefore, in **Fig. 3** on the left a topology with a message-based communication is depicted. In this example, the PHP-WebApp publishes data to a Queue and the Java-App receives data from it. After the redistribution the Java-App and the Broker are hosted in the internal datacenter and the PHP-WebApp is hosted in a public cloud, for example provided by Amazon. The Java-App is located in a restricted environment and thus, the access from outside the location is not permitted. To ensure the accessibility to a queue for each component, a queue shall be available in each location. The integration problem of these distributed queues can be solved by the *Message Mover*. However, the problem must be detected first. In the following the problem and context description of the Message Mover is presented [13]:

Problem:

How can message queues of different providers be integrated without an impact on the application component using them?

Context:

The application components comprising a distributed application (160) often exchange data using messaging. These messages are stored in message queues. [...] If these queues reside in different cloud environments that form a hybrid cloud (75) accessibility to queues of one environment may be restricted for application components that are deployed in another environment. [...] Therefore, each of the application components shall access a message queue hosted in the cloud environment where the application component itself is hosted. [...]

The pattern context is similar to the *Application Component Proxy*. The Application Component Proxy addresses the problem that a component directly accesses a component located in a restricted environment. Because direct access is not permitted to restricted environments an Application Component Proxy is required to access this component. More details on the formalization of this pattern can be found in [26]. However, in this case instead of a proxy an additional queue in the unrestricted environment and a message mover integrating the queues are required. The pattern aims to solve the accessibility of a more concrete system component, a queue. Based on the knowledge from messaging patterns, the problem description is based on the generic *Message Channel* in order to not exclude components, like e.g., topics which are publish-subscribe channels [17]. The resulting `distributed_messaging` rule for the formalized *Message Mover* is the following:

```
distributed_messaging(C1, C2):-
  messaging_communication(Channel, C1, C2),
  components_in_different_locations(C1, C2),
  hybrid_environment(C1, C2).
```

The shown facts serve as conditions for the pattern rule. Each of the facts in turn is a rule encapsulating a complex query (not shown for brevity, but can be found in the

Topology ProDec tool[5]). The first *condition* checks whether messaging is used in the topology because the pattern just relates to message-based systems. In addition, the problem and context description refers to distributed applications that are used in hybrid environments. Therefore, the second condition checks whether the communicating components are located in different environments and if they form a hybrid environment (third condition).

This rule can be applied to the deployment model depicted in **Fig. 3** on the left. Based on that, the problem is automatically detected and it can be solved according to the descripted solution for the *Message Mover* pattern.

3.2 Application Scenario: Integration Provider

The *Integration Provider* pattern is another cloud integration pattern from the cloud computing patterns [13]. In **Fig. 3** on the right a topology with two components (PHP-App and Java-App) communicate using messaging is shown. In contrast to the topology on the left, both components as well as the Channel are located in restricted environments. Thus, the components as well as the Channel are not accessible from outside the location. This problem can be solved by the Integration Provider pattern that describes the problem and its context as follows [13]:

Problem:

How can components in different environments be integrated through a 3rd-party provider?

Context:

When companies collaborate or one company has to integrate applications of different regional offices, different applications or the components of a distributed application are distributed among different hosting environments. Communication between these environments may be restricted. Especially, hosting environments may restrict any incoming communication initiated from the outside. Communication leaving the restricted environments is, however, often allowed. Therefore, additional integration components are required that have to be accessible from restricted environments. […]

From the above given description of this pattern, the difference to the `distributed_messaging` that formalizes the problem and context description of the Message Mover pattern can be seen: Instead of a hybrid environment consisting of a restricted and an unrestricted environment, these are two restricted environments that must be integrated to enable communication between the components. Besides that, the Integration Provider pattern is not limited to deployment models using messaging. In the following the `integration_of_restricted_environments` rule formalizing the problem and context of the Integration Provider pattern is shown:

[5] https://github.com/saatkamp/topology-prodec/blob/master/pattern_prologfiles/helper.pl

```
integration_of_restricted_environments(C1, C2):-
  components_in_different_locations(C1, C2),
  component_in_restricted_environment(C1),
  component_in_restricted_environment(C2),
  ((messaging_communication(Channel, C1, C2),
  component_in_restricted_environment(Channel));
  direct_communication(C1, C2)).
```

The problem only occurs in case the communicating components are located in different locations (first condition) and if these locations are restricted. The second condition is checked by the `component_in_restricted_environment` fact, which is in turn a rule encapsulating a complex query. Indicator for a restricted environment is the key-value property *inbound_communication: false*. To identify if an integration is required, the components located in different restricted environments have to communicate. For this, either a message-based or a direct communication must be part of the deployment model. In case of messaging the Integration Provider pattern only has to be applied in case the Channel, and thus the Message Broker, is also located in a restricted environment. Otherwise, an additional integration provider is not required.

The `integration_of_restricted_environments` rule can be applied to the topology presented in **Fig. 3** on the right. In this example, a problem is detected because the two communicating components (PHP-App, Java-App) as well as the used messaging system are located in different restricted environments. After detecting the problem, the solution described by the pattern can be applied.

The two problem and context formalized patterns, Message Mover and Integration Provider, show how the problem detection approach presented by Saatkamp et al. [26] can be used for detecting problems in topology models. Relevant patterns for problem recognition are not restricted to only one pattern language. They can be found in different pattern languages. In application scenarios presented in this work and in [26] patterns from the security [28] and the cloud computing [13] pattern language have been selected. Using the Topology ProDec tool the different patterns are validated based on TOSCA topologies modeled with the TOSCA modeling tool Winery[6] [21]. The application of the pattern formalization approach results in reusable rules that serve as conditions to express the actual pattern rules. For example, the rules `components_in_different_locations` and `component_in_restricted_environments` are used several times. Such reusable condition rules ease the formalization of further patterns.

4 Related Work

The underlying approach applied in this paper is presented by Saatkamp et al. [26]. We applied the approach to further cloud integration patterns and extracted reusable condition rules that ease the formalization of problem and context descriptions of further patterns. The formalization of patterns is already addressed by several other works

[6] https://github.com/eclipse/winery

[3,10,14,19,22]. However, in contrast to our work the solution provided by a pattern is formalized to identify the implemented patterns in a model instead of the problem solved by the pattern. In this paper, we present how the context and problem described by a specific pattern can be formalized to detect possibly occurring problems that can be solved by applying the respective pattern to the topology-based deployment models.

Kim and Khawand [20] presented an approach to formalize the problem domain of design patterns. They specify the problem domain as UML diagrams. Compared to logic programming, this approach has the disadvantage that the non-existence of elements cannot be specified. Furthermore, the context of the pattern is important to identify if a problem exists, this is not considered in this work. As a result of a formalized problem domain of patterns, patterns can be identified that solve the detected problems.

An approach presented by Haitzer and Zdun [16] is based on predefined architectural primitives that represent entities used in several patterns. They can be used to annotate software components to identify if a pattern is applicable. This semi-automated approach focuses on the applicability of patterns in software code. The applicability of patterns is also focused by the automated management approach presented by Breitenbücher [5] and Breitenbücher et al. [6,7,8]. Based on the cloud computing patterns [13] management idioms are defined that specify the transformation from a target topology fragment that represents the current state of an application by its components and their relations to a desired state that reflects the applied management pattern. These topology fragments are graphs that must be matched to a subgraph in the overall topology representing the current state of the application. Based on similar mechanisms, i.e., using graph matching, Arnold et al. [2] and Eilam et al. [11] presented concepts to facilitate the transformation from abstract topology-based deployment models to concrete configurations of the contained components. Also Guth and Leymann [15] use graph fragments for rewriting and refining architectural graphs. However, their approaches are based on subgraph isomorphism to identify the target fragment and thus, the non-existence of elements cannot be detected which is important to detect problems in topology-based deployment models, as shown in the formalization of the `Secure Channel` pattern in [26].

5 Conclusion

In this work we presented two application scenarios of the problem detection approach using formalized patterns by Saatkamp et al. [26]. We applied the approach to the Message Mover and the Integration Provider pattern. Both patterns are related to distributed applications and, thus, relevant for restructured topology-based deployment models. We have demonstrated that the approach is also applicable to message-based systems. Furthermore, reusable rules that serve as conditions to express the actual pattern rules are defined. In future work, we want to extend the pattern collection and want to improve the tool support to ease the authoring process for new rules.

This approach is not limited to the presented cloud computing patterns [13] or security patterns [28]. The approach could also be extended to other patterns, such as the cloud data patterns [29] or the Internet of Things patterns [25]. Furthermore, the approach can also be used for a general validation of topology-based deployment models and is not limited to the usage in restructured deployment models. The extension to further domains will be investigated in future works.

Acknowledgements. This work was partially funded by the BMWi projects IC4F (01MA17008G) and SmartOrchestra (01MD16001F), and the German Research Foundation (DFG) project ADDCompliance (636503).

References

1. Alexander, C., Ishikawa, S., Silverstein, M.: A Pattern Language: Towns, Buildings, Construction, Oxford University Press (1977).
2. Arnold, W., Eilam, T., Kalantar, M., Konstantinou, A.V., Totak, A.A.: Pattern Based SOA Deployment. In: Proceedings of the 5th International Conference on Service-Oriented Computing, pp. 1-12. Springer (2007).
3. Bergenti, F., Poggi, A.: Improving UML Designs Using Automatic Design Pattern Detection. Handbook of Software Engineering and Knowledge Engineering, 771-784 (2002).
4. Bergmayr, A., Breitenbücher, U., Ferry, N., Rossini, A., Solberg, A., Wimmer, M., Kappel, G., Leymann, F.: A Systematic Review of Cloud Modeling Languages. ACM Computing Surveys 51(1), Article 22, 38 pages (2018).
5. Breitenbücher, U.: Eine musterbasierte Methode zur Automatisierung des Anwendungsmanagements. Dissertation, University of Stuttgart, Faculty of Computer Science, Electrical Engineering, and Information Technology (2016).
6. Breitenbücher, U., Binz, T., Kopp, O., Leymann, F.: Pattern-based Runtime Management of Composite Cloud Applications. In: Proceedings of the 3rd International Conference on Cloud Computing and Service Science, pp. 475-482. SciTePress (2013).
7. Breitenbücher, U., Binz T., Kopp, O., Leymann, F.: Automating Cloud Application Management Using Management Idioms. In: Proceedings of the 6th International Conference on Pervasive Patterns and Applications, pp. 60-69. Xpert Publishing Services (2014).
8. Breitenbücher, U., Binz, T., Kopp, O., Leymann, F., Wieland, M.: Context-Aware Cloud Application Management. In: Proceedings of the 4th International Conference on Cloud Computing and Services Science, pp. 499-509. SciTePress (2014).
9. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M.: Pattern-Oriented Software Architecture, Volume 1 – A System of Patterns. Wiley (1996).
10. Di Martino, B., Esposito, A.: A rule-based procedure for automatic recognition of design patterns in uml diagrams. Software: Practice and Experience 46(7), 983-1007 (2016).
11. Eilam, T., Kalantar, M., Konstantinou, A., Pacifici, G., Pershing, J., Agrawal, A.: Managing the configuration complexity of distributed applications in Internet data centers. Communications Magazine 44(3), 166-177 (2006).
12. Endres, C., Breitenbücher, U., Falkenthal, M., Kopp, O., Leymann, L., Wettinger, J.: Declarative vs. Imperative: Two Modeling Patterns for the Automated Deployment of Applications. In Proceedings of the 9th International Conference on Pervasive Patterns and Applications, pp. 22-27. Xpert Publishing Services (2017).
13. Fehling, C., Leymann, F., Retter, R., Schupeck, W., Arbitter, P.: Cloud Computing Patterns – Fundamentals to Design, Build, and Manage Cloud Applications. Springer (2004).

14. Fontana, F.A., Zanoni, M.: A tool for design pattern detection and software architecture reconstruction. Information sciences 181(7), 1306-1324 (2011).
15. Guth, J., Leymann, F.: Towards Pattern-based Rewrite and Refinement of Application Architectures. In: Proceedings of the 12th Advanced Summer School on Service Oriented Computing. IBM Research Division (2018).
16. Haitzer, T., Zdun, U.: Semi-automatic architectural pattern identification and documentation using architectural primitives. Journal if Systems and Software 102, 35-57 (2015).
17. Hohpe, G., Woolf, B.: Enterprise Integration Patterns: Design, Building, and Deploying Messaging Solutions. Addison-Wesley Professional (2004).
18. Jamshidi, P. Pahl, C., Chinenyeze, S., Liu X.: Cloud Migration Patterns: A Multi-Cloud Service Architecture Perspective. In: Service-Oriented Computing – ICSOC 2014 Workshop, pp. 6-19. Springer (2014).
19. Kampffmeyer, H., Zschaler, S.: Finding the pattern you need: The design pattern intent ontology. In: International Conference on Model Driven Engineering Languages and Systems, pp. 211-225. Springer (2007).
20. Kim, D.K., Khawand, C.E.: An approach to precisely specifying the problem domain of design patterns. Journal of Visual Languages and Computing 18(6), 560-591 (2007).
21. Kopp, O., Binz, T., Breitenbücher, U., Leymann, F.: Winery – A Modeling Tool for TOSCA-based Cloud Applications. In: Processing of the 11th International Conference on Service-Oriented Computing, pp. 700-704. Springer (2013).
22. Lim, D.K., Lu, L.: Inference of design pattern instances in uml models via logic programming. In: 11th IEEE International Conference on Engineering of Complex Computer Systems, pp. 10-29. IEEE (2006).
23. Meszaros, G., Doble, J.: MetaPatterns: A Pattern Language for Pattern Writing. In: Proceedings of International Conference on Pattern Languages of Program Design, pp. 164-200. ACM (1997).
24. OASIS: Topology and Orchestration Specification for Cloud Applications (TOSCA) Version 1.0 (2013).
25. Reinfurt, L., Breitenbücher, U., Falkenthal, M., Leymann, F., Riegg, A.: Internet of Things Patterns. In: Proceedings of the 21th European Conference on Pattern Languages of Programs, Article Nr. 5. ACM (2016).
26. Saatkamp, K., Breitenbücher, U., Kopp, O., Leymann, F.: An Approach to Automatically Detect Problems in Restructured Deployment Models Based on Formalizing Architecture and Design Patterns. Computer Science – Research and Development (2018).
27. Saatkamp, K., Breitenbücher, U., Kopp, O., Leymann, F.: Topology Splitting and Matching for Multi-Cloud Deployments. In: Proceedings of the 7th International Conference on Cloud Computing and Services Science, pp. 247-258. ScitePress (2017).
28. Schumacher, M., Fernandez-Buglioni, E., Hybertson, D., Buschmann, F., Sommerlad, P.: Security Patterns – Integration Security and System Engineering. John Wiley & Sons (2006).
29. Strauch, S., Andrikopolous, V., Breitenbücher, U. Sáez, S.G., Kopp, O., Leymann, F.: Using Patterns to Move the Application Data Layer to the Cloud. In: Proceedings of the 5th International Conference on Pervasive Patterns and Applications, pp. 26-33. Xpert Publishing Services (2013).
30. Wellhausen, T., Fiesser, A.: How to Write a Pattern? A Rough Guide for First-time Pattern Authors. In: Proceedings of the 16th European Conference on Pattern Languages of Programs. ACM (2012).

Intrusion Detection Attack Patterns in Cloud Computing: Trust and Risk Assessment

Alexandros Chrysikos[1]

[1]Dr. Alexandros Chrysikos, Cyber Security Research Group, School of Computing & Digital Media, London Metropolitan University, London, UK.
A.Chrysikos@londonmet.ac.uk

Abstract: Dependence on cloud services has been steadily increasing in recent years, as cloud services are an attractive option to offer flexibility and cost effectiveness through economies of scale. Cloud services are also exposed to security incidents, such as data breaches and other malicious activities. To mitigate risks to the confidentiality, integrity, and availability of assets, but also minimise loss to cloud service providers and users, the attack trust and risk elements need to be identified, classified, and prioritised. The aim of the proposed conceptual framework is to combine trust and risk assessment sources with data of risk assessment related to each attack pattern. This novel approach is a new qualitative solution to examine and determine symptoms, indicators, and vulnerabilities to detect the impact and likelihood of distributed attacks directed at cloud computing environments. The proposed framework might help to reduce false positive alarms and improve performance in Intrusion Detection Systems.

Keywords: Cloud computing, Trust Assessment, Risk Assessment, Attack Pattern, IDS, Ontology

1.1 Introduction

Cloud computing is a new emerging model in Information Technology (IT) that can enable convenient, ubiquitous, on-demand network access to a shared pool of configurable computing resources. Those resources can also be released with minimal management effort and interactions can be rapidly provisioned (Zhang et al. 2010). Cloud computing represents an opportunity for both service providers and consumers, through the improvement of IT agility, efficiency, and reliability to reduce the cost of IT technologies. Specifically, on-demand self-service, resource pooling, rapid elasticity and measured service, cloud computing systems automatically control and optimize resource usage in order to offer an alternative method to rent computing and storage infrastructure services (Zissis and Lekkas 2012).

Cloud services are provided dynamically to its users via internet, which can lead to several attacks threatening their confidentiality, integrity, and availability of the data stored in the cloud (Jadeja and Modi 2012). Detecting attacks can be challenging for security administrators. Therefore, the use of Intrusion Detection Systems (IDS) can aid both cloud providers and security administrators to monitor and analyse network traffic (Aikat et al. 2017). The reason for using such systems is to prevent attacks by employing detection algorithms. Such algorithms monitor symptoms, analyse attack patterns, and then produce a multitude of alarms known as false alarms (Duque and bin Omar 2015).

The proposed framework aims to analyse risks related to each attack pattern. Specifically, it calculates risks related to each symptom, indicator and vulnerability in order to define the attack risk score, and then generate an alert.

In the subsequent sections a review of related detection approaches in cloud computing is provided. The underpinning systems required for the recommended solution are also presented. Then, the author describes the proposed framework. In the concluding section, a discussion about recommendations for further research is presented.

1.2 Related Detection Approaches

When it comes to detection approaches, security researchers require a mechanism that can integrate and analyse a wide variety of data sources. Particularly, they need a mechanism that can process information that is generated by heterogenous sources implemented in any cloud computing environment. These mechanisms should aim to detect attack patterns and reduce false positive alarms.

Hansman et al (2005) employed five classifiers to describe different types of attack. Specifically, classification by attack vendor, classification by attack target, classification by operational impact, classification by informational impact, and classification by defense. All this information can provide the network administrator with data on how to mitigate or deter an attack. Amer and Hamilton (2010) developed an ontology based attack model to assess the security of an information system from an attacker's point of view. The aim of the assessment process is to evaluate the effects of an attack. The process consists of four stages. The first stage consists of identifying the system's vulnerabilities using automated vulnerability tools. These tools evaluate vulnerabilities of computer systems, applications or networks and generate sets of scan results. The second stage, involves determining the attacks that might occur due to the previously identified vulnerabilities. In the third stage, the possible effects of those vulnerabilities are analysed. The fourth and final stage the attack effects are calculated.

Patel et al. (2013) proposed a four dimensions approach that provides classification covering network and computer attacks. Specifically, it provides assistance in improving network and computer security, as well as language consistency through attack description. The first dimension focuses on classifying the attack.

The second classifies the target of the attack. The third provides vulnerability classification or uses criteria from Howard and Longstaff's (1998) approach. The fourth dimension, addresses the effects of the attack.

Ficco et al. (2013) recommended a hybrid and event correlation approach for detecting attack patterns. The process involves detecting symptoms by collecting diverse information at several cloud levels in order to perform a complex event analysis presented in an ontology.

All of the previously mentioned methodologies demonstrate beneficial ontology that may offer informative guidelines regarding cyber intrusions and attack analysis. However, there is lack of detail required to analyse all symptoms and attacks that could in return minimise the number of false positive alarms. For instance, the same attack in two different cloud services may have a different degree of impact, but in most existing systems it would be classed as a malicious attack by both services.

The proposed framework addresses this issue, of a system generating multiple false positive alarms, through the implication of risk and trust assessment analysis in the detection process. In this approach, all actors, such as cloud providers and cloud customers participate in the data analysis to achieve a high level of information and data processing. Before describing the proposed framework, though, the underpinning systems are presented.

1.3 Intrusion Detection System (IDS)

An IDS is very important in terms of preventing an attack against an Information Technology (IT) organisation. An IDS conducts a security system diagnosis to discover all suspicious activities based on detection algorithms. Specifically, those systems can help to deter and prevent actions related to security breaches, system flaws, as well as potential threats that may lead to system violations (Bace and Mell 2001).

On the other hand, an IDS system may detect many false actions, but it may also lead to a number of false positive alarms and authorized users identified as intruders. In a cloud computing environment where all resources are shared amongst cloud customers, this point becomes even more critical. In order to minimise the number of false positive alarms and improve the efficiency of attack detection in all cloud computing environments, the proposed framework includes both cloud service providers and cloud customers as part of the correlation process in all cloud layers, such as Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS).

1.4 Trust Assessment System

Trust assessment in cloud computing facilitates a variety of information sources at different levels of abstraction and several deployment models (SaaS, PaaS, IaaS). Therefore, trust evaluation and changing nature of trust relationships among different entities in the cloud paradigm become important points to be addressed (Subashini and Kavitha 2011). Specifically, trust assessment models include a collection of rules, elements, and process' to develop trust amongst the different entities in any computing paradigm. Cloud computing environment components such as databases, virtual machines, cloud service providers, cloud service customers, and cloud services are examples of different entities. Trust models are classified in two categories, decision models and evaluation models. These models are applied to the cloud computing paradigm and are further developed through their connection with trust assessment techniques (Moyano et al. 2012).

The cloud users' service-related needs are constantly changing in the diverse environment of cloud computing. Consequently, the role of various factors, such as feedback, ratings, and Quality of Service (QoS), in trust assessment is very important. There are four main trust assessment information sources. Specifically, direct and indirect interaction, Cloud Service Provider declarations, and Third Party assessment (Mouratidis et al. 2013).

Trust dimensions is the other significant area that measures the security strength and computes a trust value. A trust value comprises of various parameters that are necessary dimensions to measure cloud services' security (Huang and Nicol 2013).

1.5 Risk Assessment System

Risk assessment can be identified as the potential that a given attack will exploit vulnerabilities of an asset or a group of assets to cause loss or damage to the assets. According to the ISO 27005 Risk Management, risk is measured by evaluating the probability of successful attacks and the subsequent impact of those attacks, should they occur (Duque and bin Omar 2015).

Risk = Impact * Likelihood (Humphreys 2008)

Specifically, the term Impact refers to the degree of which a risk event might affect an enterprise, expressed in terms of: Confidentiality, Integrity, and Authentication. The term Likelihood refers to the possibility that a given event may occur (Duque and bin Omar 2015). The implementation of the aforementioned equation in the proposed framework aims to stimulate cloud customers to evaluate security risks and simplify the analysis of all identified events.

1.6 Proposed Framework for Attack Pattern Detection through Trust and Risk Assessment

The proposed framework is a predictive model that detects attack patterns based on trust assessment and risk assessment analysis. Figure 1 presents a correlation process that consists of a sequence of activities that are designed to analyse all network traffic through cloud layers (Valeur et al. 2004). The proposed framework applies a correlation process that intends to unify different steps of correlation by adding risk and trust assessment analysis in the diagnosis step, before the taxonomy step takes place.

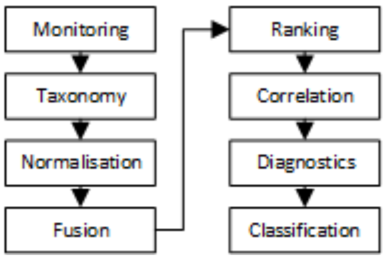


Figure 1: Correlation Process (Valeur et al. 2004)

An attack pattern is an abstraction mechanism that describes how an observed attack type is executed. Following the lifecycle of cyber-attack, when an attack occurs it uses several paths, from reconnaissance to exploitation, and aims to gain unauthorized access to data (Shin et al. 2013). Through studying the impact effects of an attack and simplifying the analysis of monitored events, then it could be possible to minimise false positive alarms.

Figure 2 shows the proposed framework's three essential security functions: (1) Monitoring & Data Collection, (2) Analysing & Detecting, and (3) Alarm & Respond.

(1) Monitoring & Data Collection. As a first step, the requirements of the organisation are defined based on monitoring the event management logs of all cloud layers (IaaS, PaaS, and SaaS). The next step is to collect data through Risk Software Agent (RSAg) programs. An RSAg is a goal-oriented computer program that reacts to its environment and operates without continuous direct supervision to perform its function. The RSAg programs store data from IaaS, PaaS, and SaaS. The data storage is structured in two separate knowledge databases that do not communicate. These are the Trust Assessment Database and the Risk Assessment Database. The reason for recommending two isolated databases is to reassure cloud providers for data pseudonymisation. The cloud providers processing of personal data is conducted in a way that the data can no longer be attributed to a specific data subject without the use of additional information (Bolognini and Bistolfi 2017). The pseudonymised information from those two databases is then combined in the Self-Learning Knowledge Base, which feeds with data the next function.

(2) Analysing & Detecting. The analysis of attack patterns is conducted by calculating the score of all indicators. Specifically, the proposed solution includes a definition for Risk (Ri) as a product of the Probability (Po) of a security compromise and its potential Impact (Im) (see 1).

$$Ri = Po * Im \quad (1)$$

The recommended correlation is used to aggregate the attack scenarios and symptoms generated by all parts in the cloud computing environment. The Impact (Im) is a value consisting of the following indicators: Trust Assessment Indicator (TaI), Vulnerability (Vu) and Symptoms (Sy). Each of these indicators has a different impact. The Probability (Po) value is increased in relation to each indicator of an attack pattern (see 2).

$$Im = TaI + Vu + Sy \quad (2)$$

The Impact (Im) and Probability (Po) of each indicator is defined by the cloud customer and cloud provider using data collected from all cloud layers. The aim is to use attackers' behavior to determine the Impact (Im) and expose a potential attacker before an attack can take place. The value of Risk (Ri) related to each attack determines whether the attack is successful or false positive alarm depending on the sensitivity of the targeted data as defined by the owner (cloud provider and cloud customer) (see 3). All this information is processed and stored in the Processing Knowledge Base.

$$Ri = Po * (TaI + Vu + Sy) \quad (3)$$

(3) Alarm & Respond. The risk of the attack is calculated and a response is sent whether it represents a suspicious threat or a false positive alarm. This is conducted with mechanisms that classify information about all attacks and determine the impact of each attack pattern and the risk of the attack. Specifically, the use of machine-learning procedures, such as supervised classification and clustering, and analytic algorithms has been proven useful to similar proactive detection and defense models (Fu et al. 2010; Osako et al. 2016). The respond function is conducted in the Decision Making server that determines the impact of every attack and serves as an Advice as a Service for the organisations.

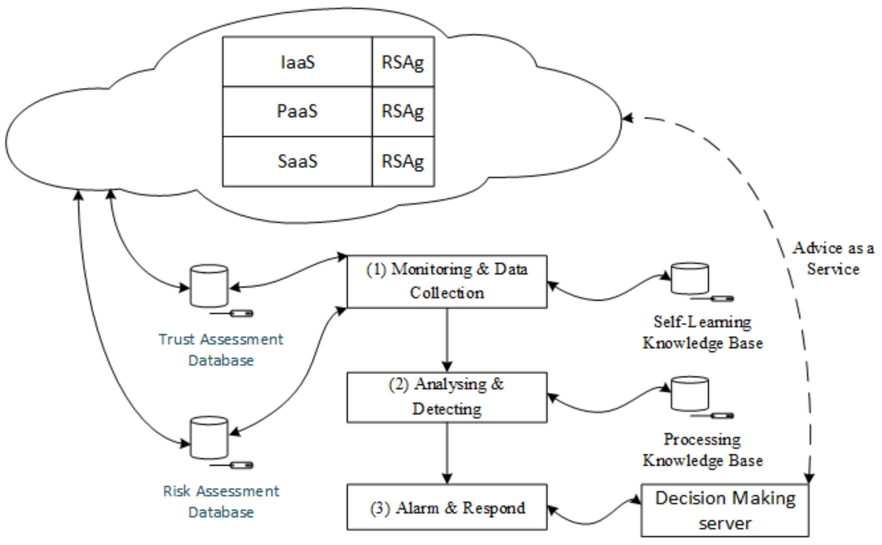

IaaS
RSAg
PaaS
RSAg
SaaS
RSAg
Advice as a Service
(1) Monitoring & Data Collection
Trust Assessment Database
Self-Learning Knowledge Base
(2) Analysing & Detecting
Processing Knowledge Base
Risk Assessment Database
(3) Alarm & Respond
Decision Making server

Figure 2: Proposed Framework for Attack Pattern Detection

1.7 Conclusion

In the current study a new framework for attack pattern detection in the cloud computing paradigm is proposed. A framework to recognise and analyse malicious actions based on risk and trust assessment factors and information sources related to attack patterns. Specifically, the recommended framework classifies attacks by evaluating the probability of a security breach and its potential impact indicators, such as trust assessment indicator, vulnerability, and symptoms. The outcome of this evaluation gives the likelihood of an attack pattern risk. Both cloud providers and cloud customers are involved in the data collection and correlation process. This classification might aid to protect data in the cloud and provide a method that could efficiently analyse suspicious attack actions and reduce false positive alarms.

In the cloud computing environment, risk and trust assessment need to be assessed continuously using multiple factors. These factors keep changing in the dynamic and constantly evolving cloud computing paradigm. Moreover, multi-cloud environments demand a more risk and trust assessment oriented analysis. Therefore, risk and trust assessment needs of cloud providers and cloud customers' have to be addressed in more detail. Therefore, a taxonomy and analysis of risk and trust assessment techniques in the cloud computing paradigm is required. Finally, future work should test the implementation of the suggested framework in an actual cloud computing environment.

References

Aikat J, Akella A, Chase JS, Juels A, Reiter M, Ristenpart T, Sekar V, Swift M (2017) Rethinking security in the era of cloud computing. IEEE Security & Privacy.

Amer SH, Hamilton J (2010) Intrusion detection systems (IDS) taxonomy-a short review. Defense Cyber Security, 13(2), 23-30.

Bace R, Mell P (2001) NIST special publication on intrusion detection systems. BOOZ-ALLEN AND HAMILTON INC MCLEAN VA.

Bolognini L, Bistolfi C (2017) Pseudonymization and impacts of Big (personal/anonymous) Data processing in the transition from the Directive 95/46/EC to the new EU General Data Protection Regulation. Computer Law & Security Review, 33(2), 171-181.

Duque S, bin Omar MN (2015) Using data mining algorithms for developing a model for intrusion detection system (IDS). Procedia Computer Science, 61, pp.46-51.

Ficco M, Tasquier L, Aversa R (2013) Intrusion detection in cloud computing. In P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2013 Eighth International Conference on (pp. 276-283). IEEE.

Fu, T., Abbasi, A. and Chen, H., 2010. A focused crawler for Dark Web forums. Journal of the Association for Information Science and Technology, 61(6), pp.1213-1231.

Hansman S, Hunt R. (2005) A taxonomy of network and computer attacks. Computers & Security, 24(1), 31-43.

Howard JD, Longstaff TA (1998) A common language for computer security incidents (No. SAND98-8667). Sandia National Labs, Albuquerque, NM (US); Sandia National Labs., Livermore, CA (US).

Humphreys E (2008) Information security management standards: Compliance, governance and risk management. Information security technical report, 13(4), 247-255.

Huang J, Nicol, DM (2013) Trust mechanisms for cloud computing. Journal of Cloud Computing: Advances, Systems and Applications, 2(1), 9.

Jadeja Y, Modi K (2012) Cloud computing-concepts, architecture and challenges. In Computing, Electronics and Electrical Technologies (ICCEET), International Conference on (pp. 877-880). IEEE.

Moyano F, Fernandez-Gago C, Lopez J (2012) A conceptual framework for trust models. In International Conference on Trust, Privacy and Security in Digital Business (pp. 93-104). Springer, Berlin, Heidelberg.

Mouratidis H, Shareeful I, Kalloniatis C, Gritzalis S (2013) A framework to support selection of cloud providers based on security and privacy requirements. Journal of Systems and Software 86: 2276–93.

Osako, T., Suzuki, T. and Iwata, Y., 2016. Proactive Defense Model Based on Cyber Threat Analysis. FUJITSU Sci. Tech. J, 52(3), pp.72-77.

Patel A, Taghavi M, Bakhtiyari K, JúNior JC (2013) An intrusion detection and prevention system in cloud computing: A systematic review. Journal of network and computer applications, 36(1), 25-41.

Shin JS, Son HS, Heo G (2013) Cyber security risk analysis model composed with activity-quality and architecture model. In International conference on computer, networks and communication engineering (pp. 609-612).

Subashini S, Kavitha V (2011) A survey on security issues in service delivery models of cloud computing. Journal of network and computer applications, 34(1), 1-11.

Valeur F, Vigna G, Kruegel C, Kemmerer RA (2004) Comprehensive approach to intrusion detection alert correlation. IEEE Transactions on dependable and secure computing, 1(3), 146-169.

Zhang Q, Cheng L, Boutaba R (2010) Cloud computing: state-of-the-art and research challenges. Journal of internet services and applications, 1(1), pp.7-18.

Zissis D, Lekkas D (2012) Addressing cloud computing security issues. Future Generation computer systems, 28(3), pp.583-592.

Supporting Application Deployment and Management in Fog Computing

Stefano Forti[⋆]

Department of Computer Science, University of Pisa, Italy
`stefano.forti@di.unipi.it`

Abstract. Deploying and managing multi-component IoT applications in Fog computing scenarios is challenging due to the heterogeneity, scale and dynamicity of Fog infrastructures, as well as to the complexity of modern software systems. When deciding on where/how to (re-)allocate application components over the continuum from the IoT to the Cloud, application administrators need to find the best deployment, satisfying all application (hardware, software, QoS, IoT) requirements over the contextually available resources, also trading-off non-functional desiderata (e.g., financial costs, security). This PhD thesis proposal aims at devising models, algorithms and methodologies to support the adaptive deployment and management of Fog applications.

Keywords: Fog computing · IoT · QoS-aware application deployment · Application Management.

1 Introduction

Context – Connected devices are changing the way we live and work. In the next years, the Internet of Things (IoT) is expected to bring more and more intelligence around us, being embedded in or interacting with the objects that we use every day [22, 21]. Self-driving cars, autonomous domotics systems, energy production plants, agricultural lands, supermarkets, healthcare, embedded AI will more and more exploit devices and Things that are integral part of the Internet and of our existence without us being aware of them.

As a consequence of this trend, enormous amounts of data – the so-called *Big Data* [42] – are collected by IoT sensors and stored in Cloud data centres [35]. Once there, data are subsequently analysed to determine reactions to events or to extract analytics or statistics. Whilst data-processing speeds have increased rapidly, bandwidth to carry data to and from data centres has not increased equally fast [47]. On one hand, supporting the transfer of data from/to billions of IoT devices is becoming hard to accomplish due to the volume and geo-distribution of those devices. On the other hand, the need to reduce latency for time-sensitive applications, to eliminate mandatory connectivity requirements, and to support computation closer to where data is generated 24/7, is evident [9].

[⋆] PhD Thesis Supervisor: Prof. Antonio Brogi, University of Pisa, Italy.

Fog Computing – Recent research efforts are investigating how to better exploit capabilities along the continuum from the edge of the Internet to the Cloud data centres, to support new IoT applications and their needs. Computational nodes closer to the edge will act both as *filters* – reducing the amount of data sent to the Cloud – and as *processing capabilities* – producing analytics closer to where data is being sensed or used.

Among the existing proposals, *Fog computing* [10, 26] aims at better supporting the growing processing demand of (time-sensitive and bandwidth hungry) IoT applications by selectively pushing computation closer to where data is produced and by relying on a geographically distributed multitude of heterogeneous devices (e.g., personal devices, gateways, micro-data centres, embedded servers) spanning the continuum from the Cloud to the IoT. A substantial amount of computation, storage and networking is therefore expected to happen closer to where data is produced and to IoT-based cyber-physical systems, contiguously to and interdependently with the Cloud. In general, Fog computing platforms are expected to guarantee that processing always occurs wherever it is *best-placed* for any given IoT application, thereby accelerating the velocity of decision making, by enabling prompter responses to sensed events [45].

Scope of the Thesis – Modern large-scale applications are not monolithic anymore [51]. Therefore, an application running in a Fog computing infrastructure consists of a set of independently deployable components (or services, or microservices) that work together and must meet some requirements. Deploying and managing such applications in Fog computing scenarios is, therefore, a challenging task. Indeed, it requires to dynamically map each of the (possibly many) application components (i.e., functionalities) to the computational node(s) that will host them at runtime.

Whilst some application functionalities are naturally suited to the Cloud (e.g., service back-ends) and others are naturally suited to edge devices (e.g., industrial control loops), there are applications for which functionality segmentation is not as straightforward (e.g., short to medium term analytics). Future tools for the deployment and management of IoT applications should consider application requirements (i.e., hardware, software, IoT, QoS), infrastructure capabilities (i.e., hardware, software, IoT devices, network conditions, security) and deployers' desiderata (i.e., business and security policies, cost constraints) to efficiently support adaptive segmentation of functionalities from the Cloud to the IoT.

In this context, we are investigating the design, prototyping and validation of novel models, and predictive algorithms and methodologies which will be useful to *(i)* process data about the application, the infrastructure and their monitored performance so to informedly suggest how to (re-)distribute application components, *(ii)* identify and validate the best sequence of actions to (re-)distribute components to different Fog or Cloud nodes based on specified policies, and *(iii)* choose when/how to (re-)deploy, (re-)configure or scale components in response to workload or network variations, churn and failures.

2 State of the Art

The problem of deciding how to deploy multi-component applications has been thoroughly studied in the Cloud scenario. Projects like SeaClouds [16], Aeolus [28] or Cloud-4SOA [24], for instance, proposed model-driven optimised planning solutions to deploy software applications across different (IaaS or PaaS) Clouds. [39] proposed to use OASIS TOSCA [17] to model IoT applications in Cloud+IoT scenarios. Also, solutions to automatically provision and configure software components in Cloud (or multi-Cloud) scenarios are currently used by the DevOps community to automate application deployment or to lead deployment design choices (e.g., Puppet [3] and Chef [2]). However, only few efforts in Cloud computing considered non-functional requirements *by-design* [44, 25] or uncertainty of execution (as in Fog nodes) and security risks among interactive and interdependent components [61]. With respect to the Cloud paradigm, the Fog introduces new problems, mainly due to its pervasive geo-distribution and heterogeneity, need for QoS-awareness, dynamicity and support to interactions with the IoT, that were not taken into account by previous works [56, 62, 4].

Among the first proposals investigating this new lines, [34] proposed a Fog-to-Cloud search algorithm as a first way to determine an eligible deployment of (multi-component) DAG applications to tree-like Fog infrastructures. Their placement algorithm proceeds Edge-ward, i.e. it attempts the placement of components *Fog-to-Cloud* by considering hardware capacity only. An open-source simulator – iFogSim – has been released to test the proposed policy against Cloud-only deployments. Building on top of iFogSim, [40] refines the Edge-ward algorithm to guarantee the application service delivery deadlines and to optimize Fog resource exploitation. Limiting their work to linear application graphs and tree-like infrastructure topologies, [60] used iFogSim to implement an algorithm for optimal online placement of application components, with respect to load balancing. An approximate extension handling tree-like application was also proposed. Recently, exploiting iFogSim, [33] proposed a distributed search strategy to find the best service placement in the Fog, which minimises the distance between the clients and the most requested services, based on request rates and available free resources. Their results showed a substantial improvement on network usage and service latency for the most frequently called services. [36] proposed a (linearithmic) heuristic algorithm that attempts deployments prioritising placement of smaller applications to devices with less free resources. Along the same line, [54] proposed an Edge-ward linearithmic algorithm that assigns application components to the node with the lowest capacity that can satisfy all application requirements.

From an alternative viewpoint, [57] proposed the design of a framework for application deployment in Fog computing, based on Integer Linear Programming (ILP). [5] in addition to proposing a Fog architectural framework, gave a Mixed-Integer Non-Linear Programming (MINLP) formulation of the problem of placing application components so to satisfy end-to-end delay constraints. The problem is then solved by linearisation into a Mixed-Integer Linear Programming (MILP), showing potential improvements in latency, energy consumption and costs for routing and storage that the Fog might bring. Skarlat et al. designed a hierarchical modelling of Fog infrastructures, consisting of a centralised management system to control Fog nodes organised per *colonies* ([48, 50, 49]). Particularly, [48] adopted an ILP formulation of the problem of allocating computation to Fog nodes in order to optimise (user-defined) time deadlines on application execution, considering IoT devices needed to properly run the application. A simple linear model for the Cloud costs is also taken into account. Similar solutions were proposed, attempting to optimise various metrics such as access latency, resource usage, energy consumption or data migrations cost [64, 32, 65, 53, 7, 37]. [41] described instead a fuzzy QoE extension of iFogSim – based on an ILP modelling of users expectation – which achieved improvements in network conditions and service QoS.

Regrettably, none of the discussed ILP/MILP approaches came with the code to run the experiments. Conversely, [58] proposed a software platform to support optimal application placement in the Fog, within the framework of the CoSSMic European Project [1]. Envisioning resource, bandwidth and response time constraints, they compare a Cloud-only, a Fog-only or a Cloud-to-Fog deployment policy. Additionally, the authors of [18, 20, 19] released S-ODP, an open-source extension of Apache Storm that performs components placement with the goal of minimising the end-to-end application latency and the availability of deployed applications. Finally, also dynamic programming (e.g., [46, 52], genetic algorithms (e.g., [48, 50]) and deep learning (e.g., [55]) were exploited to tackle the placement of application components with some promising results.

After the first deployment, the management of applications in the Fog is also time-consuming and error-prone to be tuned manually, lacking adequate support. [43] proposed a MAPE-K loop to identify action plans to minimise SLA violations while maximising the use of allocated resources by simulating different strategies to manage deployed applications. [30] highlighted the need to check for inconsistencies that can arise within or between different stages of a deployment plan. [30] proposed a deployment management system model to enable the automated generation of deployment plans for distributed infrastructures after identifying (with static analysis techniques) possible flaws in deployment plan specifications. The use of formal models to verify properties of application deployments to Cloud infrastructure has been advocated by various authors. [38] for instance defined a process calculus to specify deployment, migration and security policies of virtual machines (VMs) across different Clouds, in order to enable the verification of security policies after live VM reconfigurations. [6] proposed a similar approach to preserve data consistency when migrating deployed applications in Fog scenarios. [29] proposed a pseudo-dynamic testing approach, which combines emulation, simulation, and existing real testbeds, whilst leveraging multiple methodologies to test complex and large Fog infrastructures taking into account also scalability and churn conditions. While various proposals exist to automate the management of applications, to verify the correctness of deployments to the Cloud, to the best of our knowledge, none of the existing approaches addresses the validation of application management for the Fog.

3 Thesis Objectives

This section aims at illustrating the objectives of the thesis work, seeking to suitably support automated application deployment (and functionality allocation) in Fog computing. The provision of adequate support to adaptively deploy applications and manage their components in Fog scenarios is among the crucial steps for the success of Fog computing. In this context, we intend to design, prototype and validate novel models, and predictive algorithms and methodologies, which will improve the decision-making process related to the life-cycle management of Fog applications.
In what follows, we detail the research goals we intend to accomplish during this research, from the point of view of modelling (Section 3.1), design of algorithms and methodologies (Section 3.2), and prototyping and validation (Section 3.3).

3.1 Modelling

First, we aim at contributing to the modelling of the Fog scenario with a particular focus on:

1. describing arbitrary *multi-component applications* topologies considering their processing (e.g., hardware, software and IoT devices), QoS (e.g., latency, bandwidth, security) requirements and component inter-dependencies, along with the possibility for their components to scale both vertically and horizontally, according to workload demand and behaviour models,
2. describing accordingly *Fog infrastructures* in terms of their capabilities (i.e., Cloud data-centres, Fog nodes, Things) and previous performance/utilisation (e.g., QoS of communication links, historical data on nodes utilisation, reliability of nodes and links), considering IoT-Fog, Fog-Fog and Fog-Cloud interactions,
3. accounting for *dynamicity* and *churn* of the infrastructure (e.g., variations in the QoS of communication links, mobility of IoT devices and Fog nodes, failures) and in the users' demand, as well as for *application scalability* on heterogeneous devices so to be able to plan for scalable, reliable and dependable application deployments,
4. including the possibility of expressing *preferences* on application deployment that have to be enforced due to particular end-user targets (e.g., QoS-assurance, financial budget, resource usage) or deployment needs (e.g., security, trust, reliability, energy consumption),
5. identifying and devising appropriate *metrics* and *performance indicators* (e.g., QoS-assurance, resource consumption, reliability) to characterise eligible application deployments and plans, also considering their behaviour over time, as well as financial costs and energy consumption to keep the application up and running.

Naturally, to support the deployment of applications to Fog infrastructures, we intend to accompany the devised models with novel algorithms and methodologies that exploit them as illustrated in the next section.

3.2 Algorithms and Methodologies

To exploit the models described in the previous section, we intend to devise algorithms and methodologies in order to:

1. efficiently *determine eligible* context- and QoS-aware *deployments* of application components to Fog infrastructures, according to different strategies and by adopting proper heuristics to reduce the search space, whilst selecting cost-/energy- aware matchings between application requirements (viz., hardware and software) and available Fog/Cloud offerings,
2. *simulate* and *predict* the (expected) behaviour of different eligible deployments under the proposed metrics at varying *(i)* QoS of available communication links, *(ii)* available resources in the current state of the infrastructure, *(iii)* workload and users demand, also considering historical data about the monitored infrastructure and feedback about previously enacted deployments,
3. *compare* and recommend and/or automatically select *best candidate deployments* – among the eligible ones – based on predicted metrics, expressed targets and historical data, by plotting results to empower experts to make informed choices, and by exploiting multi-objective optimisation or learning techniques,
4. *determine* and *optimise plans* that take into account dependencies between components so to perform application deployment to a given infrastructure, envisioning deployment (vertical and horizontal) scalability on heterogeneous devices and optimal resources exploitation (e.g., hardware, energy), and considering alternative backup deployments to tackle dynamicity issues (e.g., increasing workload, mobility, QoS variations, churn and failures),
5. understand when to trigger and how to (optimally) perform *reconfiguration actions* (e.g., enactment of an alternative plan), scaling of application components, or components re-allocation to different nodes so to guarantee QoS or SLA constraints will be met by enacted deployments, whilst avoiding (or minimising) the likelihood of service disruption.

3.3 Prototyping and Validation

To provide some validation to our approaches we aim, when possible, at providing formal properties (e.g., correctness, completeness) of the proposed methodologies, along with a systematic evaluation of their computational complexity. Then, we plan to prototype all proposed models and methodologies in open-source tools, so to show feasibility, utility and practicality of the devised solutions.

Finally, with the purpose of testing and demonstrating our prototypes at work, we aim at designing lifelike use cases and testbeds, by implementing meaningful IoT applications and deploying them to experimental Fog infrastructures.

4 First Results

The first results of this work have been already published in some conferences and journals. In this section, we briefly summarise them and the research they triggered in the community.

QoS-aware Deployment of Fog Applications – In [11], we proposed a simple, yet general, model of multi-component IoT applications and Fog infrastructures. After proving that the problem of determining eligible deployments is NP-hard, we devised a heuristic backtracking search algorithm to solve it and we run it on a motivating example from smart agriculture (viz., 3 application components, 2 Clouds, 3 Fog nodes). The heuristic attempts the placement of components sorted in ascending order on the number of compatible nodes (i.e., *fail-first*), considering candidate nodes one by one sorted in decreasing order on the available resources (i.e., *fail-last*).

In [12], we combined an exhaustive version of our search algorithm with Monte Carlo simulations so to consider variations in the QoS of communication links (modelled by probability distributions) and to predict how likely a deployed application is to comply with the desired network QoS (viz., latency and bandwidth) and how much Fog resources it will consume. In [13], we further enhanced the proposed methodology by proposing a cost model that extends Cloud cost models to Fog scenarios and integrates them with costs coming from the IoT. It is worth noting that, with respect to the majority of related works, our approach works on arbitrary application and infrastructure graph topologies.

All proposed predictive methodologies have been implemented in an open-source prototype[1], FogTorchΠ, and are described in detail in [14], which also offers a comparison with one of the first tools for simulating Fog scenarios (iFogSim [34]). FogTorchΠ can be used to determine, simulate and compare eligible deployments of applications to given infrastructures in a QoS- (with respect to network variations), context- (with respect to the considered resources), and cost-aware (estimating monthly revenues and outflows) manner, meeting all deployers' desiderata. Despite exploiting worst-case exponential-time algorithms, the prototype has been shown to scale [14] also on the larger VR game example (viz., 3 to 66 app components, 1 Cloud, up to 80 Fog nodes) proposed in [34].

Inspired by FogTorchΠ models and algorithms, Xia et al. [63] proposed a backtracking solution to FAPP to minimise the average response time of deployed IoT applications. Two new heuristics were devised. The first one sorts the nodes considered for deploying each component in ascending order with respect to the (average) latency between each node and the IoT devices required by the component. The second one considers a component that caused backtracking as the first one to be mapped in the next search step. Despite discussing improved results on latency with respect to exhaustive backtracking and first-fit strategies, no prototype implementations were released. Finally, FogTorchΠ was also modularly extended by De Maio et al. [27] to simulate mobile task offloading in Edge computing scenarios.

[1] Available at: `https://github.com/di-unipi-socc/FogTorchPI/`

Mimicking Fog Application Management – CISCO FogDirector [23] is among the first available management tools for large-scale production deployments of Fog applications. It provides centralised management services that span the entire lifecycle of Fog applications, and it can be used via REST APIs that enable integration with client programs implementing application management. In [31] we presented a simple operational semantics of all basic functionalities of FogDirector, describing the effects of the operations that client programs can perform to publish, deploy, configure, start, monitor, stop, undeploy and retire their applications in a Fog-Director-managed infrastructure. Based on the given formalisation, we implemented a prototype[2], FogDirMime, which is the core of a simulator environment for FogDirector. The prototype also simulates probabilistic (hardware and network QoS) variations of the infrastructure that happen independently from the considered application management.

On one hand, the proposed semantics constitutes a concise and unambiguous reference of the (basic) behaviour of FogDirector that can be used to quickly understand its functioning and to check the correctness of management scripts at design time. On the other hand, FogDirMime can be fruitfully exploited to experiment and compare different application management policies, so to predict their effectiveness and tune them in a simulated environment, according to user-defined metrics. The prototype was used over a smart building use case.

5 Conclusions & Future Work

We consider our preliminary results and prototypes the first promising steps to support decision-making when deploying or managing IoT applications to Fog infrastructures. Yet, such results clearly present some limitations with respect to the objectives of this thesis, as set in Section 3.

In our future work, we intend to:

1. extend our methodologies to include more aspects of the life-cycle of application management, including new features such as components upgrade, reconfiguration and scaling, while envisioning the possibility for components to be deployed in different flavours like in *Osmotic Computing* [59],
2. consider new metrics and dimensions that will be important in Fog scenarios (e.g., security, mobility, energy consumption) and propose ways to automatically and efficiently select best candidate (re-)deployments – i.e., matching deployers' desiderata – using (explainable) probabilistic AI [8] or multi-objective optimisation, and
3. prototype, validate and assess all new methodologies as extensions to our prototypes or as new open-source tools that can synergically work with them, and assess them in controlled settings (e.g., over the simple Fog application we proposed in [15]) as well as, possibly, in lifelike Fog environments.

Naturally, we plan to validate the proposed approaches by formally proving the correctness and completeness of the proposed algorithms, when possible.

[2] Available at: `https://github.com/di-unipi-socc/FogDirMime/`

References

1. CoSSMiC – Collaborating Smart Solar-powered Microgrids, `http://cossmic.eu/`
2. Opscode. Chef. http://www.opscode.com/chef/
3. Puppetlabs. Puppet. http://puppetlabs.com
4. Arcangeli, J.P., Boujbel, R., Leriche, S.: Automatic deployment of distributed software systems: Definitions and state of the art. Journal of Systems and Software **103**, 198–218 (2015)
5. Arkian, H.R., Diyanat, A., Pourkhalili, A.: Mist: Fog-based data analytics scheme with cost-efficient resource provisioning for iot crowdsensing applications. Journal of Network and Computer Applications **82**, 152 – 165 (2017). https://doi.org/https://doi.org/10.1016/j.jnca.2017.01.012, `http://www.sciencedirect.com/science/article/pii/S1084804517300188`
6. Bao, W., Yuan, D., Yang, Z., Wang, S., Li, W., Zhou, B.B., Zomaya, A.Y.: Follow Me Fog: Toward Seamless Handover Timing Schemes in a Fog Computing Environment. IEEE Communications Magazine **55**(11), 72–78 (Nov 2017). https://doi.org/10.1109/MCOM.2017.1700363
7. Barcelo, M., Correa, A., Llorca, J., Tulino, A.M., Vicario, J.L., Morell, A.: Iot-cloud service optimization in next generation smart environments. IEEE Journal on Selected Areas in Communications **34**(12), 4077–4090 (Dec 2016). https://doi.org/10.1109/JSAC.2016.2621398
8. Belle, V.: Logic meets probability: towards explainable ai systems for uncertain worlds. In: Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI. pp. 19–25 (2017)
9. Bonomi, F., Milito, R., Natarajan, P., Zhu, J.: Fog computing: A platform for internet of things and analytics. In: Big Data and Internet of Things: A Roadmap for Smart Environments, pp. 169–186 (2014)
10. Bonomi, F., Milito, R., Zhu, J., Addepalli, S.: Fog computing and its role in the internet of things. In: Proceedings of the first edition of the MCC workshop on Mobile cloud computing. pp. 13–16. ACM (2012)
11. Brogi, A., Forti, S.: QoS-Aware Deployment of IoT Applications Through the Fog. IEEE Internet of Things Journal **4**(5), 1185–1192 (Oct 2017). https://doi.org/10.1109/JIOT.2017.2701408
12. Brogi, A., Forti, S., Ibrahim, A.: How to best deploy your Fog applications, probably. In: Rana, O., Buyya, R., Anjum, A. (eds.) Proceedings of 1st IEEE International Conference on Fog and Edge Computing, Madrid (2017)
13. Brogi, A., Forti, S., Ibrahim, A.: Deploying Fog Applications: How Much Does It Cost, By the Way? In: Proceedings of the 8th International Conference on Cloud Computing and Services Science, CLOSER 2018, Funchal, Madeira, Portugal, March 19-21, 2018. pp. 68–77 (2018)
14. Brogi, A., Forti, S., Ibrahim, A.: Predictive Analysis to Support Fog Application Deployment. In: Buyya, R., Srirama, S.N. (eds.) Fog and Edge Computing: Principles and Paradigms. Wiley (2018), *In press*
15. Brogi, A., Forti, S., Ibrahim, A., Rinaldi, L.: Bonsai in the fog: An active learning lab with fog computing. In: Fog and Mobile Edge Computing (FMEC), 2018 Third International Conference on. pp. 79–86. IEEE (2018)
16. Brogi, A., Ibrahim, A., Soldani, J., Carrasco, J., Cubo, J., Pimentel, E., D'Andria, F.: SeaClouds: a European project on seamless management of multi-cloud applications. ACM SIGSOFT SEN **39**(1), 1–4 (2014)

17. Brogi, A., Soldani, J., Wang, P.: TOSCA in a Nutshell: Promises and Perspectives, pp. 171–186. Proceedings of ESOCC 2014. (2014)
18. Cardellini, V., Grassi, V., Lo Presti, F., Nardelli, M.: Distributed qos-aware scheduling in storm. In: Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems. pp. 344–347. ACM (2015)
19. Cardellini, V., Grassi, V., Lo Presti, F., Nardelli, M.: Optimal operator placement for distributed stream processing applications. In: Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems. pp. 69–80. DEBS '16, ACM, New York, NY, USA (2016). https://doi.org/10.1145/2933267.2933312, `http://doi.acm.org/10.1145/2933267.2933312`
20. Cardellini, V., Grassi, V., Presti, F.L., Nardelli, M.: On qos-aware scheduling of data stream applications over fog computing infrastructures. In: Computers and Communication (ISCC), 2015 IEEE Symposium on. pp. 271–276. IEEE (2015)
21. CISCO: Cisco Global Cloud Index: Forecast and Methodology, 20162021 (2015)
22. CISCO: Fog computing and the internet of things: Extend the cloud to where the things are (2015), `https://www.cisco.com/c/dam/en_us/solutions/trends/iot/docs/computing-overview.pdf`
23. CISCO: Cisco Fog Director Reference Guide (v. 1.5) (2017), `https://www.cisco.com/c/en/us/td/docs/routers/access/800/software/guides/iox/fog-director/reference-guide/1-5/fog_director_ref_guide.html`
24. Corradi, A., Foschini, L., Pernafini, A., Bosi, F., Laudizio, V., Seralessandri, M.: Cloud PaaS Brokering in Action: The Cloud4SOA Management Infrastructure. In: VTC 2015. pp. 1–7 (2015)
25. Cucinotta, T., Anastasi, G.F.: A heuristic for optimum allocation of real-time service workflows. In: Service-Oriented Computing and Applications (SOCA), 2011 IEEE Int. Conf. on. pp. 1–4. IEEE (2011)
26. Dastjerdi, A.V., Buyya, R.: Fog Computing: Helping the Internet of Things Realize its Potential. Computer **49**(8), 112–116 (2016)
27. De Maio, V., Brandic, I.: First hop mobile offloading of dag computations (2018), *In press*
28. Di Cosmo, R., Eiche, A., Mauro, J., Zavattaro, G., Zacchiroli, S., Zwolakowski, J.: Automatic deployment of software components in the cloud with the aeolus blender. In: ICSOC 2015, pp. 397–411 (2015)
29. Ficco, M., Esposito, C., Xiang, Y., Palmieri, F.: Pseudo-Dynamic Testing of Realistic Edge-Fog Cloud Ecosystems. IEEE Communications Magazine **55**(11), 98–104 (Nov 2017). https://doi.org/10.1109/MCOM.2017.1700328
30. Fischer, J., Majumdar, R., Esmaeilsabzali, S.: Engage: A Deployment Management System. SIGPLAN Not. **47**(6), 263–274 (Jun 2012). https://doi.org/10.1145/2345156.2254096
31. Forti, S., Ibrahim, A., Brogi, A.: Mimicking FogDirector Application Management. Computer Science - Research and Development (2018), *In press*
32. Gu, L., Zeng, D., Guo, S., Barnawi, A., Xiang, Y.: Cost efficient resource management in fog computing supported medical cyber-physical system. IEEE Transactions on Emerging Topics in Computing **5**(1), 108–119 (Jan 2017). https://doi.org/10.1109/TETC.2015.2508382
33. Guerrero, C., Lera, I., Juiz, C.: A lightweight decentralized service placement policy for performance optimization in fog computing. Journal of Ambient Intelligence and Humanized Computing (Jun 2018). https://doi.org/10.1007/s12652-018-0914-0

34. Gupta, H., Vahid Dastjerdi, A., Ghosh, S.K., Buyya, R.: iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, Edge and Fog computing environments. Software: Practice and Experience **47**(9), 1275–1296 (2017)
35. Hashem, I.A.T., Yaqoob, I., Anuar, N.B., Mokhtar, S., Gani, A., Khan, S.U.: The rise of big data on cloud computing: Review and open research issues. Information Systems **47**, 98–115 (2015)
36. Hong, H.J., Tsai, P.H., Hsu, C.H.: Dynamic module deployment in a fog computing platform. In: 2016 18th Asia-Pacific Network Operations and Management Symposium (APNOMS). pp. 1–6 (Oct 2016). https://doi.org/10.1109/APNOMS.2016.7737202
37. Huang, Z., Lin, K.J., Yu, S.Y., Hsu, J.Y.j.: Co-locating services in iot systems to minimize the communication energy cost. Journal of Innovation in Digital Ecosystems **1**(1-2), 47–57 (2014)
38. Jarraya, Y., Eghtesadi, A., Debbabi, M., Zhang, Y., Pourzandi, M.: Cloud Calculus: Security Verification in Elastic Cloud Computing Platform. In: 2012 International Conference on Collaboration Technologies and Systems (CTS). pp. 447–454 (May 2012). https://doi.org/10.1109/CTS.2012.6261089
39. Li, F., Vögler, M., Claeßens, M., Dustdar, S.: Towards automated IoT application deployment by a cloud-based approach. In: SOCA 2013. pp. 61–68 (2013)
40. Mahmud, R., Ramamohanarao, K., Buyya, R.: Latency-aware application module management for fog computing environments. ACM Transactions on Internet Technology (TOIT) (2018)
41. Mahmud, R., Srirama, S.N., Ramamohanarao, K., Buyya, R.: Quality of experience (qoe)-aware placement of applications in fog computing environments. Journal of Parallel and Distributed Computing (2018)
42. Manyika, J., Chui, M., Brown, B., Bughin, J., Dobbs, R., Roxburgh, C., Byers, A.H.: Big data: The next frontier for innovation, competition, and productivity. 2011 **5**(33), 222 (2014)
43. Maurer, M., Brandic, I., Sakellariou, R.: Adaptive resource configuration for Cloud infrastructure management. Future Generation Computer Systems **29**(2), 472 – 487 (2013). https://doi.org/https://doi.org/10.1016/j.future.2012.07.004, special section: Recent advances in e-Science
44. Nathuji, R., Kansal, A., Ghaffarkhah, A.: Q-clouds: Managing performance interference effects for qos-aware clouds. Association for Computing Machinery, Inc. (April 2010)
45. OpenFog: OpenFog Reference Architecture (2016)
46. Rahbari, D., Nickray, M.: Scheduling of fog networks with optimized knapsack by symbiotic organisms search. In: 2017 21st Conference of Open Innovations Association (FRUCT). pp. 278–283 (Nov 2017). https://doi.org/10.23919/FRUCT.2017.8250193
47. Shi, W., Dustdar, S.: The Promise of Edge Computing. Computer **49**(5), 78–81 (2016)
48. Skarlat, O., Nardelli, M., Schulte, S., Dustdar, S.: Towards qos-aware fog service placement. In: 2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC). pp. 89–96 (May 2017). https://doi.org/10.1109/ICFEC.2017.12
49. Skarlat, O., Schulte, S., Borkowski, M., Leitner, P.: Resource provisioning for iot services in the fog. In: 2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA). pp. 32–39 (Nov 2016). https://doi.org/10.1109/SOCA.2016.10

50. Skarlat, O., Nardelli, M., Schulte, S., Borkowski, M., Leitner, P.: Optimized IoT service placement in the fog. Service Oriented Computing and Applications **11**(4), 427–443 (Dec 2017). https://doi.org/10.1007/s11761-017-0219-8
51. Sommerville, I.: Software Engineering. Pearson (2015), 10th edition
52. Souza, V.B., Masip-Bruin, X., Marin-Tordera, E., Ramirez, W., Sanchez, S.: Towards distributed service allocation in fog-to-cloud (f2c) scenarios. In: 2016 IEEE Global Communications Conference (GLOBECOM). pp. 1–6 (Dec 2016). https://doi.org/10.1109/GLOCOM.2016.7842341
53. Souza, V.B.C., Ramrez, W., Masip-Bruin, X., Marn-Tordera, E., Ren, G., Tashakor, G.: Handling service allocation in combined fog-cloud scenarios. In: 2016 IEEE International Conference on Communications (ICC). pp. 1–5 (May 2016). https://doi.org/10.1109/ICC.2016.7511465
54. Taneja, M., Davy, A.: Resource aware placement of iot application modules in fog-cloud computing paradigm. In: 2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM). pp. 1222–1228 (May 2017). https://doi.org/10.23919/INM.2017.7987464
55. Tang, Z., Zhou, X., Zhang, F., Jia, W., Zhao, W.: Migration modeling and learning algorithms for containers in fog computing. IEEE Transactions on Services Computing pp. 1–1 (2018). https://doi.org/10.1109/TSC.2018.2827070
56. Varshney, P., Simmhan, Y.: Demystifying fog computing: Characterizing architectures, applications and abstractions. In: 2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC). pp. 115–124 (May 2017). https://doi.org/10.1109/ICFEC.2017.20
57. Velasquez, K., Abreu, D.P., Curado, M., Monteiro, E.: Service placement for latency reduction in the internet of things. Annals of Telecommunications **72**(1-2), 105–115 (2017)
58. Venticinque, S., Amato, A.: A methodology for deployment of iot application in fog. Journal of Ambient Intelligence and Humanized Computing (Apr 2018). https://doi.org/10.1007/s12652-018-0785-4
59. Villari, M., Fazio, M., Dustdar, S., Rana, O., Ranjan, R.: Osmotic computing: A new paradigm for edge/cloud integration. IEEE Cloud Computing **3**(6), 76–83 (2016)
60. Wang, S., Zafer, M., Leung, K.K.: Online placement of multi-component applications in edge computing environments. IEEE Access **5**, 2514–2533 (2017). https://doi.org/10.1109/ACCESS.2017.2665971
61. Wen, Z., Caa, J., Watson, P., Romanovsky, A.: Cost effective, reliable and secure workflow deployment over federated clouds. IEEE Transactions on Services Computing **10**(6), 929–941 (Nov 2017). https://doi.org/10.1109/TSC.2016.2543719
62. Wen, Z., Yang, R., Garraghan, P., Lin, T., Xu, J., Rovatsos, M.: Fog orchestration for internet of things services. IEEE Internet Computing **21**(2), 16–24 (Mar 2017). https://doi.org/10.1109/MIC.2017.36
63. Xia, Y., Etchevers, X., Letondeur, L., Coupaye, T., Desprez, F.: Combining hardware nodes and software components ordering-based heuristics for optimizing the placement of distributed iot applications in the fog. In: Proceedings of the 33rd Annual ACM Symposium on Applied Computing. pp. 751–760. ACM (2018)
64. Yang, L., Cao, J., Liang, G., Han, X.: Cost aware service placement and load dispatching in mobile cloud systems. IEEE Transactions on Computers **65**(5), 1440–1452 (May 2016). https://doi.org/10.1109/TC.2015.2435781
65. Zeng, D., Gu, L., Guo, S., Cheng, Z., Yu, S.: Joint optimization of task scheduling and image placement in fog computing supported software-defined embedded system. IEEE Transactions on Computers **65**(12), 3702–3712 (Dec 2016)

An Approach to Automatically Check the Compliance of Declarative Deployment Models

Christoph Krieger, Uwe Breitenbücher, Kálmán Képes, and Frank Leymann

Institute of Architecture of Application Systems, University of Stuttgart
Universitätsstr. 38, 70569 Stuttgart, Germany
`{firstname.lastname}@iaas.uni-stuttgart.de`

Abstract. The automation of application deployment has evolved into one of the most important issues in modern enterprise IT. Therefore, many deployment systems have been developed that process deployment models for automating the installation of systems. Creating such deployment models becomes more and more complex as compliance plays an increasingly important role. Not only external laws and regulations must be considered, but also a company's internal requirements must be fulfilled. However, this is a very complex challenge for the modelers as they require a firm knowledge of all the compliance rules that must be observed. As a result, this often leads to deployment models that violate compliance rules due to manual modeling mistakes or because of unawareness. In this paper, we introduce an approach that enables modeling of reusable Deployment Compliance Rules that can be executed automatically to check such regulations in declarative deployment models at design time. We validate our approach with a prototype based on the TOSCA standard and the OpenTOSCA ecosystem.

Keywords: Cloud Computing, Compliance, Deployment Modeling.

1 Introduction

There are many laws, regulations, and guidelines that influence an enterprise's information technology (IT), such as the General Data Protection Regulation [1], the ISO 2018 standard [2], or company internal regulations. Threats to security through the use of outdated software versions or financial risks due to the use of unlicensed software also have an impact on IT landscapes. Moreover, modern application landscapes often consist of complex composite applications and different heterogeneous systems [3]. Such large landscapes and systems present a challenging and expensive task to be maintained and managed manually. To decrease the costs and to minimize human errors [4], the automation of deployment and provisioning of applications has become an important subject in academia and industry. Approaches for deployment automation systems, such as Chef [5], Juju [6], Ansible [7], and Kubernetes [8], consume deployment models that describe the desired deployment and automatically execute all required tasks. However, enterprise's applications and IT landscapes are subject to a magnitude of requirements. If a company has a vast amount of requirements, it would require great effort and expertise of modelers of deployment models to know all rules and regulations that concern a particular application deployment. Moreover, rules are subject to change as new requirements and needs arise, and others are suspended. Not being compliant to all necessary rules can quickly result in serious consequences for companies, such as lawsuits due to unlicensed software or even the loss of customers due to leaks of personal data and subsequent trust issues by the customers. Thus, checking deployment models for compliance with a company`s requirements is of vital importance, but practically not possible if performed manually by modelers and operators.

In this paper, we present an approach that automates compliance checking of declarative deployment models to ensure that new or updated deployment models are always compliant with a company's set of constraints. To achieve this, we present an approach to specify reusable Deployment Compliance Rules that enables automated compliance checking of declarative deployment models at design time, which decreases the chance of application deployments that are not compliant to a company's regulations. Furthermore, the approach separates the modeling of compliance rules from the modeling of deployment models. This ensures that modelers do not need to know all constraints and requirements to specify compliant deployment models.

The remainder of this paper is structured as follows: Section 2 motivates our approach, followed by Section 3 which presents the main concept. Section 4 provides a formal model for the approach. Section 5 describes the prototypical implementation. In Section 6, we discuss related work. Finally, Section 7 concludes the paper and describes future work.

2 Motivation Scenario

Declarative deployment models describe the structure of an application to be deployed by specifying the required components, their relationships, as well as all artifacts required for the deployment, thus, the *topology* of the application [9]. A topology is a directed, weighted, and possibly disconnected graph consisting of nodes representing the components of the application and edges representing the relationships between them. Types associated with the components and relations specify the desired semantics. In addition, attributes associated with components or relations represent properties, such as ports or network addresses for servers. Figure 1 illustrates a topology describing the deployment of an application. The left-hand side shows a stack consisting of a component of type *Web-application* which is hosted on a Tomcat server of type *Tomcat8.5.23* as indicated by the *hostedOn* relation between the two components. The Tomcat server is hosted on a component of type *Ubuntu16.04VM*, which is hosted on a component of type *AmazonEC2*. The web application is connected to a component of type *MySQLDB5.0*, which provides persistent data storage. The attribute with key *DataType* and value *PersonalData* associated to the component indicates the type of data stored in this database. The database is managed through a Database Management System (DBMS) of type *MySQLDBMS5.0* that is also hosted on a component of type *Ubuntu16.04VM*. However, this Ubuntu VM is hosted on an OpenStack cloud with a specific IP address, as indicated by the IP attribute associated with the component. Thus, this deployment scenario describes a hybrid cloud application deployment which is partly hosted in a private cloud (OpenStack) and partly in a public cloud (Amazon EC2) [10]. In our scenario, the web application is a stateless component hosted on Amazon EC2 to be scaled out automatically based on the workload. The web application retrieves data from the MySQL database, performs processing tasks, and stores the results back to the database.

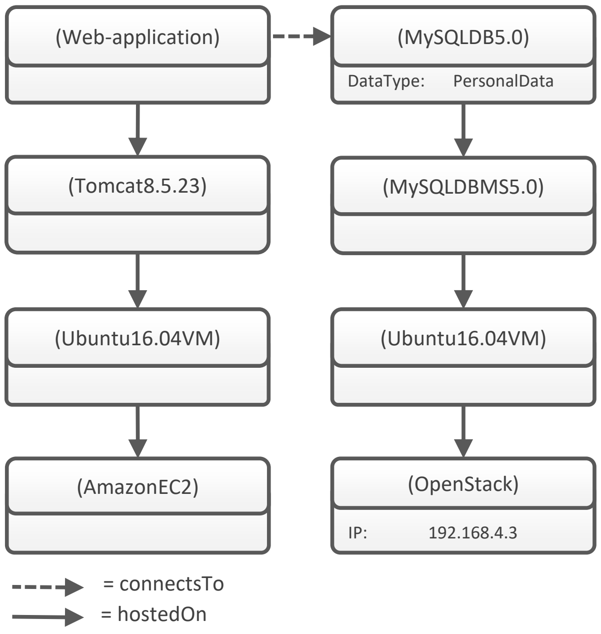


Fig. 1. A declarative deployment model describing the deployment of an application

Deployment models can be subject to a variety of constraints and rules which must be adhered to. However, it is difficult for modelers to be aware of all regulations, which quickly results in non-compliant deployment models that violate a company's rules. For the presented deployment model, there may be several rules that must be adhered to. In the scenario, the database contains personal data, i.e., customer names, which is sensitive data. A company could decide that all personal data must only be stored in a specific private cloud. The application uses components that should receive regular updates since old versions of software artifacts often present security risks. In our scenario, this is relevant for the MySQL database and management system, the Tomcat server, and the Ubuntu VMs. For example, an outdated Tomcat could expose vulnerabilities that have been fixed by later versions, such as remote code execution. In this paper, we present an approach to express such regulations as reusable compliance rules that can be automatically checked at design time.

3 Concept to Automatically Check the Compliance of Declarative Deployment Models

An overview of the included roles, components, and tasks of the approach to automatically check the compliance of declarative deployment models is shown in Figure 2. There, the *Compliant Deployment Modeling System* is divided into the two separate areas *Compliance Modeling* and *Deployment Modeling*. The system separates concerns, as the expertise of compliance experts and deployment experts is integrated in an automated fashion without the need to exchange knowledge between the involved roles.

The left-hand side of Figure 2 shows the involved roles, components, and tasks of the Compliance Modeling area. Compliance experts use *the Compliance Modeling Tool* for the definition and maintenance of *Deployment Compliance Rules*, which formally describe and capture all regulations that must be fulfilled by deployment models. These rules are stored persistently in a *Compliance Rule Repository* within the Compliance Modeling Tool. The stored rules provide a means to detect potentially relevant areas of deployment models and check their compliance by comparing them to a compliant fragment. A respective method will be elaborated in the next sections.

Deployment Modeling is concerned with the definition and maintenance of deployment models. The right-hand side of Figure 2 shows the involved roles, components, and tasks of the Deployment Modeling area. Deployment experts use the *Deployment Modeling Tool* to define and maintain deployment models that are stored persistently in a *Deployment Model Repository*. The deployment experts do not have to be aware of all regulations concerning deployment models since the Deployment Modeling Tool uses the *Compliance Checker* to verify the compliance of declarative deployment models based on the stored compliance rules. This ensures that only compliant deployment models are stored in the repository.

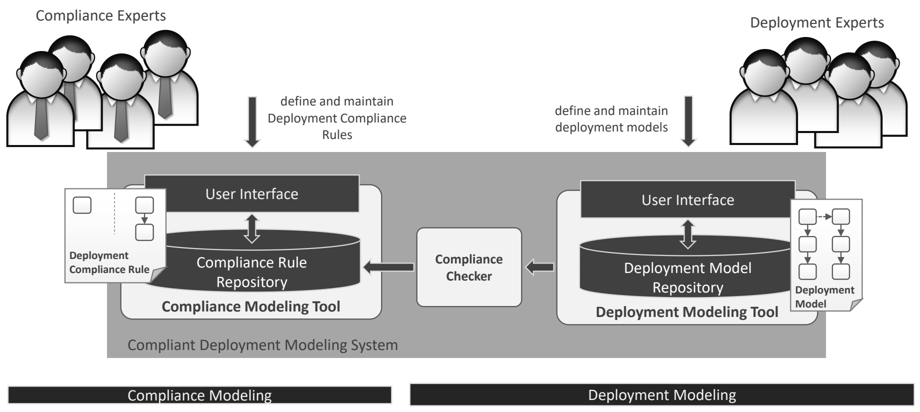

Compliance Experts
define and maintain Deployment Compliance Rules
define and maintain deployment models
Deployment Experts
User Interface
Deployment Compliance Rule
Compliance Rule Repository
Compliance Modeling Tool
Compliance Checker
User Interface
Deployment Model Repository
Deployment Modeling Tool
Deployment Model
Compliant Deployment Modeling System
Compliance Modeling
Deployment Modeling

Fig. 2. Overview of roles, components, and tasks of the approach

4 Metamodel and Formalization

In our previous work [11], we have conceptually introduced Deployment Compliance Rules. In the context of this paper, we will present a formal metamodel for the definition of such Deployment Compliance Rules. For this, in Section 4.1, we first provide a formal metamodel for declarative deployment models, which is based on topologies. Section 4.2 provides the metamodel of Deplyoment Compliance Rules, while Section 4.3 describes the algorithm used for automated compliance checking.

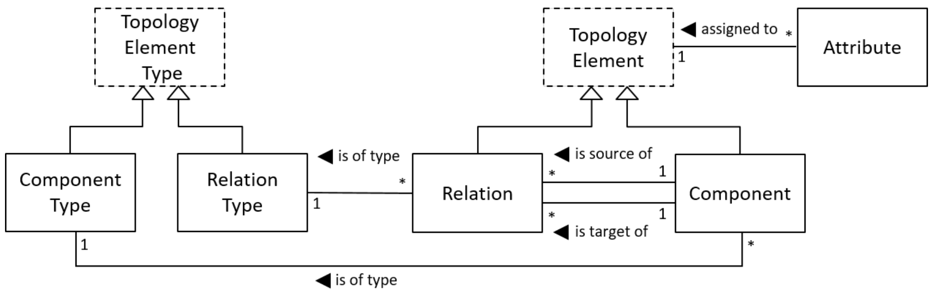


Fig. 3. Metamodel of topologies

4.1 Basic Metamodel of Topologies

The metamodel of topologies, is based on the Declarative Application Management Modeling and Notation (DMMN) introduced by Breitenbücher [12] and further abstracted by Saatkamp et al. [13]. Figure 3 gives an overview of the metamodel. There, class names contain a starting capital letter.

Let T be the set of all Topologies, then $t \in T$ is defined as an eight-tuple:

$$t = (C_t, R_t, CT_t, RT_t, A_t, type_t, attributes_t, supertype_t) \quad (1)$$

The elements of t are defined as follows:

- C_t: The set of Components in t, whereby each $c_i \in C_t$ represents a component of the application to be deployed.
- $R_t \subseteq C_t \times C_t$: The set of Relations in t, whereby each $r_i = (c_s, c_t) \in R_t$ represents the relationship between two components, where c_s is the source and c_t is the target of the relationship.
- CT_t: The set of Component Types in t, whereby each $ct_i \in CT_t$ describes the semantics for the Components that have this Component Type assigned.
- RT_t: The set of Relation Types in t, whereby each $rt_t \in RT_t$ describes the semantics for the Relations that have this Relation Type assigned.
- $A_t \subseteq \Sigma^+ \times \Sigma^+ \times \Sigma^+$: The set of Attributes in t, whereby each $a_i = (Id,\ Key,\ Value) \in A_t$ describes a property of a Component or Relation with a key and a value. Each Id must be unique within a Topology. Id, Key and $Value \in \Sigma^+$ are typically strings.

- $type_t$: The mapping that assigns all Relations and Components in t to their Relation Type or Component Type. Let the set of Topology Elements $E_t \coloneqq C_t \cup R_t$ be the union of the set of Components and the set of Relations of t, and the set of Topology Element Types $ET_t \coloneqq CT_t \cup RT_t$ be the union of the set of Component Types and the set of Relation Types of t. Then, the mapping $type_t$ associates each $e_i \in E_t$ with an $et_j \in ET_t$ to provide the semantics for each Topology Element.

$$type_t : E_t \rightarrow ET_t \tag{2}$$

- $attributes_t$: The mapping that assigns each Topology Element to a set of Attributes.

$$attributes_t : E_t \rightarrow \wp(A_t) \tag{3}$$

- $supertype_t$: The mapping that assigns Relation Types and Component Types to their respective supertype. Let ET_t be the set of Topology Element Types of t. Then, the mapping $supertype_t$ associates an $et_i \in ET_t$ with an $et_j \in ET_t$ with $i \neq j$. This means that et_j is the supertype of et_i.

$$supertype_t : ET_t \rightarrow ET_t \tag{4}$$

Additionally, we define the mapping $supertypes_t$ that maps a Topology Element of t to its respective Topology Element Type specified by $type_t$ combined with all transitively resolvable supertypes of $type_t$. Let E_t be the set of Topology Elements and ET_t be the set of Topology Element Types of t.

$$supertypes_t : E_t \rightarrow \wp(ET_t) \tag{5}$$

4.2 Metamodel of Deployment Compliance Rules

In the following, we will provide a formal metamodel for Deployment Compliance Rules. Additionally, we define the conditions under which such a rule is *valid*, *detected*, or *satisfied*. The concept of automated compliance checking based on Deployment Compliance Rules is illustrated in Figure 4. There, the left-hand side of the figure illustrates a declarative deployment model, the right-hand side illustrates a Deployment Compliance Rule. A Deployment Compliance Rule consists of two Topologies called Detector and Required Structure. The Detector is used to detect areas in the deployment model that are subject to the rule while the Required Structure is used to verify if the rule is satisfied, i.e., if the rule is fulfilled. The rule shown in Figure 4 is concerned with the storage of personal data in a specific private cloud as discussed in the motivation scenario presented in Section 2. It specifies that any database that stores personal data has to be managed by a database management system (DBMS).

The DBMS must in turn be hosted in a virtual machine provided by a specific OpenStack cloud.

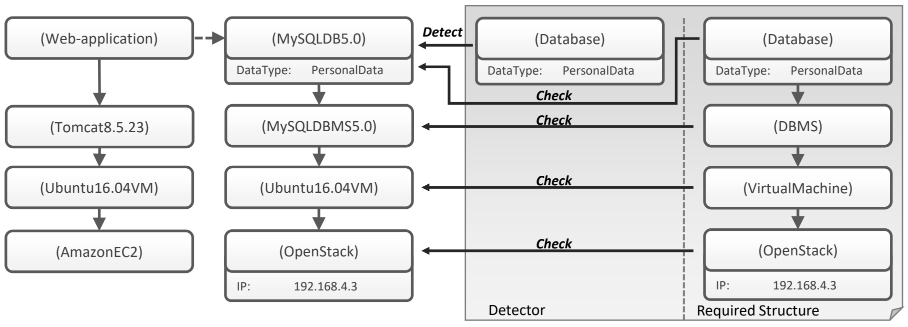


Fig. 4. Concept of detection and checking of Deployment Compliance Rules in declarative deployment models

Let $CR \subseteq T \times T$ be the set of all Deployment Compliance Rules, then $cr \in CR$ is defined as:

$$cr \in CR := (Detector,\ RequiredStructure) \tag{6}$$

The $Detector$ and $RequiredStructure$ are provided as a Topology, i.e., $Detector \in T$ and $RequiredStructure \in T$.

The detection and checking of Deployment Compliance Rules is based on the detection of subgraph isomorphisms that also considers the types and attributes of the Topology Elements that form the deployment model. Subgraph isomorphisms can be detected by using the VF2 algorithm described by Cordella et al. [14]. We omit the VF2 algorithm here and refer interested readers to the provided reference. The detection of relevant areas in a deployment model is formally the detection of subgraph isomorphisms. To detect semantically compatible subgraph isomorphisms, i.e. to find areas with the same structure, types, and attributes, we define a matching relation to decide if a Topology Element can be matched to another Topology Element. Let $e_i \in E_t$ and $e_j \in E_t$ with $i \neq j$ be two Topology Elements in t. Then the matching relation $\equiv_e$ is defined as follows:

$$e_1 \equiv_e e_2 :\Leftrightarrow \big(type_t(e_1) \in supertypes_t(e_2)\big) \wedge \Big(\forall a_i \in attributes_t(e_1) \exists a_j \in attributes_t(e_2) :\ \pi_2(a_i) \Leftrightarrow \pi_2\big(a_j\big) \wedge \pi_3(a_i) \Leftrightarrow \pi_3\big(a_j\big)\Big) \tag{7}$$

A Topology Element e_1 can be matched to a Topology Element e_2 if the type of e_1 is of the same type or supertype of the type of e_2. Furthermore, all attributes of e_1 must also be present in e_2 with equivalent Key and $Value$. For example, the Component with Component Type “Database” shown in Figure 4 can be matched to the Component with Component Type “MySQLDB5.0”. The “Database” Component has an attribute with Key = “$DataType$” and $Value$ = “PersonalData”. The equivalent attribute, i.e., same Key and $Value$, is also assigned to the “MySQLDB5.0” Component. In addition, the Component Type “Database” is a supertype of the Component Type “MySQLDB5.0”.

Let E be the set of all Topology Elements. Then $m \in M, m \subseteq E \times E$ is defined as a bijective mapping that maps a Topology Element $e_i \in E$ to another Topology Element $e_j \in E$ with $i \neq j$:

$$M := \wp(E \times E) \setminus \emptyset \tag{8}$$

A matching mapping is a mapping between two Topologies that preserves the structure and semantics of the Topologies and the individual Topology Elements by also considering the matching relation $\equiv_e$. A mapping $m \in M$ is a matching mapping from $t_1 \in T$ to $t_2 \in T$ if t_1, t_2, and m fulfill the relation $matching \subseteq T \times T \times M$:

$$(t_1, t_2, m_1) \in matching :\Leftrightarrow \left(m_1 \subseteq E_{t_1} \times E_{t_2}\right) \wedge \left(\forall c_i \in C_{t_1} \exists!\, c_j \in C_{t_2} \colon (c_i, c_j) \in m_1 \wedge c_i \equiv_e c_j\right) \wedge \left(\forall r_k \in R_{t_1} \exists!\, r_l \in R_{t_2} : (r_k, r_l) \in m_1 \wedge r_k \equiv_e r_l \wedge \left(\pi_1(r_k), \pi_1(r_l)\right) \in m_1 \wedge \left(\pi_2(r_k), \pi_2(r_l)\right) \in m_1\right) \tag{9}$$

The mapping m_1 represents a subgraph isomorphism from t_1 to t_2 that also considers the matching relation $\equiv_e$. Each Component in t_1 is mapped to exactly one matching Component in t_2, to which no other Component in t_1 has been mapped to. Analogously each Relation in t_1 is mapped to exactly one matching Relation in t_2 with the addition that the relations sources and targets have also been mapped to each other and therefore the mapping preserves the structure of the topologies. For example, in Figure 4, the Required Structure can be matched completely to the right-hand stack of the Topology since all indicated component pairs fulfill the matching relation $\equiv_e$ and both stacks are structurally identical.

A Deployment Compliance Rule cr_1 is *valid* if there is exactly one matching mapping from the Detector to the Required Structure and the following holds:

$$\exists!\, (\pi_1(cr_1), \pi_2(cr_1), m) \in matching \tag{10}$$

This ensures that each matching mapping found for the Required Structure corresponds to a matching mapping found for the Detector.

A Deployment Compliance Rule cr_1 is *detected* in a Topology t_1 if there is at least one matching mapping from the rule’s Detector $\pi_1(cr_1)$ to the Topology:

$$\exists (\pi_1(cr_1), t_1, m) \in matching \tag{11}$$

For a Deployment Compliance Rule cr_1 to be *satisfied* for a Topology t_1, there has to be exactly one corresponding matching mapping from the Required Structure $\pi_2(cr_1)\ to\ t_1$ for each matching mapping form the Detector $\pi_1(cr_1)\ to\ t_1$:

$$\forall (\pi_1(cr_1), t_1, m_i) \in matching\ \exists!\, \left(\pi_2(cr_1), t_1, m_j\right) \in matching : \left(\forall e_k \in \pi_2(m_i) : e_k \in \pi_2\left(m_j\right)\right) \tag{12}$$

Each area that has been found in t_1 by mapping the Detector to t_1 also satisfies the Required Structure and therefore, the Deployment Compliance Rule cr_1 is *satisfied*, i.e., the rule is fulfilled. For example, the rule in Figure 4 is *detected* in the declarative deployment model as there is a matching mapping from the rule's Detector to the Topology of the deployment model. Since there exists exactly one corresponding matching mapping from the Required Structure to the Topology for each matching mapping from the Detector to the Topology, the rule is also *satisfied*.

4.3 Compliance Checking Algorithm

Based on the presented metamodel, an algorithm is proposed to automatically check a Topology for a given Deployment Compliance Rule. This algorithm is described in Algorithm 1 in pseudocode. It is executed once for every Deployment Compliance Rule that is defined for a Topology. The algorithm uses a Deployment Compliance Rule and a Topology as inputs. The result is a set containing matching mappings of the Detector to the Topology to indicate rule violations. If the set is empty, the given Deployment Compliance Rule is satisfied for that Topology. In the case, that a Deployment Compliance Rule is not detected for a Topology, the algorithm also returns an empty set indicating that the rule is satisfied. The given Topology is analyzed for matches to both the Detector (line 3) and the Required Structure. Subsequent, all mappings found for the Detector are checked for a corresponding mapping for the Required Structure (line 5). If none is found, the Detector's mapping is added to the result set R (line 6). At the end of the algorithm, $Result$ contains mappings to areas where the given Deployment Compliance Rule is violated. If $Result$ is empty, the rule is satisfied for the given Topology.

```
procedure: $findViolations(cr \in \mathcal{CR},\ t \in \mathcal{T})$
1: $Result := \emptyset$
2: $//find\ all\ matching\ mappings\ for\ the\ Detector\ of\ cr$
3: for all $((\pi_1(cr_1), t, m_i) \in matching)$ do
4:    $//find\ a\ matching\ mapping\ for\ the\ Required\ Structure\ of\ cr$
5:    if $\nexists(\pi_2(cr_1), t, m_j) \in matching : (\forall e_k \in \pi_2(m_i) : e_k \in \pi_2(m_j))$ then
6:       $Result := Result \cup m_j$
7:    end if
8: end for
9: return $Result$
```

Algorithm 1. Pseudocode to identify Deployment Compliance Rule violations in a Topology

5 Prototypical Implementation

The prototype described in this section implements the metamodel for Topologies and Deployment Compliance Rules as well as the algorithm introduced in Section 4. Furthermore, the prototype enables compliance experts and deployment experts to model Deployment Compliance Rules, Topologies, Component Types, and Relation Types with a graphical user interface while also providing the functionality to check the Topologies for compliance.

The prototype consists of a graphical user interface, a compliance checker, a model repository, and an application programmer interface (API). In our prototype, we combine the Compliance Checker, the Compliance Modeling Tool, and the Deployment Modeling Tool introduced in Section 3 to one component, the Modeling Tool. The graphical user interface enables modelers to create and update deployment models as well as Deployment Compliance Rules. The user interface also enables the definition of reusable component and relation types and their hierarchical structure. The compliance checker provides the functionality to validate Deployment Compliance Rules and to check the compliance of deployment models. For that purpose, the compliance checker has access to the model repository. The model repository is a combination of the Compliance Rule Repository and the Deployment Rule Repository introduced in Section 3. It is used to persistently store deployment models, Deployment Compliance Rules, component types, and relation types. The API of the modeling tool provides external access to the functionality of the modeling tool, such as the storing and checking of deployment models.

Since our prototype is based on TOSCA and the OpenTOSCA ecosystem [15], we briefly introduce the TOSCA standard and provide a mapping from the formal metamodel to TOSCA. TOSCA is an OASIS standard that enables the specification of cloud applications by defining their structure and their orchestration. We only introduce constructs in TOSCA that are necessary for our method and refer interested readers to the specification [16], the simple profile [17], and the primer [18]. A TOSCA *Topology Template* represents the structure of an application. It specifies all components needed for the deployment of an application as well as the relationships between them. The *Topology Template* is a directed graph with typed nodes and weighted edges and corresponds to a Topology defined in the metamodel. *Node Templates* and *Relationship Templates* represent the Components and Relations of the metamodel. Component Types and Relation Types can be mapped to *Node Types* and *Relationship Types* of TOSCA. The attributes defined by the metamodel can be mapped to *Properties*. In TOSCA, Node Types can be specified with a *PropertiesDefinition* that defines the structure of possible properties for the Node Templates. The actual attribute values are specified by Node Templates. Since TOSCA allows extensions to the specification, we introduce the new element *ComplianceRule* that has exactly two Topology Templates as elements: *Detector* and *RequiredStructure*. Therefore, we have a complete mapping from the metamodel to TOSCA.

The prototypical implementation extends Winery [19][1], which is a graphical modeling tool for modeling and managing applications using TOSCA. Winery already provides a mechanism for persistent storage of TOSCA elements which was extended

[1] http://eclipse.github.io/winery

to also store Deployment Compliance Rules. Additionally, we added the new component *Compliance Checker* to Winery which realizes the concepts presented in this paper. To associate TOSCA deployment models with Deployment Compliance Rules, we use the concept of namespaces, i.e., all Topology Templates must be checked for all Deployment Compliance Rules that are in the same namespace. Therefore, each rule defined for a certain namespace automatically applies to all Topology Templates defined in that namespace.

6 Related Work

In this section, we discuss and compare works that are related to our method. Software architecture is often described with 5 views first described by Kruchten [20]. However, there can still be other views on the architecture of a software system, such as the view on the deployment model of a system. Software architecture erosion [21] describes the deviation of actual software artifacts from their architecture that manifests over time. It has been addressed with various methods such as model-driven approaches, through dependency analysis, or through checking mechanisms, such as the reflexion models approach by Murphy et al. [22]. They describe high-level models, i.e. boxes and arrows that are used by software architects to describe and reason about the architecture of a software artifact. To test the compliance with the architecture, they generate a low-level model from existing artifacts through the use of code-analysis and compare the two models to find convergences, divergences, and absences, e.g., correct, additional, and missing connections between components. Koschke and Simon extended this approach by introducing hierarchical reflexion models [23]. The method uses an architecture model to express implicit requirements and constraints for the resulting software artifact. With our method, we provide reusable rules, that can be applied to deployment models in general. Other approaches to control software architectures include Architecture Description Languages [24] to describe architectures or Architecture Constraint Languages [25] to express constraints. Deiters et al. [26] introduce Rule-Based Architectural Compliance Checks for Enterprise Architecture Management that are expressed as queries. However, these approaches for describing constraints to check the compliance of software artifacts to their intended architecture are application specific and on the level of artifacts. With our method, we enable to specify compliance rules on the level of deployment models in a very generic manner. Hence, they can be used to check the deployment models of different applications.

There are many works in the area of business process verification to ensure that business process models adhere to regulations. Due to the great number of works we refer interested readers to a survey on business process compliance by Fellmann and Zasada [27]. Schleicher et al. [28] introduce an approach to express control-flow and data-related compliance rules for business processes while Koetter et al. [29] introduce a generic Compliance Descriptor that links compliance rules to their source. Tran et al. [30] introduce an approach that enables compliance modeling for service-oriented systems through the use of domain-specific languages to model compliance for different areas. Liu et al. [31] propose to separate the modeling of business processes from the modeling of compliance rules. In their method, they use the Business Process Execution Language (BPEL) [32] as a model for business processes and the Business Property Specification Language (BPSL) [33] to specify their compliance rules. They use established model checkers based on linear temporal logic to verify process models with their rules. These compliance rules deal mostly with temporal aspects in the execution of business processes while our method provides a compliance checking mechanism for deployment models that can be used to address issues, such as outdated software versions or structural properties, i.e., how components may be hosted under certain circumstances.

7 Conclusion & Future Work

In this paper, we presented Deployment Compliance Rules that enable to describe restrictions, constraints, and requirements for declarative deployment models in a reusable manner. We provided a formalized model for Deployment Compliance Rules and an algorithm to automatically check if a deployment model violates a given rule. Based on this, we presented an approach to ensure that created or updated deployment models are compliant to the current set of Deployment Compliance Rules. Further, the approach allows separating the definition of Deployment Compliance Rules from the creation and maintenance of deployment models. A validation of the approach is provided via the implementation of a TOSCA-based prototype although the provided formalization enables to apply the approach to any graph-based and declarative deployment model language. In the future, we plan to extend the approach to also verify the consistency of the rule repository through detection of conflicting rules within the repository.

Acknowledgments. This work was partially funded by the German Research Foundation (DFG) project ADDCompliance (636503) and the BMWi project SmartOrchestra (01MD16001F).

References

1. General Data Protection Regulation, https://eur-lex.europa.eu/legal-content/EN/TXT/?qid=1532348683434&uri=CELEX:02016R0679-20160504.
2. ISO/IEC 27018:2014 Code of practice for protection of personally identifiable information (PII) in public clouds acting as PII processors. International Organization for Standardization (2014).
3. Breitenbücher, U., Binz, T., Kopp, O., Leymann, F., Wettinger, J.: Integrated Cloud Application Provisioning: Interconnecting Service-Centric and Script-Centric Management Technologies. In: On the Move to Meaningful Internet Systems: OTM 2013 Conferences (CoopIS 2013). pp. 130–148. Springer (2013).
4. Oppenheimer, D., Ganapathi, A., Patterson, D.A.: Why do internet services fail, and what can be done about it? In: Proceedings of the 4th Conference on USENIX Symposium on Internet Technologies and Systems (USITS 2003). USENIX (2003).

5. Opscode, Inc.: Chef Official Site. (2017).
6. Canonical Group Ltd(GB): Juju Official Site. (2017).
7. Mohaan, M., Raithatha, R.: Learning Ansible. Packt Publishing (2014).
8. Kubernetes, https://kubernetes.io/.
9. Breitenbücher, U., Képes, K., Frank, L., Wurster, M.: Declarative vs. Imperative: How to Model the Automated Deployment of IoT Applications? (2017).
10. Fehling, C., Leymann, F., Retter, R., Schupeck, W., Arbitter, P.: Cloud Computing Patterns: Fundamentals to Design, Build, and Manage Cloud Applications. Springer (2014).
11. Fischer, M.P., Breitenbücher, U., Képes, K., Leymann, F.: Towards an Approach for Automatically Checking Compliance Rules in Deployment Models. In: Proceedings of The Eleventh International Conference on Emerging Security Information, Systems and Technologies (SECURWARE). pp. 150–153. Xpert Publishing Services (XPS) (2017).
12. Breitenbücher, U.: Eine musterbasierte Methode zur Automatisierung des Anwendungsmanagements, (2016).
13. Saatkamp, K., Breitenbücher, U., Kopp, O., Leymann, F.: Topology Splitting and Matching for Multi-Cloud Deployments. In: Proceedings of the 7th International Conference on Cloud Computing and Services Science (CLOSER 2017). pp. 247–258. SciTePress (2017).
14. Cordella, L.P., Foggia, P., Sansone, C., Vento, M.: A (sub) graph isomorphism algorithm for matching large graphs. IEEE Transactions on Pattern Analysis and Machine Intelligence. 26, 1367–1372 (2004).
15. Binz, T., Breitenbücher, U., Haupt, F., Kopp, O., Leymann, F., Nowak, A., Wagner, S.: OpenTOSCA - A Runtime for TOSCA-based Cloud Applications. In: Proceedings of the 11th International Conference on Service-Oriented Computing (ICSOC 2013). pp. 692–695. Springer (2013).
16. OASIS: Topology and Orchestration Specification for Cloud Applications (TOSCA) Version 1.0. Organization for the Advancement of Structured Information Standards (OASIS) (2013).
17. OASIS: TOSCA Simple Profile in YAML Version 1.0. Organization for the Advancement of Structured Information Standards (OASIS) (2015).
18. OASIS: Topology and Orchestration Specification for Cloud Applications (TOSCA) Primer Version 1.0. Organization for the Advancement of Structured Information Standards (OASIS) (2013).
19. Kopp, O., Binz, T., Breitenbücher, U., Leymann, F.: Winery – A Modeling Tool for TOSCA-based Cloud Applications. In: Proceedings of the 11th International Conference on Service-Oriented Computing (ICSOC 2013). pp. 700–704. Springer (2013).
20. Kruchten, P.B.: The 4+1 view model of architecture. IEEE software. 12, 42–50 (1995).
21. De Silva, L., Balasubramaniam, D.: Controlling software architecture erosion: A survey. Journal of Systems and Software. 85, 132–151 (2012).
22. Murphy, G.C., Notkin, D., Sullivan, K.J.: Software reflexion models: Bridging the gap between design and implementation. IEEE Transactions on Software Engineering. (2001).
23. Koschke, R., Simon, D.: Hierarchical Reflexion Models. In: WCRE. pp. 186–208 (2003).
24. Medvidovic, N., Taylor, R.N.: A classification and comparison framework for software architecture description languages. IEEE Transactions on software engineering. 26, 70–93 (2000).

25. Tibermacine, C., Fleurquin, R., Sadou, S.: A family of languages for architecture constraint specification. Journal of Systems and Software. 83, 815–831 (2010).
26. Deiters, C., Dohrmann, P., Herold, S., Rausch, A.: Rule-based architectural compliance checks for enterprise architecture management. In: Enterprise Distributed Object Computing Conference, 2009. EDOC'09. IEEE International. pp. 183–192. IEEE (2009).
27. Fellmann, M., Zasada, A.: State-of-the-art of business process compliance approaches. (2014).
28. Schleicher, D., Grohe, S., Leymann, F., Schneider, P., Schumm, D., Wolf, T.: An approach to combine data-related and control-flow-related compliance rules. In: Service-Oriented Computing and Applications (SOCA), 2011 IEEE International Conference on. pp. 1–8. IEEE (2011).
29. Koetter, F., Kochanowski, M., Weisbecker, A., Fehling, C., Leymann, F.: Integrating compliance requirements across business and it. In: Enterprise Distributed Object Computing Conference (EDOC), 2014 IEEE 18th International. pp. 218–225. IEEE (2014).
30. Tran, H., Zdun, U., Oberortner, E., Mulo, E., Dustdar, S., others: Compliance in service-oriented architectures: A model-driven and view-based approach. Information and Software Technology. 54, 531–552 (2012).
31. Liu, Y., Muller, S., Xu, K.: A static compliance-checking framework for business process models. IBM Systems Journal. 46, 335–361 (2007).
32. OASIS: Web Services Business Process Execution Language (WS-BPEL) Version 2.0. Organization for the Advancement of Structured Information Standards (OASIS) (2007).
33. Xu, K., Liu, Y., Wu, C.: Bpsl modeler–visual notation language for intuitive business property reasoning. Electronic Notes in Theoretical Computer Science. 211, 211–220 (2008).

Towards Pattern-based Rewrite and Refinement of Application Architectures

Jasmin Guth and Frank Leymann

Institute of Architecture of Application Systems, University of Stuttgart
Universitätsstr. 38, 70569 Stuttgart, Germany
[lastname]@iaas.uni-stuttgart.de

Abstract. With the ongoing growth of IT application systems, the development and modeling process of their architectures becomes increasingly complex. Architectural patterns capturing proven solutions for recurring problems in an abstract and human readable way should support this process. Due to the abstract character of patterns, they cannot be applied to a concrete architecture automatically: For each use case, patterns have to be read, understood, adapted to the respective use case, and realized manually. In this work, we tackle these issues by proposing an approach for an automated realization of architectural patterns within a given architectural graph based on graph transformation techniques.

Keywords: Application Architectures, Patterns, Solution Paths, Rewrite, Refinement, Graph Transformation.

1 Introduction

With the ongoing growth of IT application systems, the manual modeling and development phase of their architectures becomes increasingly complex. Diverse modeling tools are available to support this process, whereby each tool employs a different format and, hence, the graphical representation, if available, differs as well. This also affects the granularity of details depicted, like, e.g., abstract components are depicted or even implementation details are given. To enable a classification of architectures, Malan and Bredemeyer [1] delimit between three levels: (i) Conceptual architectures which are abstract, (ii) logical architectures which are detailed, and (iii) execution architectures which describe the process and deployment view. Further, each level can be subdivided into a behavioral and structural view. Since an application system architecture has significant impact on the prospective usability, performance, and maintainability, its development phase is of explicit importance [2]. To ease this process and to support the developer, diverse pattern languages can be used [3]. Patterns describe proven solutions for recurring problems documented in an abstract and human readable way [4]. Due to the abstract character of patterns, they cannot be applied to an architecture directly: Patterns have to be read, understood, adapted, and realized manually for each use case [4]. However, there are many pattern languages available, but a general approach to automatically apply patterns to architectures to refine and model the architecture by selecting patterns is still missing. Hence, each pattern is realized over and over again manually, which leads to a multitude of possibly incorrect pattern realizations.

The result of the discussion above is that patterns are a profound support during the development and modeling phase of an architecture, missing a tool and method integration. The manual modeling and development of application system architectures is already a time consuming and error prone task. The lack of pattern integration leads to an amplification of those negative aspects and a multitude of possibly incorrect pattern realizations. Hence, the usage of patterns during the modeling phase, which should actually support and ease this phase, impedes it and makes it even more complex.

In this paper, we tackle these issues by introducing an approach to enable an automated pattern integration within the modeling and development process of the structural view of application system architectures on a conceptual level. In general, this leads to an adapted modeling process: (i) Abstract architectures can be refined by applying patterns, i.e., corresponding components and connectors are added which results in a more concrete architecture [5], and (ii) existing architectures can be rewritten based on applied patterns, i.e., components and connectors get exchanged or deleted [6].

Following, we will combine (i) the knowledge of proven solutions for recurring problems in terms of patterns, (ii) the modeling and development process of architectures, and (iii) architectural effects of patterns if applied onto architectures.

The remainder of this paper is structured as follows: Fundamentals to ease the understanding are presented in Section 2. We introduce the concept of our approach in Section 3 and work related to our approach is discussed in Section 4. We conclude this work and give a short overview of future work in Section 5.

2 Fundamentals

Within this section we give an overview of fundamentals to ease the understanding of our approach. First, background of application architectures and architectural graphs that we use to represent an application system architecture is given. Then fundamentals of patterns, pattern languages, and solution paths are presented.

2.1 Application System Architectures & Architectural Graphs

The structure of an IT system is described by its architecture, depicting the system's components and their relationships, as well as their external visible properties [7]. Thus, an IT architecture describes the composition of architectural elements without implementation details. For the documentation and visualization of an architecture, multiple architecture description languages, modeling languages, and modeling tools are present, such as ACME[1], ArchiMate[2], or particular UML[3] diagrams. Since each language and tool differs within their capabilities and, hence, their representation range, and due

[1] http://www.cs.cmu.edu/~acme/
[2] http://www3.opengroup.org/subjectareas/enterprise/archimate-overview
[3] http://www.uml.org

to the fact, that in most cases diverse people are involved, like, e.g., business process managers and programmers, which develop and discuss the architecture, the level of details within an architecture differs [1]. It has become common practice to use drawings of boxes to represent components and lines representing relationships [8]. This informal proceeding adapts Le Métayer [9] and describes a representation and formalization of software architectures as graphs, in which nodes represent components and edges represent relationships among them. Within this work, we use such a graph representation of architectures, referred to as an architectural graph to formalize an application system architecture. Those architectural graphs represent structural architectures on the conceptual level [1]. **Fig. 1** shows an exemplary architectural graph, whereby nodes represent the components of the architecture and edges represent the connectors among the components. Furthermore, each node is mapped to a type, such as application, server, or virtual machine. This mapping is required for a verification if the corresponding pattern is applicable to the architectural graph, since a specific pattern cannot be applied to all kind of architectural graphs due to possibly required components and relationships, a detailed description follows in Sect. 3.3.

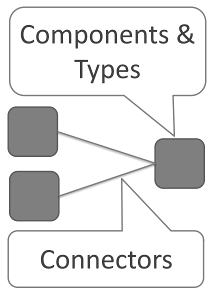


Fig. 1. Architectural Graphs

2.2 Patterns, Pattern Languages & Solution Paths

Patterns describe proven solutions for recurring problems within a certain context in an abstract and human readable way [4]. For example, a pattern may provide an abstract solution for common problems on how to design an application architecture. Patterns abstractly document an approach on how to solve a certain problem, since they are independent of the underlying technologies, such as specific runtime environments or programming languages [4]. Typically, concrete implementation realizations, such as code snippets, are not documented. Hence, patterns need to be adapted to the respective use case, and can, therefore, be used within various kinds of IT environments [4].

A pattern language comprises a collection of related patterns, forming a network as depicted in **Fig. 2**, in which one can navigate from one pattern to another related one that might become relevant after the application of the first pattern [3,10]. Several pattern languages are available in the field of IT, whereby each language has a different focus. For example, Gamma et al. [11] published design patterns focusing on object-oriented software, Fowler [12] introduced patterns for the development of enterprise application systems, and Fehling et al. [13] published patterns which focus on cloud computing and its architectures. Besides such general pattern languages, several patterns are available, published and realized by platform providers, such as the AWS Cloud Design Patterns [14]. Such provider-specific patterns are excluded from this work since they are bound to the provider and implement provider-specific realizations.

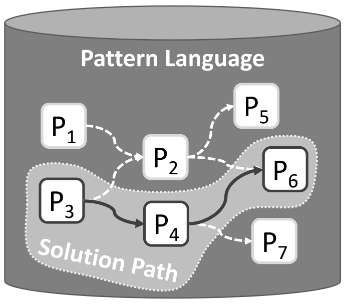


Fig. 2. Pattern Language and Exemplary Solution Path [15]

As described above, a pattern language forms a network of patterns, which connects related patterns [3,10]. After an entry point, i.e., a pattern which solves the problem at least partially, to this network is found and selected, subsequent and related patterns can be navigated to and selected as well [15]. Each such a possible path of selected patterns within this network forms a solution path [15,16], one exemplary solution path is shown in **Fig. 2**. Within our approach, we use solution paths to define the application order of patterns to architectural graphs, since solution paths are directed paths.

3 Concept of Pattern-based Rewrite & Refinement of Application Architectures

The aim of our approach is to enable a pattern-based rewrite and refinement of architectures through an automated application of patterns. Therefore, architectures are represented as architectural graphs as described in Sect. 2.1. To define the application order of patterns, we use the selected solution path, as described in Sect. 2.2. **Fig. 3** gives an overview of our approach. On the upper left side, the input is shown, i.e., the basis architectural graph and a selected solution path, which is described in Sect. 3.1. The remaining steps are iterative, since each pattern of the selected solution path is applied individually: First, the pattern which is applied next is defined, as described in Sect. 3.2.

Then the requirements of this pattern are checked, as explained in Sect. 3.3. In the last step, the architectural graph is rewritten or refined, as described in Sect. 3.4.

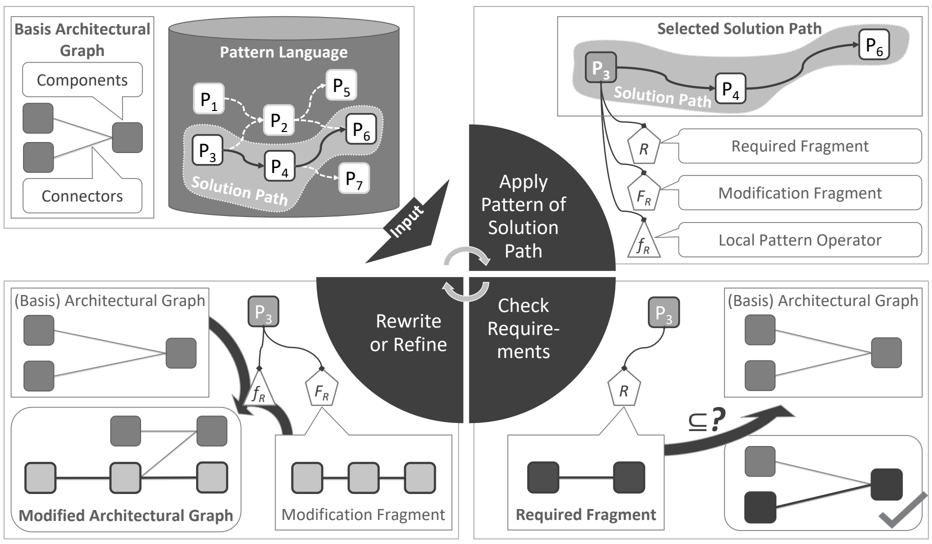


Fig. 3. Concept of Pattern-based Rewrite and Refinement of Application Architectures

3.1 Input

A basis architectural graph and a selected solution path of a specific pattern language forms the input of our iterative approach, as shown on the top left of **Fig. 3**. The basis architectural graph is a possibly abstract architecture represented as a graph, as described within Sect. 2.1, on which the modifications resulting of the application of a pattern are performed. For instance, a client server architecture could be such a basis architectural graph with three nodes, i.e., two nodes of type client and another node of type server, as well as two edges connecting both client components with the server component. Furthermore, the selected solution path, as described within Sect. 2.2, forms the second part of the input. The selected solution path comprises all patterns that have to be applied, and the order of the directed solution path defines the application order in which the patterns are applied to the basis architectural graph. For each pattern of the solution path all following described three steps are performed. The resulting architectural graph of one iteration serves as an input for the next iteration. Within the first iteration the basis architectural graph of the input is used to operate on.

3.2 Apply Pattern of Solution Path

Within the step *Apply Pattern of Solution Path*, as shown on the top right-hand side of **Fig. 3**, the next pattern of the solution path, as defined within the input, is taken to be applied to the (basis) architectural graph. The solution path defines the application order, i.e., starting with the first pattern P_3 of the solution path, within the next iteration the second pattern P_4 gets applied, etc. To achieve a proper modification of the architectural graph, each pattern consists of three attachments: (i) R - The *required fragment*, which describes a possibly empty subgraph containing the required components and connectors of which the pattern cannot be applied without. If the pattern comprises no requirements the required fragment is an empty graph. (ii) F_R - The *modification fragment*, comprising a possibly empty graph, contains all components and connectors which have to be embedded in the architectural graph. Embedding a modification fragment covers adding and removing the fragment or parts of it, as well as replacing components and connectors of the architectural graph with components of the modification fragment or even replacing the whole graph. (iii) f_R - The *local pattern operator* defines the modification of the architectural graph, i.e., how the modification fragment is embedded, like e.g., the fragment is added and connected to a specific node.

3.3 Check Requirements

Within the step *Check Requirements*, as shown on the bottom right side of **Fig. 3** the above described requirements of the pattern to be applied get verified. Therefore, it is checked if the required fragment R is a subgraph of the (basis) architectural graph. For a positive verification, the (basis) architectural graph has to contain the required fragment R, i.e., R is a subgraph of the (basis) architectural graph. If the required fragment R is not contained within the (basis) architectural graph, i.e., the required fragment R is not a subgraph of the (basis) architectural graph, the pattern cannot be applied. This verification is done within each iteration, checking if the actual pattern of the solution path can be applied. Corresponding to the example within **Fig. 3**, for applying the pattern P_3 the (basis) architectural graph must contain a subgraph with two nodes of a specific type, as well as an edge connecting both components with each other. This procedure ensures that for the (basis) architectural graph only suitable patterns are applied and, thus, that only reasonable architectural graphs result of the next step.

3.4 Rewrite or Refine

The *Rewrite or Refine* step, shown on the bottom left side of **Fig. 3** modifies the architectural graph based on the modification fragment and the local pattern operator. As described above, the modification fragment F_R consists of all components and connectors to be embedded within the (basis) architectural graph and the local pattern operator f_R defines how the modification fragment gets embedded. This results in a modified architectural graph. As shown in **Fig. 3**, embedding the modification fragment F_R in the basis architectural graph means that the lower left node of the basis architectural graph is replaced by the middle node of the modification fragment and the remaining two component nodes and the corresponding connectors are added to the architectural graph. The modification may either result in a refined architectural graph, i.e., more details in terms of added components and connectors are depicted, or it results in a rewritten architectural graph, i.e., components and connectors are exchanged or deleted. The modified architectural graph is then used within the next iteration as the underlying architectural graph on which the next pattern of the solution path gets applied.

4 Related Work

In this section we delimit our approach against existing works that combine the knowledge of proven solutions for recurring problems, in terms of patterns, and the modeling and development process of architectures of application systems.

Eden et al. [17] introduce an approach for an automated application of design patterns. For this, programmers have to specify a pattern in an abstract way and the realization of the pattern in a specific program. Following, the pattern can be applied automatically, whereby the programmer may edit the implementation manually. Contrary to our work, patterns are used to add source code to a given program and not to model and define the architecture of an application system on a conceptual level.

Bergenti and Poggi [18] introduce the IDEA (Interactive DEsign Assistant) system to detect design patterns within UML class and collaboration diagrams. The system further enables to improve the detected pattern realizations within an UML diagram. This work focuses on the detection and improvement of design patterns and operates on a logical architecture, whereby the structural as well as the behavioral view is considered [1]. Contrary, our work operates on the conceptual level, focusing on the structural view and uses patterns to automatically model and define the architecture.

Bolusset and Oquendo [5] introduce a formal approach to refine software architectures based on transformation patterns using rewriting logic. Within their approach the refinement of an architecture does not change the architecture but specifies the components of an architecture in more detail, such as the definition of ports. Contrary, our approach results in a more detailed and possibly changed architecture. Furthermore, they use transformation patterns in terms of rewriting logic rules and equations and not in terms of best-practices and proven solutions for recurring problems.

Zdun and Avgeriou [19] present an approach to model architectural patterns through architectural primitives using UML profiles. This work focuses on modeling architectural patterns. Contrary, they do not use patterns to model and define an architecture.

Arnold et al. [20,21] introduce an approach for an automated realization of deployment patterns, which describe service deployment best-practices as model-based patterns capturing the structure of a solution without the binding to a specific resource instance. Therefore, deployment patterns have to be defined and modeled by experts so that deployer can use them. In contrast to our work, they do not use architectural patterns to model and define the architecture of an application system.

Zimmermann et al. [22] present an architectural design method based on the combination of pattern languages and reusable architectural design decision models. Contrary to our approach, within their work they use patterns in terms of architectural decisions.

Eilam et al. [23] present an approach for an automated transformation of deployment models onto workflow models. Within this work, transformation operations are represented as automation signatures including a model pattern representing the effects of operations on resources. Those automation signatures, i.e., model-based patterns are matched to a deployment desired state model. Hence, in this work patterns are used in terms of automation signatures and not as proven solutions for recurring problems.

Fehling et al. [24] present an approach to enrich application architecture diagrams by pattern annotations during the development phase. In contrast to our approach, those pattern annotations express the requirements of application components and usage dependencies on each other and on the runtime. They do not use patterns to model or define the refinement of the architecture of an application system itself.

Breitenbücher [25] and Breitenbücher et al. [26,27] introduce an approach to automatically apply management patterns onto topologies to enable the management of composite cloud applications. Contrary to our approach, they do not focus on modeling an application architecture through a selection and application of patterns. Since they operate on topology graphs and use graph isomorphism and subgraph isomorphism methods to verify if a pattern can be applied to a topology graph, this can be used within our approach as a basis to check if a pattern is applicable to a given architectural graph. Furthermore, the algorithm used to transfer the topology, i.e., the application of the pattern, can be used as a basis for the modification of architectural graphs as well.

Jamshidi et al. [28] describe a set of patterns which document how to migrate an application to a cloud environment. Within each pattern the initial as well as resulting architecture of the application is described. Even though this can be used for an automated migration, each pattern is applied to an architecture manually.

Lytra et al. [29] introduce an approach and prototypical implementation for transformation actions and consistency checking rules to (semi-)automatically map architectural design decisions onto architectural component models. Furthermore, they introduce an architectural knowledge transformation language to define and realize the mentioned mapping. In contrast to our approach, this work does not use patterns to define an architecture, but the selection of architecture design decisions results in pattern implementations. Thus, pattern realizations are one result of their approach, but patterns are not used to model or define the refinement or rewrite of an architecture.

Hirmer and Mitschang [30] describe an approach to transform non-executable data mashup plans into an executable format by selecting an appropriate pattern and further parameters. This approach is based on rule-based transformations and focuses on data processing and integration scenarios. In contrast to our work, this approach uses predefined modularized implementation fragments which are selected and scripted together.

Amato and Moscato [31] present an approach for a manual formalization of patterns resulting in workflows and automatic verification of soundness. Contrary, the developer has to model or formalize the pattern by hand and cannot apply it automatically.

Lehrig [32] and Lehrig et al. [33] introduce the architectural template (AT) method, which enables design-time analyses of quality-of-service properties of software systems based on reusable modeling templates capturing architectural knowledge. Contrary, their approach operates on existing architectures aiming the analysis of its behavioral models, and, therefore architectural templates are embedded within the architecture automatically. Nevertheless, the embedding of architectural templates can serve as a basis for the refinement and rewrite of architectural graphs.

Saatkamp et al. [34,35] present an approach to automatically detect problems in restructured deployment models by formalizing the problem and context domain of architecture and design patterns. This approach can be adapted and integrated within our approach to verify if a pattern is applicable to an architectural graph. Nevertheless, they do not apply patterns to a deployment model or use patterns to model or define the rewrite or refinement of an architecture of an application system.

Falkenthal et al. [36,37] introduce concrete solutions of patterns capturing implementation realizations, such as code snippets. Based on the selected concrete solution path, they further present a method to aggregate multiple concrete solutions into an overall solution. Contrary to our work, Falkenthal et al. do not operate on the structural view of conceptual architectures. Nevertheless, a pattern realization within an architectural graph can be considered as a concrete solution, this indicates that the aggregation of concrete solutions is also possible within our approach. In future work we will investigate if their approach is applicable to the structural view of conceptual architectures and if an aggregated application of patterns is a promising approach for our work.

5 Conclusion & Future Work

Up to this point, patterns have to be read, understood, adapted, and implemented for each use case manually. This procedure has to be integrated into the modeling process of architectures of application systems. Our approach eases the development and modeling process of architectures by combining this process with the knowledge of proven solutions for recurring problems in terms of patterns.
Within this work we introduced our approach to enable a refinement and rewrite of architectures based on selected patterns. Therefore, architectures are depicted as architectural graphs, with nodes representing components and edges representing connectors among the components of an architecture of an application system. Following a selected solution path, patterns are applied to the architectural graph successively. To achieve a coherent resulting architectural graph, patterns are only applied if their requirements are fulfilled, i.e., if the required possibly empty subgraph is present within the underlying (basis) architectural graph. The modification of the architectural graph resulting of the application of a specific pattern is based on a modification fragment, which depicts a possibly empty graph to be embedded within the architectural graph, and on the local pattern operator, which defines how the modification fragment is embedded within the architectural graph. As a result, the architectural graph can be refined through the application of patterns, i.e., components and connectors are added, or the architectural graph can be rewritten, i.e., already present components and connectors are exchanged or even deleted. Hence, our approach enables a pattern-based refinement and rewrite of application architectures by using graph transformation techniques.

Within future work, we will formalize and further elaborate the presented approach. Furthermore, we will investigate, whether there are better ways to define the application order of patterns, for example, if an application order based on the required fragments or modification fragments of all patterns of the selected solution path are more effective. Additionally, we will consider the approach of Falkenthal et al. [36,37] to investigate if an application of the aggregated patterns of the selected solution path is effective.

Acknowledgements. This work was partially funded by the German Research Foundation (DFG) project ADDCompliance (636503).

References

1. Malan, R., Bredemeyer, D.: Software architecture: Central concerns, key decisions. Software Architecture Action Guide. Architecture Resources Pubs., Bredemeyer Consulting (2002).
2. Kaartinen, J., Palviainen, J., Koskimies, K.: A pattern-driven process model for quality-centered software architecture design - A case study on usability-centered design. In: Proceedings of the Australian Software Engineering Conference, pp. 17–26. IEEE (2007).
3. Alexander, C., Ishikawa, S., Silverstein, M.: A Pattern Language: Towns, Buildings, Construction. Oxford University Press (1977).
4. Alexander, C.: The Timeless Way of Building. Oxford University Press (1979).
5. Bolusset, T., Oquendo, F.: Formal Refinement of Software Architectures Based on Rewriting Logic. Proceedings of the International Workshop on Refinement of Critical Systems: Methods, Tools and Experience 29(5), 1-20 (2002).
6. Meseguer, J.: Research Directions in Rewriting Logic. In: Computational Logic. Springer (1999).
7. Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice. Addison-Wesley (2003).
8. Allen, R., Garlan, D.: Formalizing architectural connection. In: Proceedings of the 16th International Conference on Software Engineering, pp. 71–80. IEEE (1994).
9. Le Métayer, D.: Describing software architecture styles using graph grammars. IEEE Transactions on Software Engineering 24, 521–533 (1998).
10. Fehling, C., Barzen, J., Falkenthal, M., Leymann, F.: PatternPedia – Collaborative Pattern Identification and Authoring. In: Proceedings of Pursuit of Pattern Languages for Societal Change. The Workshop 2014, pp. 252–284 (2015).
11. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-oriented Software. Addison-Wesley (1994).
12. Fowler, M.: Patterns of Enterprise Application Architecture. Addison-Wesley (2002).
13. Fehling, C., Leymann, F., Retter, R., Schupeck, W., Arbitter, P.: Cloud Computing Patterns: Fundamentals to Design, Build, and Manage Cloud Applications. Springer (2014).
14. Amazon Web Services LLC: AWS Cloud Design Pattern. http://en.clouddesignpattern.org/index.php/Main_Page, last accessed 2018/06/22.
15. Falkenthal, M., Barzen, J., Breitenbücher, U., Fehling, C., Leymann, F., Hadjakos, A., Hentschel, F., Schulze, H.: Leveraging Pattern Applications via Pattern Refinement. In: Proceedings of the International Conference on Pursuit of Pattern Languages for Societal Change, pp. 38–61. epubli (2016).
16. Zdun, U.: Systematic Pattern Selection Using Pattern Language Grammars and Design Space Analysis. In: Software: Practice & Experience 37, 983–1016 (2007).
17. Eden, A. H., Yehudai, A., Gil, J.: Precise Specification and Automatic Application of Design Patterns. In: Proceedings of the 12th IEEE International Conference Automated Software Engineering, pp. 143–152. IEEE (1997).
18. Bergenti, F., Poggi, A.: Improving UML Designs Using Automatic Design Pattern Detection. Handbook of Software Engineering and Knowledge Engineering: Volume II: Emerging Technologies, pp. 771-784 (2002).
19. Zdun, U., Avgeriou, P.: Modeling Architectural Patterns Using Architectural Primitives. In: Proceedings of the 20th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications, pp. 133–146. ACM (2005).
20. Arnold, W., Eilam, T., Kalantar, M., Konstantinou, A.V., Totok, A.A.: Pattern Based SOA Deployment. Proceedings of the 5th International Conference on Service-Oriented Computing, pp. 1–12. Springer (2007).

21. Arnold, W., Eilam, T., Kalantar, M., Konstantinou, A.V., Totok, A.A.: Automatic Realization of SOA Deployment Patterns in Distributed Environments. Proceedings of the 6th International Conference on Service-Oriented Computing, pp. 162–179. Springer (2008).
22. Zimmermann, O., Zdun, U., Gschwind, T., Leymann, F.: Combining Pattern Languages and Reusable Architectural Decision Models into a Comprehensive and Comprehensible Design Method. In: 7th Working IEEE/IFIP Conference on Software Architecture, pp. 157–166. IEEE (2008).
23. Eilam, T., Elder, M., Konstantinou, A.V, Snible, E.: Pattern-based Composite Application Deployment. In: Proceedings of the 12th IFIP/IEEE International Symposium on Integrated Network Management, pp. 217–224. IEEE (2011).
24. Fehling, C., Leymann, F., Rütschlin, J., Schumm, D.: Pattern-Based Development and Management of Cloud Applications. In: Future Internet 4, 110–141 (2012).
25. Breitenbücher, U.: Eine musterbasierte Methode zur Automatisierung des Anwendungsmanagements. Dissertation, University of Stuttgart, Faculty of Computer Science, Electrical Engineering, and Information Technology (2016).
26. Breitenbücher, U., Binz, T., Kopp, O., Leymann, F.: Pattern-based Runtime Management of Composite Cloud Applications. In: Proceedings of the 3rd International Conference on Cloud Computing and Services Science, pp. 475–482. SciTePress (2013).
27. Breitenbücher, U., Binz, T., Kopp, O., Leymann, F.: Automating Cloud Application Management Using Management Idioms. In: Proceedings of the 6th International Conferences on Pervasive Patterns and Applications, pp. 60–69. Xpert Publishing Services (2014).
28. Jamshidi, P., Pahl, C., Chinenyeze, S., Liu, X.: Cloud Migration Patterns: A Multi-Cloud Service Architecture Perspective. In: Service-Oriented Computing - ICSOC 2014 Workshop, pp. 6–19. Springer (2014).
29. Lytra, I., Tran, H., Zdun, U.: Harmonizing architectural decisions with component view models using reusable architectural knowledge transformations and constraints. In: Future Generation Computer Systems 47, 80–96 (2015).
30. Hirmer, P., Mitschang, B.: FlexMesh - Flexible Data Mashups Based on Pattern-Based Model Transformation. In: Rapid Mashup Development Tools, pp. 12–30. Springer (2016).
31. Amato, F., Moscato, F.: Pattern-based orchestration and automatic verification of composite cloud services. In: Computers and Electrical Engineering 56, 842–853. Elsevier Ltd (2016).
32. Lehrig, S. M.: Efficiently Conducting Quality-of-Service Analyses by Templating Architectural Knowledge. Dissertation, University of Stuttgart, Faculty of Computer Science, Electrical Engineering, and Information Technology (2018).
33. Lehrig, S., Hilbrich, M., Becker, S.: The architectural template method: templating architectural knowledge to efficiently conduct quality-of-service analyses. In: Software: Practice and Experience 48, 268–299 (2018).
34. Saatkamp, K., Breitenbücher, U., Kopp, O., Leymann, F.: An Approach to Automatically Detect Problems in Restructured Deployment Models based on Formalizing Architecture and Design Patterns. Computer Science - Research and Development (2018).
35. Saatkamp, K., Breitenbücher, U., Kopp, O., Leymann, F.: Application Scenarios for Automated problem Detection in TOSCA Topologies by Formalized Patterns. In: Proceedings of the 12th Advanced Summer School on Service Oriented Computing. IBM Research Division (2018).
36. Falkenthal, M., Barzen, J., Breitenbücher, U., Fehling, C., Leymann, F.: Efficient Pattern Application: Validating the Concept of Solution Implementations in Different Domains. In: International Journal On Advances in Software 7, 710–726 (2014).
37. Falkenthal, M., Barzen, J., Breitenbücher, U., Leymann, F.: On the Algebraic Properties of Concrete Solution Aggregation. In: Computer Science - Research and Development (2018).

The SustainLife Project – Living Systems in Digital Humanities

Claes Neuefeind[1], Lukas Harzenetter[2], Philip Schildkamp[1], Uwe Breitenbücher[2], Brigitte Mathiak[1], Johanna Barzen[2], and Frank Leymann[2]

[1] University of Cologne, 50923 Cologne, Germany
[2] University of Stuttgart, 70569 Stuttgart, Germany
{c.neuefeind,philip.schildkamp,bmathiak}@uni-koeln.de
{lukas.harzenetter,uwe.breitenbuecher,
johanna.barzen,frank.leymann}@iaas.uni-stuttgart.de

Abstract. In the arts and humanities, research applications play a central role in securing and presenting digital results. However, due to their steadily increasing number and their heterogeneity, it is difficult to ensure the sustainability and durability of this kind of living systems from an organizational point of view. This paper describes a project for the preservation of specialized web-based research applications in the humanities. The SustainLife project investigates to what extent methods and technologies of professional cloud deployment and provisioning strategies can be applied to problems of long-term availability of research software as they are omnipresent in humanities data centers such as the Data Center for the Humanities (DCH) at the University of Cologne. Technological basis of the project is the OASIS standard TOSCA and the Open Source implementation OpenTOSCA, respectively, which was developed at the Institute for Architecture of Application Systems (IAAS) at the University of Stuttgart. In the course of the project selected use cases from the field of Digital Humanities (DH) will be modeled in TOSCA to be able to automatically deploy them upon request at any time. The TOSCA standard enables a portable description of the modeled systems independent of specific providers to facilitate their long-term availability. The aim is to provide system components described in the use cases in a component library, as well as in the form of TOSCA-compliant application templates to make them available for reuse in other DH projects.

Keywords: Living Systems, Sustainability, Research Software.

1 Introduction

The exponential growth and the increasing use of digital research data have a significant impact on the research process in the humanities. To take advantage of the benefits of digitization, appropriate infrastructure must be created that guarantees the management of research data, its permanent availability, and free access. The large number of scientific recommendations, stocktakings, surveys, and institutional guidelines on the subject of research data, which have been published in recent years, are signs of increasing awareness of the problem, political willingness to act, but also of a continuing need for action [18, 23, 7, 19]. Existing European and national infrastructure projects in the humanities (e.g., CLARIN and DARIAH) as well as the establishment of subject-specific data centers such as, for example, the Data Center for the Humanities (DCH) of the University of Cologne (see http://dch.uni-koeln.de), have steadily improved the supply situation for research data. However, not all of the produced research results are actually made available for reuse or, respectively, are equipped for permanent availability in a highly dynamic digital world [20, p. 31, 23, p. 54].

Up to now, the discussion has mostly focused on information systems for the standardized storage and provision of primary research data. It is largely overlooked that the majority of the digital products in the humanities do not only consist of primary research data, but are, above all, available in the form of research software. In fact, digital systems such as research databases, digital editions, digital presentation systems, interactive visualizations, and virtual research environments – to name only a few – represent an essential component of the research results, especially in the context of Digital Humanities (DH). Often, these are the actual bearers of information content or, respectively, of the added value of the scientific output provided in the project [22]. A sustainability policy that falls back onto the separation and archiving of the primary data alone inevitably leads to the loss of information, and in the worst case reduces the scientific benefit to zero [1]. In contrast to traditional forms of securing results, published books for instance, the permanent maintenance, support and provisioning of such "living systems" [21] continues to be a major technical, organizational, and thereby ultimately financial challenge. While it is comparatively easy to preserve data sets of primary research data in data archives for posterity, living systems are part of a digital ecosystem and must regularly adapt to it, e.g., in the form of updates.

1.1 Problem Statement

Humanities data centers face the challenge of preserving an unknown, potentially unlimited number of research software, in order to assure their availability on a permanent basis. Internal evaluations from the consulting activities of the DCH at the Faculty of Arts and Humanities of the University of Cologne show that more than one third of all inquiries aim at the creation of individual research platforms [1]. In a survey on research data among the scientific staff at the Faculty of Arts and Humanities conducted in 2016, 62% of the respondents said that they need support for their running applications [11]. It is therefore all the more surprising that there have been only few dedicated studies so far on the exact nature of these living systems. Wuttke et al. [22] found that "research data types" in humanities can be categorized into three types of applications, namely (i) digital editions, (ii) databases, and (iii) interactive visualizations [22]. A systematic survey of all humanities research applications is still missing, both in the general research landscape and at the Faculty of Arts and Humanities. Not only a high methodological diversity is to be expected, but also a considerable diversity of the technologies used.

This heterogeneity is a major challenge for a sustainable conservation strategy. Research applications are by no means static objects, but are subject to continuous change.

For example, many applications are platforms that accept user input and, thus, constantly change their database. Browser updates and changes in usage habits can make certain components unusable or obsolete and, thus, trigger the need for a serious code revision. Vital security updates regularly require actions to be taken and can result in cascades of further updates and software adaptations. To ignore continuous maintenance may save costs in the short term, but seriously increases the problem in the medium and long term. Experiences from the “LAZARUS Project” carried out at the DCH in 2013-2015 show that applications that remain without maintenance for a longer period of time can only be revitalized at a great expense of resources [6].

Furthermore, data centers are confronted with the dilemma that limited project durations make permanent operation significantly more difficult from an organizational point of view. This is particularly fatal in humanities, where often a different measure of sustainability is applied than in the comparatively fast-moving natural sciences. The institutional provision of research data for a few decades fulfills the purpose of verifiability and reproducibility in the sense of good scientific practice; but for the provision of cultural heritage objects, a time limit does not even make sense. Even if a shutdown cannot be avoided for safety or cost reasons, it must at least be ensured that the application and the data basis are archived in a way that they can be restored loss-free and reusable at any time.

1.2 Key Challenges

In summary, we see the following challenges in connection with living systems, that have to be addressed to establish a strategy for their sustainable preservation:

1. So far there is no overview of the overall landscape of research applications in the Digital Humanities, which is a necessary prerequisite for dealing with the problem.
2. A quick analysis of use cases from the field of DH shows a wide range of technologies and methodological approaches. In contrast to large commercial information systems, the field of DH is characterized by a great number of smaller, highly heterogeneous software solutions, which are all equally subject to the problem of “software aging” [17].
3. The exposure of these technologies to the Internet is inherently associated with continuous development efforts, e.g., to keep the systems permanently available through security updates.
4. However, funding for long-term software maintenance is usually very limited.
5. If a service is not needed anymore, its current application state must be stored so that it can be reinstated in the same state upon request during the next provisioning. Both concepts are currently missing.
6. Starting (research) applications ten years after their last successful provisioning, e.g. to verify the research results, does not work in most cases. Especially, if the underlying infrastructure has changed to current state-of-the-art components because of security and availability issues.

2 The SustainLife Project

Against the background of the challenges described above, the Institute for Architecture of Application Systems (IAAS) of the University of Stuttgart and the Data Center for the Humanities (DCH) of the Faculty of Arts and Humanities, University of Cologne, started the joint project "SustainLife" in March 2018, funded by the DFG under the funding line "Scientific Library Services and Information Systems" (LIS) for a period of three years. The project develops solutions based on the OASIS standard TOSCA [14, 16]. TOSCA provides a portable description of IT systems in the form of models to automate their provisioning and management which is described in more detail in Section 2.1. This enables the creation of adaptable and future-proof software architectures. In the following sections 2.1 and 2.2, the technical basis (TOSCA and OpenTOSCA) of the project is explained in more detail. The main objectives of the project are then described in Section 2.3, followed by the projects' methodology in Section 2.4.

2.1 TOSCA

The Topology Orchestration Specification for Cloud Applications (TOSCA) [14, 15, 16] is an OASIS standard for modelling, provisioning, and managing cloud applications in a standardized and provider-independent way. In TOSCA, a cloud application or service is modelled in a "Service Template". Inside a Service Template, the "Topology Template" describes the service's topology as a directed multigraph, whereby the nodes are represented by "Node Templates" and "Relationship Templates" detail the edges. Underneath, TOSCA defines a type system defining common properties and attributes in "Node Types" and "Relationship Types" respectively. For example, a web application written in PHP may be hosted on an Apache webserver which itself is hosted on an Ubuntu virtual machine. Therefore, three Node Types, "web-application", "Apache-server", and "Ubuntu-VM", as well as the "hosted-on" Relationship Type must be available. A Service Template describing this application will contain three Node Templates, "app", "webserver", and "VM", and two Relationship Templates modelling the relationship between the app and the webserver, as well as the webserver and the virtual machine – where each template is an instance of the respective type definition.

In general, TOSCA provides various extensions and mechanisms for modeling, provisioning and managing any type of software component. In addition, existing technologies, such as container systems like Docker, can be seamlessly integrated. TOSCA is therefore not a competitive approach to existing technologies, but a way to combine them [12]. However, in contrast to Docker and Kubernetes, TOSCA enables the deployment of arbitrary software components such as legacy software [12].

To automatically deploy, provision and manage the modelled service, TOSCA defines a self-contained archive called "Cloud Service Archive" (CSAR) which contains the Service Template, all Node Types and Relationship Types, as well as all required software artifacts, scripts, and binaries required for provisioning. A TOSCA runtime environment can consume a CSAR to automatically deploy and instantiate the enclosed application. For provisioning a service, TOSCA supports both, declarative and imperative [8] deployments using plans like BPMN4TOSCA [9].

2.2 OpenTOSCA

In a series of research projects, the University of Stuttgart has developed the Open-TOSCA ecosystem – an open source implementation for the TOSCA standard, which will be adapted to the specific needs of the DH and which will form the basis for the modelling of use cases within the SustainLife project. The OpenTOSCA ecosystem includes (i) the modeling tool Winery, which enables the creation of TOSCA-based application models [10], (ii) the runtime environment OpenTOSCA for automated provisioning and management of the modelled applications [4], and (iii) the self-service portal Vinothek [5], which lists all applications installed in the OpenTOSCA container and serves as a graphical interface to the user.

In a first publication by Breitenbücher et al. [3] it was already shown that the TOSCA standard is generally suitable for assuring the digital sustainability of research results [3], as research applications, which are packaged in CSARs, can be executed years later by a TOSCA runtime environment. However, there is currently no possibility to update the service's components. If a component must be exchanged because of security issues, or because it is not available anymore, the CSAR may no longer be deployable. This currently limits the use of CSARs for living systems.

2.3 Objectives

The overall goal of the project is to improve the sustainability of living systems in the Digital Humanities. Against the background of the technological requirements associated with the TOSCA standard, the subordinated goals of the project result directly from the challenges described above:

1. Our first goal is to provide a systematic overview of the field of DH that goes beyond previous literature, particularly in the area of technological and methodological classification employed in the software artifacts.
2. Investigating the multitude of possible solutions, a set of key components that are frequently used and that consequently have high potential for synergetic effects will be identified and modeled in TOSCA artifacts.
3. The project extends the OpenTOSCA ecosystem to include application templates, component types, and to implement additional management functionalities which provide standardized operating and maintenance solutions for these components, e.g., applying (security) updates or software patches.
4. Additionally, concepts are needed which are able to "freeze" and "defrost" an application to ensure that they can be reinstated in the same application state as they were decommissioned.
5. We expect this to reduce maintenance costs and will evaluate this expectation on the basis of selected use cases. Gained experiences and best practices will be fed back to the community, e.g., in workshops and publications.

2.4 The Sustainability

To achieve these goals, it is essential to first obtain an overview of the specific needs of living systems in the Digital Humanities in detail; for example, which technologies are actually being used and how can they be automated to a greater extent using the TOSCA standard. This is meant to give a general idea of the technological needs and the potential of the methodology in the context of real use cases. Based on this requirement analysis, a set of concrete use cases is selected and a concrete implementation plan is created for each of them.

On the basis of the concrete use cases, frequently used key components with great synergetic potential, typical application structures and central maintenance tasks are identified. First, the components fundamentally required to implement the selected use cases – such as web servers or operating systems – are modeled using TOSCA in order to use them for automated provisioning and sustainable maintenance. For example, the Spring framework, and various databases such as MongoDB, have to be deployable by the OpenTOSCA ecosystem. The components can then be used in further use cases and are intended to simplify future application developments, as they can be reused when modelling other applications in TOSCA.

For automated provisioning and maintenance of applications using TOSCA, associated models describing the structure of the application must be developed. This is done in close cooperation with the use case partners. In the process of modelling the use case applications, the expenses incurred and savings made are recorded, e.g. which problems occur and which requirements still exist, so that these can be processed further in the further course of the project.

To facilitate an easier application development using TOSCA and, respectively, the description of existing applications in TOSCA, model templates are developed which can be reused as a basis. For this purpose, the use cases are analyzed for typical application structures and corresponding models are created. For example, a common pattern for web applications is to use LAMP-based technologies that use **L**inux-based operating systems to run **A**pache web servers and **M**ySQL databases to run **P**HP/**P**erlin/**P**ython-based applications. Common structures of this kind should be covered by the templates. The aim of these model templates is therefore to develop new applications efficiently and with little effort using TOSCA and the OpenTOSCA ecosystem and to maintain them with the help of the management concepts developed in this project. In addition, a number of further extensions of the OpenTOSCA ecosystem are necessary for their modelling, which will be carried out in the further course of the project. These are explained in section 4.

3 Use Cases

Up to this stage of the project, four potential use cases were selected for the project as examples of “living systems”, all originating from the Faculty of Arts and Humanities of the University of Cologne. The selected use cases illustrate the diversity of the technologies used and the high degree of methodological and content-related specialization that is typical for the DH research community.

3.1 Digital Romansh Chrestomathy

Our first use case is the “Digital Romansh Chrestomathy” (DRC, see http://www.crestomazia.ch), which contains typical DH technologies like XML as well as very specific tailor-made system components. It is therefore a representative for most DH software components, as it combines several different problems existing in common DH software. The DRC was a DFG-funded cooperation project of the Department for Linguistic Information Processing and the University and City Library of Cologne between 2009 and 2011. The aim of the DRC project was to create a text corpus based on Caspar Decurtins' “Rätoromanische Chrestomathie” (RC). With its approximately 7,500 pages of text from four centuries and the coverage of the five main idioms, the RC is considered the most important collection of texts for the Romansh language. The project implemented a virtual research environment consisting of an Eclipse-based editor and a XML database, as well as a portal page for searching the texts [12]. There are still active users of the editor who contribute corrections and comments, and the search functionalities are also still in use. Since project completion in 2011, the application had to be adapted several times to ensure its usability.

Technologies (selection): Eclipse-RAP, eXist-db, Java, SCALA

3.2 Digital Averroes Research Environment

The “Digital Averroes Research Environment” (DARE, http://dare.uni-koeln.de) is a digital research platform of the Thomas Institute of the University of Cologne. The aim of the portal is to make the complete work of the philosopher Averroës publicly accessible. In recent years, a large number of manuscripts and print editions have been digitized, scientifically encoded on the basis of the TEI XML standard, and were published via the DARE portal. The current DARE platform uses the Oxygen XML-editor for data input and provides a custom search implemented in Free Pascal and XSL-generated visualizations. In the future, the DARE web application will not only provide a search and research functionality, but also the possibility to annotate the published material by external users. The application is currently being refactored to provide a browser-based user interface based on state-of-the art web technologies.

Technologies (selection): XSL, Free Pascal, Oxygen, Express-js, Nodejs, MySQL

3.3 Vedaweb

Subject of the cooperation project “VedaWeb” (see http://vedaweb.uni-koeln.de), involving the department of Linguistics, the Cologne Center for eHumanities (CCeH) and the Data Center for the Humanities (DCH) is a web-based platform for the linguistic research of old Sanskrit texts. VedaWeb will make it possible to view the texts in digital format as well as to search through them based on lexical and corpus linguistic criteria. The project builds upon the Rigveda, one of the oldest and most important texts of the Indo European language family which was composed in Vedic i.e., the oldest form of Sanskrit in late second millennium B.C. In the course of the project, various research and analytical tools will be developed and integrated into the VedaWeb platform; for example, a combined search function according to linguistic parameters (lemma, word forms, morphological and metrical information), access to various translations and commentaries, as well as the possibility to export retrieved texts in TEI format according to user defined criteria. Of central importance is the linking of the text to the Digital Sanskrit Dictionaries Cologne (see http://www.sanskrit-lexicon.uni-koeln.de), which are hosted by the CCeH.

Technologies (selection): Spring.io, elasticsearch, mongoDB, Javascript, TEI

3.4 German Early Cinema Database

Our fourth use case is the "German Early Cinema Database" (see http://earlycinema.uni-koeln.de), which is maintained by the Institute of Media Culture and Theatre at the University of Cologne. The German Early Cinema Database contains data related to film supply, distribution, exhibition and reception in Germany between 1895 and 1926. The database consists of four parts: (i) approx. 5,000 texts on early cinema in Germany between 1895 and 1914, from a wide selection of non-film sources, like specialist journals and general newspapers. (ii) Information on itinerant and fairground cinemas in Germany and neighboring countries between 1896 and 1926. (iii) Information on approx. 45,000 films available on the German market between 1895 and 1920 (irrespective of their country of origin). (iv) A sample of film programs from 1905 to 1914, mostly from permanent cinemas compiled from the newspapers of nine German cities of different regions and sizes (approx. 1,200 programs from approx. 100 cinemas, containing approx. 3,800 different films). The implementation is based on a combination of CakePHP and MySQL. Due to the age of the application, the technologies used are no longer supported by the local data processing center in cologne. Since the German Early Cinea Database is an absolutely unique resource for this type of information, it is a typical example to illustrate the need to preserve this kind of applications.

Technologies (selection): CakePHP, MySQL, JavaScript

4 Work Packages

Existing work in the TOSCA environment already contributes to the goal of modelling and preserving applications for an instantiation after several years. However, most of the work did not consider living systems and their resulting requirements. During the SustainLife project, we want to address these requirements and extend the OpenTOSCA ecosystem as depicted in **Fig. 1**. The already existing components Winery and OpenTOSCA Container will be extended by the green elements in the respective work packages (WP) during the project. While boxes describe extensions to the software components, cylinders represent repositories containing, for example, the artifacts which are describing the aforementioned use cases in TOSCA. The work packages wrapping the planned extensions are described in the following sections.

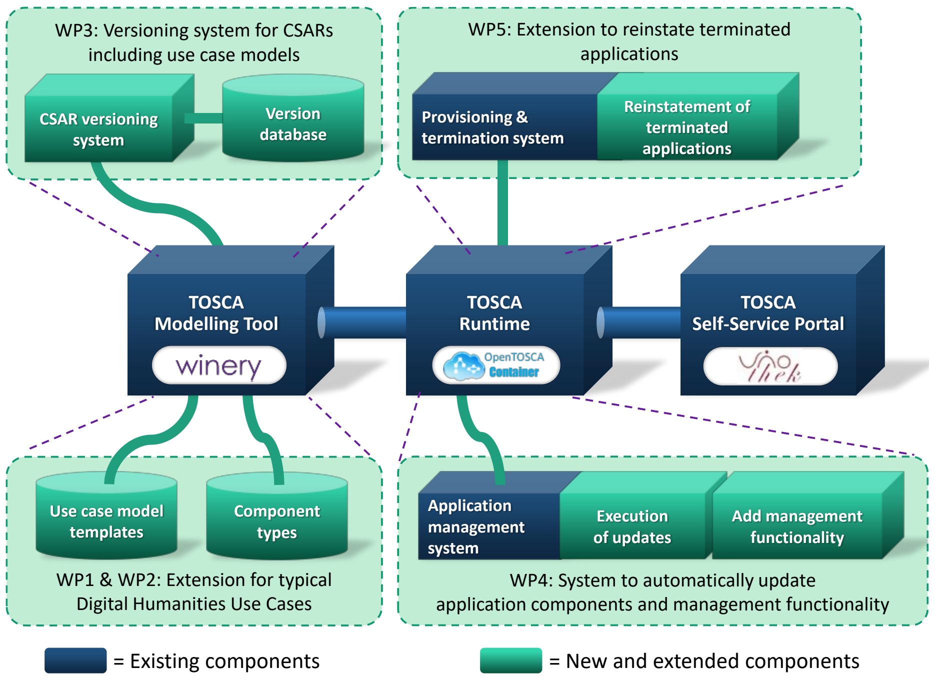

WP3: Versioning system for CSARs including use case models
CSAR versioning system
Version database
WP5: Extension to reinstate terminated applications
Provisioning & termination system
Reinstatement of terminated applications
TOSCA Modelling Tool
winery
TOSCA Runtime
OpenTOSCA Container
TOSCA Self-Service Portal
Use case model templates
Component types
WP1 & WP2: Extension for typical Digital Humanities Use Cases
Application management system
Execution of updates
Add management functionality
WP4: System to automatically update application components and management functionality
= Existing components
= New and extended components

Fig. 1. Work plan of the SustainLife project

4.1 Development of Templates for Typical Use Cases (WP1 & WP2)

TOSCA depends on a generic type system enabling the reuse of recurring components like webservers, operating systems, or messaging middleware. Within the first two work packages, we will identify the components with the highest synergetic effects and model them as TOSCA artifacts to enable their automated provisioning using the OpenTOSCA Container. Examples for components which were already identified in the presented use cases are an Ubuntu virtual machine, Java runtime environments, and several types of databases like MySQL and mongoDB.

4.2 Versioning of TOSCA Models (WP3)

Living Systems are subject to constant changes. Webservers, for example, must be updated or even exchanged regularly. Furthermore, if an application is deployed ten years after its last successful provisioning, many components will be outdated and should not be used because of, e.g., security or legal issues. Most of these changes are required because new security issues become public and need to be fixed to ensure that no attacker can gain access to the server or the data. Therefore, a versioning concept for TOSCA models is required to maintain an overview over all available versions of the components modeled in TOSCA. Since there is no versioning mechanism for TOSCA available yet, we will work on an approach in the context of the SustainLife project.

Besides the management of security updates and patches, versioning of TOSCA artifacts is also required in a more general sense. For instance, it is an important prerequisite to track the evolution of a component over time. By comparing different versions of a service or component, it is possible to reproduce the performed steps and modifications. Beyond that, versioning also yields the possibility to maintain multiple service architectures of a single application. For example, if a service was designed to run on a local infrastructure in one version, another version can describe the topology in a public cloud setting.

4.3 Modification of Management Functionality (WP4)

Currently, TOSCA supports two kinds of implementing management functionality: (i) management operations at Node Types, and (ii) separately defined management plans. In TOSCA, every component type can define custom management functionality by defining management interfaces providing different operations. For example, TOSCA describes a lifecycle interface with the operations install, configure, start, stop, and uninstall [15, 16]. The operations can be implemented, for instance, by shell scripts performing the desired operation. More complex management functionality can be described and executed by management plans. The process of provisioning a whole service can be modeled with a management plan describing every step of the process which can be executed automatically by a TOSCA runtime. OpenTOSCA's Winery supports both flavors and is able to generate management plans for the provision and termination of an application [2].

There is one major constraint, however: all management functionality must be known at modelling time. While the installation of security updates is a known functionality which can be considered during modelling time, new and unforeseen management operations cannot be added during runtime which is a requirement coming from the living systems to remain sustainable. Therefore, we want to set our research focus in work package four on developing concepts to extend the management functionality of software components during their runtime.

4.4 Concepts to Reinstate Terminated Applications (WP5)

In the context of living systems, there is the requirement for stopping and reinstating applications at any subsequent point in time. In this context, the Digital Romansh Chrestomathy (DRC) presented above serves as an example. The DRC service provides a web-based editor building upon the Eclipse-RAP framework enabling users to correct the digital version of the Romansh Chrestomathy. After the remediation is completed, the editor should still be available for documentation purposes. Since the editor may not be used for some time, it should be possible to stop the application and reinstate it in the same application state to save computing resources and consequently costs. To achieve this, the application state of the DRC service must be stored and recovered during reinstatement.

Currently, neither the OpenTOSCA ecosystem, nor other cloud management systems supporting the TOSCA standard provide a functionality for saving and restoring the application state of services which are modeled and instantiated using TOSCA in a generic way. Consequently, the fifth work package in SustainLife is dedicated to the development of concepts and functionalities to support freezing and defrosting of stateful services using TOSCA and especially the OpenTOSCA ecosystem.

5 Summary

During the SustainLife project we want to focus our research towards supporting living systems in the Digital Humanities using TOSCA. The overall objective of the project described here is to develop generic concepts for standards-based operation and maintenance solutions and to implement them for specific components and application structures in a way that they can find practical application in humanities data centers like the DCH. One major shortcoming we want to address is that CSARs should still be deployable ten years after their development. Therefore, approaches to freeze and defrost whole applications, as well as updating used components to state-of-the-art ones, will be in the focus of our research in SustainLife. Findings and best practices from the project are prepared in a way that solution models can be transferred to partners and other data centers, and are communicated to the scientific public community through workshops and publications.

Acknowledgements

This work is partially funded by the German Research Foundation (Deutsche Forschungsgemeinschaft, DFG) project SustainLife (641730).

References

1. J. Blumtritt, B. Mathiak, “Consulting Workflow for Humanities Research Data”, In: Forschungsdaten in den Geisteswissenschaften (FORGE 2016), Hamburg, 2016.
2. U. Breitenbücher, T. Binz, K. Képes, O. Kopp, F. Leymann, J. Wettinger. “Combining Declarative and Imperative Cloud Application Provisioning based on TOSCA”, IC2E, S. 87–96, 2014.
3. U. Breitenbücher, J. Barzen, M. Falkenthal, F. Leymann. “Digitale Nachhaltigkeit in den Geisteswissenschaften durch TOSCA: Nutzung eines standardbasierten Open-Source Ökosystems”. Konferenzabstracts DHd 2017: Digitale Nachhaltigkeit, S. 235-237, 2017.
4. T. Binz, U. Breitenbücher, F. Haupt, O. Kopp, F. Leymann, A. Nowak, S. Wagner. “OpenTOSCA - A Runtime for TOSCA-based Cloud Applications”. In: ICSOC, 2013, S. 692-695, 2013.
5. U. Breitenbücher, T. Binz, O. Kopp und F. Leymann. “Vinothek - A Self-Service Portal for TOSCA”. In: ZEUS 2014, p. 69-72, 2014.
6. S. Bingert, J. Blumtritt, S. Buddenbohm, C. Engelhardt, S. Kronenwett, D. Kurzawe. “Anwendungskonservierung und die Nachhaltigkeit von Forschungsanwendungen” In: Forschungsdaten in den Geisteswissenschaften (FORGE 2016), Hamburg, 2016.

7. DV-ISA. “Umgang mit digitalen Daten in der Wissenschaft: Forschungsdatenmanagement in NRW. Eine erste Bestandsaufnahme”, 2016, Version 0.7, https://www.dvisa-nrw.de/veroeffentlichungen/veroeffentlichungen-container-oeffentlich/dv-isa-vorstudie-bestandsaufnahme-forschungsdatenmanagement
8. C. Endres, Breitenbücher, U., Falkenthal, M., Kopp, O., Leymann, F., & Wettinger, J. “Declarative vs. Imperative: Two Modeling Patterns for the Automated Deployment of Applications”. In Proceedings of the 9th International Conference on Pervasive Patterns and Applications (PATTERNS), 2017, pp. 22-27
9. O. Kopp, T. Binz, U. Breitenbücher, F. Leymann. „BPMN4TOSCA: A domain-specific language to model management plans for composite applications”. In International Workshop on Business Process Modeling Notation, pp. 38-52. Springer, Berlin, Heidelberg, 2012.
10. O. Kopp, T. Binz, U. Breitenbücher und F. Leymann. “Winery – A Modeling Tool for TOSCA-based Cloud Applications”. In: ICSOC, 2013, S. 700-704, 2013.
11. S. Kronenwett. “Forschungsdaten an der Philosophischen Fakultät der Universität zu Köln” (Kölner Arbeitspapiere zur Bibliotheks- und Informationswissenschaft, Bd. 78), 2017.
12. F. Leymann, U. Breitenbücher, S. Wagner, J. Wettinger. “Native Cloud Applications: Why Monolithic Virtualization Is Not Their Foundation”. Cloud Computing and Services Science, Springer, pp. 16-40, 2017.
13. C. Neuefeind, J. Rolshoven, F. Steeg. “Werkzeuge und Verfahren für die Korpuserstellung durch kollaborative Volltexterschließung”. In: Conference of the German Society for Computational Linguistics and Language Technology (GSCL), Hamburg, pp. 163-168, 2011.
14. OASIS: “Topology and Orchestration Specification for Cloud Applications Version 1.0”, 2013.
15. OASIS. “Topology and Orchestration Specification for Cloud Applications (TOSCA) Primer Version 1.0”. 2013.
16. OASIS: “TOSCA Simple Profile in YAML”, Version 1.0, 2015.
17. D. L. Parnas. “Software Aging”. In: Proceedings of the 16th International Conference on Software Engineering (ICSE 1994). IEEE, May 1994, pp. 279-287, 1994.
18. H. Pampel, R. Bertelmann. “Data Policies im Spannungsfeld zwischen Empfehlung und Verpflichtung”. In: Handbuch Forschungsdatenmanagement, 2011, pp. 49–61.
19. Rat für Informationsinfrastrukturen. “Leistung aus Vielfalt. Empfehlungen zu Strukturen, Prozessen und Finanzierung des Forschungsdatenmanagements in Deutschland”, 2016, http://www.rfii.de/download/rfii-empfehlungen-2016/
20. M. Razum, J. Neumann. “Das RADAR Projekt: Datenarchivierung und -publikation als Dienstleistung - disziplinübergreifend, nachhaltig, kostendeckend”. In: o | bib Das offene Bibliotheksjournal, 1/1, 2014, pp. 30–44, https://www.o-bib.de/article/view/2014H1S30-44/117
21. P. Sahle, S. Kronenwett. “Jenseits der Daten: Überlegungen zu Datenzentren für die Geisteswissenschaften am Beispiel des Kölner Data Center for the Humanities”. In: LIBREAS. Library Ideas 23, pp. 76-96, 2013.
22. U. Wuttke, C. Engelhardt, S. Buddenbohm. “Angebotsgenese für ein geisteswissenschaftliches Forschungsdatenzentrum”. In: Zeitschrift für digitale Geisteswissenschaften, 2016.
23. Wissenschaftsrat. “Empfehlungen zur Weiterentwicklung der wissenschaftlichen Informationsinfrastrukturen in Deutschland bis 2020”, 2012, http://www.wissenschaftsrat.de/download/archiv/2359-12.pdf

Smart Interoperability for the Internet of Things

Sebastian Kotstein, Christian Decker

Herman Hollerith Zentrum (Reutlingen University), 71034 Boeblingen, Germany
sebastian.kotstein@reutlingen-university.de,
christian.decker@reutlingen-university.de

Keywords: Internet of Things, Interoperability, Automatic Protocol Adaptation

Seamless interoperability to share functions between Internet of Things (IoT) components is the key for building complex IoT system landscapes supporting novel and comprehensive applications [1]. Uniform data formats and communication standards, which can be understood by any involved component, are the prerequisites for such a seamless interoperability [1][2]. However, the domain of IoT is very versatile and heterogeneous from an application and technical point of view [1][3], which inevitably leads to a variety of different and often incompatible communication standards and architectural proposals [2][3]. To make matters worse, there is usually no consensus among manufactures of IoT-enabled products regarding a uniform communication standard and proprietary protocols and formats are used to isolate their own devices form extensions and communication scenarios with products from other manufactures [3][4]. Developers and system integrators of IoT-enabled devices and components are faced with the immense challenge of integrating these heterogeneous components in a uniform system landscape such that the previously postulated seamless interoperability is enabled. In practice, either an IoT platform supporting as many different communication standards and formats as possible for all involved components (so-called pre-integrated solution) is used or a solution (e.g. a middleware) allowing adaptations to the respective communication standards is chosen [3]. However, an unconstrained interoperability cannot be achieved with either approaches, since the first approach (pre-integrated solution) limits interoperability to a few supported standards and the second approach only extends this limit by manual adaptation, which is expensive and time consuming [5], but does not completely eliminate it.

Motivated by the finding that an unconstrained interoperability between IoT components cannot be solved neither by a universal communication standard, because the requirements are too versatile due to the high degree of fragmentation of the domain of IoT, nor by manual adaptation, a new approach is introduced called Smart Interoperability.

Smart Interoperability automates the adaptation of different communication standards of IoT components at runtime and enables interoperable systems in heterogeneous IoT environments. It follows the approaches of a universal IoT communication standard and the adaptation of protocols by generating a communication model between two heterogeneous IoT components at runtime. The individual aspects of communication between two heterogeneous components are recognized at runtime by applying Service Discovery techniques. Afterwards, the individual communication model is generated by combining a known set of universally valid basic primitives for communication and interaction in IoT. All steps should be performed automatically such that interoperability is achieved completely autonomously.

The ongoing research focuses on the identification and modelling of basic primitives, the autonomous adaptation of communication such that a communication model based on these basic primitives can be constructed at runtime as well as adapting Service Discovery techniques to the concept of Smart Interoperability.

This poster abstract presents the intended concept of Smart Interoperability. Furthermore, it amplifies existing approaches for autonomous adaptation of communication protocols and service composition as well as examines approaches for the identification of basic primitives based on design patterns.

References

1. Miorandi, D. et.al.: Internet of things: Vision, applications and research challenges. Ad Hoc Networks 10(7), 1497 – 1516 (2012).
2. Al-Fuqaha, A. et. al.: Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. IEEE Communications Surveys & Tutorials 17(4), 2347 – 2376 (2015).
3. Pazos, N. et.al.: ConnectOpen – automatic integration of IoT devices. In: Proceedings of the 2015 IEEE 2nd World Forum on Internet of Things, pp. 640-644. Milan, Italy (2015).
4. Rose, K. et.al.: The Internet of Things: An Overview. Internet Society (2015).
5. Sheng, Q. Z. et.al.: Web services composition: A decade's overview. Information Sciences 280, 218 – 238 (2014).

The Next Generations of Smart Data Centers*

Brian Setz[0000−0002−9750−2888]

Distributed Systems, Johann Bernoulli Insititute, University of Groningen,
Nijenborgh 9, 9747 AG, Groningen, The Netherlands
b.setz@rug.nl
http://cs.rug.nl/ds/

Data centers are responsible for 14% of the global IT energy footprint . Inside a data center, the ventilation and cooling accounts for 50% of the energy consumption, servers and storage account for 26%, followed by power conversion losses (11%), network hardware (10%), and lighting (3%)[1] . To meet sustainability goals, data centers have to increase their efficiency, especially when considering that data centers account for 1-2% of the world's energy consumption.

The NextGenSmart DC project aims to improve the overall efficiency of data centers by taking advantage of the Internet of Things (IoT) principles. The consortium of NextGenSmart DC is formed by key industrial players Cognizant (the second largest IT company in India) and Shell (the Anglo-Dutch major gas/oil company), and established research institutions in India (Institute for Development and Research in Banking Technology) and The Netherlands (Distributed Systems, University of Groningen). The central research question is: *can a combination of large-scale distributed sensors with the IoT approach and machine learning deliver a competitive and feasible solution to manage (and then design the next generation of) utility (energy, water, waste) systems for data centers.*

First, we have identified the metrics and key performance indicators that are applicable to data centers [2]. These metrics have been divided into the following categories: energy efficiency (33 metrics), cooling (8), greenness (12), performance (19), thermal and air management (17), network (10), storage (8), security (20), and financial impact (11). For each of the 130 metrics, we determine the unit, the objective (e.g. maximize), the optimal value of the metric, and the category on which the metric operates. The category refers to the level on which the metric operates: facility, IT equipment, server rack, or individual server. Selecting the correct metrics, and continuously monitoring those metrics, in order to optimize efficiency, is important in order to meet goals outlined by key performance indicators.

The quantity of data that can potentially be collected from a data center is enormous. Especially considering that there are 130 different metrics on which to evaluate a data center, each requiring multiple data sources as input. For example, the Data Center Performance Per Energy (DPPE) metric depends on 4 other metrics, which each depend on two data sources, requiring 8 data sources to evaluate the DPPE metric. The problem becomes even more striking when

* The presented research is funded by the Netherlands Organisation for Scientific Research (NWO) in the framework of the Indo-Dutch Science Industry Collaboration programme with project NextGenSmart DC (629.002.102).

metrics on individual server level are evaluated, as there can be many thousands of servers in a data center. To collect data from all data sources would require a large scale deployment of IoT devices. We have designed and developed an IoT middleware capable of processing large quantities of data in real-time, using a stream-based approach to service-oriented computing, relying on data sources, transformations, and data sinks, in order to define data streams.

To evaluate our IoT middleware in the context of data centers, we have deployed our solution to a small-scale high performance computing cluster consisting of 165 servers. From each server, 13 metrics are collected, bringing the total to 2145 metrics. Each metric is collected every 10 seconds, resulting in 18 million new data points per day. Over a period of 5 months, the data set has grown to 2.5 billion data points. We identified the correlation between metrics, utilizing the metrics with a strong correlation to build new models of the servers. Including models which determine the CPU thermals, and models which determine the load of a server, merely by measuring environmental factors external to a server. The root-mean-squared error (RMSE) of the thermal models is between 3 and 4 degrees Celsius, whereas the RMSE of the load models is between 7 to 8 percent. These models can be used to optimize load and thermal characteristics of a data center. Data from an additional 100 servers is currently being collected.

Our preliminary results highlight the potential of deploying IoT in data centers. We have analyzed 130 metrics, and surveyed 7 data centers. Based on this analysis we have determined the need for a streaming-oriented IoT middleware in order to achieve real-time processing and monitoring of all possible data streams within a data center. We have evaluated our IoT middleware on a high performance computing cluster, and uncovered interesting correlations between metrics that allowed us to define thermal and load models. Based on our preliminary results, we identify multiple directions for future research:

- Define an ontology of data center metrics, for which each unique data source can be defined and cross-metric correlations can be discovered.
- Design custom IoT-based hardware to collect even larger quantities of data from within the data center.
- Enable the automatic composition of streaming data sources to enable dynamic and real-time composition.
- Improve existing data center simulators by including environmental metrics and models which are grounded on real-world data.
- Explore the potential of automated planning and scheduling to optimize the efficiency of data center cooling based on thermal models.

References

1. Dayarathna, M., Wen, Y., Fan, R.: Data center energy consumption modeling: A survey. IEEE Communications Surveys Tutorials pp. 732–794 (Jan 2016). https://doi.org/10.1109/COMST.2015.2481183
2. Reddy, V.D., Setz, B., Rao, G.S.V.R.K., Gangadharan, G.R., Aiello, M.: Metrics for sustainable data centers. IEEE Transactions on Sustainable Computing pp. 290–303 (July 2017). https://doi.org/10.1109/TSUSC.2017.2701883

Hyperledger Fabric: Ideas for a Formal Analysis

Mike Simon and Ralf Küsters

Institute of Information Security
University of Stuttgart
Stuttgart, Germany
`{mike.simon,ralf.kuesters}@sec.uni-stuttgart.de`

1 Motivation

Since the invention of the blockchain technology in 2008 [9], interest in this rather new technology has grown rapidly. Initially invented as a solution for decentralized payment systems without a central authority, the blockchain technology has evolved. Most blockchain implementations are similar from a high-level perspective: They have a build-in cryptocurrency and operations within the blockchain are paid via this cryptocurrency. So-called smart contracts enable participants to deploy and execute code on top of a blockchain.

The possibilities of such a decentralized system draw the industries attention: (1) blockchains might make intermediaries in several processes superfluous. Without a man in the middle, processes might be faster and cheaper, (2) when transferring business processes to smart contracts, one could automate many still manual processes. Unfortunately, most famous blockchains, such as Bitcoin and Ethereum [10], do not fit the needs of the industry. These blockchains are too slow in terms of handling transactions per second. Data privacy and the realization of data protection laws is a problem, as data stored in the blockchain is typically public.

In order to solve these problems, the Linux Foundation runs the *Hyperledger Fabric* project. Hyperledger Fabric (or short *Fabric*) is a permissioned, pluggable blockchain solution without a built-in cryptocurrency [1]. It is designed for usage in inter-company and inter-country applications. Hyperledger Fabric provides solutions for the above mentioned issues: (1) it adds a certain degree of centralization to the blockchain and makes parallel execution of transaction possible. Thus, Fabric has a significantly higher transaction throughput than other blockchains, e.g., it is 200 times faster than Ethereum. (2) Fabric introduces the concept of *channels* to allow companies to handle their private data easily and to implement the requirements of data protection laws.

At first glance, Fabric seems to provide the desired security properties, such as the common prefix property, persistence, and liveness as introduced by Garay et al. [4]. Other security notions, such as the chain-quality property, may not be applicable to Fabric because blocks are "cut" (generated) by special, normally trusted network participants.

There is no formal analysis or model of Hyperledger Fabric. As, e.g., Gazi et al. showed in [5], only small mistakes in the construction of a blockchain protocol can lead to an insecure protocol. Especially in the presence of corrupted participants, one wants to know which security guarantees one can give. Only a rigorous security analysis shows whether the construction of a blockchain system is secure.

2 Research Plan

The goal of this research is to model Hyperledger Fabric in one of the common models for security analysis of cryptographic protocols, the UC model by Canetti [2] or the IITM model by Küsters et al. [8], and formally *prove* that Fabric is secure. Most of the formal analyses of blockchains protocols, such as [4], [6], [3], or [7], use the UC model. However, the UC model has several technical problems that, e.g., invalidate the composition theorem. Therefore it is preferable to use the simpler, more general, and formally sound IITM model.

The following steps are planned:

Step 1: Determine the most important configuration of Fabric in industry. Especially the selection of the used consensus algorithm and the so-called *endorsement policy* (which defines the rules “who” executes a transaction) have a direct impact on how to model Fabric.

Step 2: Devise a model of the selected configuration. Prepare an appropiate, realistic attacker model which captures the expected capabilities of an adversary in the context of Fabric.

Step 3: Analyze and adapt the above mentioned security notions for Fabric. Investigate whether the chain-quality notion is applicable for Fabric. Develop a useful security notion to measure the power of an adversary if chain-quality does not fit.

Step 4: Prove security of Fabric under realistic assumptions, and possibly propose fixes if necessary.

Step 5: Generalize the developed model and try to cover the pluggable design of Fabric. Extract requirements for the different Fabric components, which are necessary to provide security.

Acknowledgement. This research was partially funded by the Ministry of Science of Baden-Württemberg, Germany, for the Doctoral Program “Services Computing”.[1]

Keywords: Hyperledger Fabric, Formal Security Analysis, Blockchain, Smart Contracts

[1] `http://www.services-computing.de/?lang=en`

References

1. Androulaki, E., Barger, A., Bortnikov, V., Cachin, C., Christidis, K., Caro, A.D., Enyeart, D., Ferris, C., Laventman, G., Manevich, Y., Muralidharan, S., Murthy, C., Nguyen, B., Sethi, M., Singh, G., Smith, K., Sorniotti, A., Stathakopoulou, C., Vukolic, M., Cocco, S.W., Yellick, J.: Hyperledger fabric: a distributed operating system for permissioned blockchains. In: Proceedings of the Thirteenth EuroSys Conference, EuroSys 2018, Porto, Portugal, April 23-26, 2018. pp. 30:1–30:15 (2018), http://doi.acm.org/10.1145/3190508.3190538
2. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: 42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA. pp. 136–145 (2001), https://doi.org/10.1109/SFCS.2001.959888
3. David, B.M., Gazi, P., Kiayias, A., Russell, A.: Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake protocol. IACR Cryptology ePrint Archive 2017, 573 (2017), http://eprint.iacr.org/2017/573
4. Garay, J.A., Kiayias, A., Leonardos, N.: The bitcoin backbone protocol: Analysis and applications. In: Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II. pp. 281–310 (2015), https://doi.org/10.1007/978-3-662-46803-6_10
5. Gazi, P., Kiayias, A., Russell, A.: Stake-bleeding attacks on proof-of-stake blockchains. IACR Cryptology ePrint Archive 2018, 248 (2018), http://eprint.iacr.org/2018/248
6. Kiayias, A., Russell, A., David, B., Oliynykov, R.: Ouroboros: A provably secure proof-of-stake blockchain protocol. In: Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I. pp. 357–388 (2017), https://doi.org/10.1007/978-3-319-63688-7_12
7. Kosba, A.E., Miller, A., Shi, E., Wen, Z., Papamanthou, C.: Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In: IEEE Symposium on Security and Privacy, SP 2016, San Jose, CA, USA, May 22-26, 2016. pp. 839–858 (2016), https://doi.org/10.1109/SP.2016.55
8. Küsters, R., Tuengerthal, M.: The IITM model: a simple and expressive model for universal composability. IACR Cryptology ePrint Archive 2013, 25 (2013), http://eprint.iacr.org/2013/025
9. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. Bitcoin Whitepaper (2008), https://bitcoin.org/bitcoin.pdf
10. Wood, G.: Ethereum: A secure decentralised generalised transaction ledger. Ethereum Project Yellow Paper 151, 1–32 (2014), http://gavwood.com/paper.pdf

All links were last followed on April 26, 2018.

Efficient Data and Indexing Structure for Blockchains in Enterprise Systems

Christian Riegger, Tobias Vinçon, and Ilia Petrov

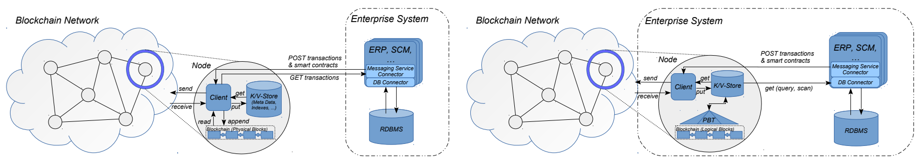
Data Management Lab, Reutlingen University
Alteburgstraße 150, 72762 Reutlingen, Germany
{first}.{last}@reutlingen-university.de

Abstract. Blockchains are a technique for managing transactions in trustless distributed systems and target among other domains new markets for enterprises. At present, there is limited support for trustworthy K/V-Stores for clients of nodes in blockchain networks. Furthermore, mixed workloads from blockchain network and enterprise systems require full support of storage, schema and indexing of blockchain data in K/V-Stores. However, there is at best a partial match between the characteristics of current data structures and the properties of blockchain and enterprise workloads. We claim that Partitioned B-Trees represent an appropriate scalable data and indexing structure, which fits well these properties on modern hardware technologies, due to its flexible partition management, single index structure and multi-version capabilities.

Keywords: Blockchain · Enterprise Workload · Data Structure.

1 Blockchain Network

Blockchains receive growing attention as a possible technology for Distributed Ledgers. The logical data organization in blockchains is a backward linked list of blocks, which contain encrypted *transaction* data. A blockchain network consists of several nodes, over which the blockchain is synchronized. On every node, a blockchain network client is implemented [1–4]. Clients implement and are responsible for operations interacting with the network. K/V-Stores are the backbone of clients and manage blockchain (meta) data. Businesses include blockchain data in their evolved enterprise systems [6].



(a) State-of-the-Art Architecture (b) Required Architecture

Fig. 1. Architecture of Nodes in a Blockchain Network for Enterprise Systems

In Fig 1(a), the enterprise systems interact with the blockchain network by messaging services to a node network client. A K/V-Store maintains meta data and few indexes (cf. [1–4]). Data in blocks is separated from the K/V-Store. Current architecture on nodes, separation of indexes and data, as well as format conversions for messaging services, result in stagnation of business processes and management decisions, due to high latencies on data gathering. Including the K/V-Store in enterprise systems as a database (depicted in Fig. 1(b)) minimizes format conversions and latencies, because all data is managed by the K/V-Store. Required data can be directly accessed by the enterprise systems. Therefore, a flexible data and indexing structure is required, that is able to efficiently manage all blockchain data in "Logical Blocks" and scale on modern hardware.

A data and indexing structure has to consider characteristics of blockchain data. Uniformly distributed hash values of blockchain data are stored in blocks. Hashes refer to real *accounts*, *transactions* and *smart contracts*, which are appended all at once in a block. Each block is analogous to a version state in the Distributed Ledger, whereby data is ordered by processing time. Blocks retain consensus and store immutable data. Recently appended blocks may be rejected by the network and if so their data becomes invalid. *Unspent transactions* are not immediately added to the blockchain, but have to be validated and eventually processed next. *Smart contracts* probably contain analyzable data.

2 K/V-Store Data and Indexing Structure

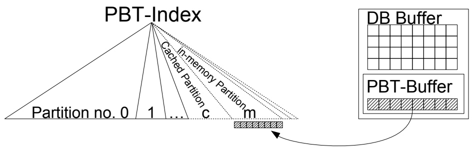


Fig. 2. Structure of a Partitioned B-Tree

Partitioned B-Trees (PBT) [5] enable flexible partition management in a common index structure by prepending a partition number to each record. Updates are absorbed by a mutable partition in main memory [7]. Further complexity (bulk loads of block data, *unspent transactions*, multi-version capabilities for performance gains in mixed workloads [8], etc.) can be handled by maintaining additional mutable in-memory partitions [5], which are conditionally merged in further steps. For instance, partitions, which contain data of rejected blocks, can be cropped from the index with low effort, what solves maintenance problems outlined in [9]. Immutable partitions are results of evictions from main memory to secondary storage media in a beneficial sequential write pattern [7]. Version ordering of blockchain data buoys for fast look-ups in partitions, which can be also reorganized in merge operations and "Cached Partitions" for query optimization or switches in workloads. Furthermore, a common index structure, partition skipping techniques, asymmetry in semi-conductors and near-data-processing approaches guarantee acceptable look-up performance. PBTs scale well on modern hardware and their capabilities facilitate the performant architecture for blockchains and enterprise systems, which is depicted in Fig. 1(b).

References

1. bitcoin/bitcoin: Bitcoin Core integration/staging tree (2018), https://github.com/bitcoin/bitcoin, accessed: 2018-03-01
2. ethereum/go-ethereum: Official Go implementation of the Ethereum protocol (2018), https://github.com/ethereum/go-ethereum, accessed: 2018-03-01
3. Hyperledger Project (2018), https://github.com/hyperledger, accessed: 2018-03-01
4. paritytech/parity: Fast, light, robust Ethereum implementation. (2018), https://github.com/paritytech/parity, accessed: 2018-03-01
5. Graefe, G.: Sorting and indexing with partitioned b-trees. In: CIDR 2003, First Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 5-8, 2003, Online Proceedings (2003), http://www-db.cs.wisc.edu/cidr/cidr2003/program/p1.pdf
6. Morabito, V.: Blockchain and Enterprise Systems, pp. 125–142. Springer International Publishing (2017), https://doi.org/10.1007/978-3-319-48478-5_7
7. Riegger, C., Vinçon, T., Petrov, I.: Write-optimized indexing with partitioned b-trees. In: Proceedings of the 19th International Conference on Information Integration and Web-based Applications & Services. pp. 296–300. iiWAS '17, ACM (2017). https://doi.org/10.1145/3151759.3151814
8. Schwalb, D., Faust, M., Wust, J., Grund, M., Plattner, H.: Efficient transaction processing for hyrise in mixed workload environments. In: Proceedings of the 2nd International Workshop on In Memory Data Management and Analytics, IMDM 2014, Hangzhou, China, September 1, 2014. pp. 16–29 (2014), http://www-db.in.tum.de/hosted/imdm2014/papers/schwalb.pdf
9. Third, A., Domingue, J.: Linked data indexing of distributed ledgers. In: Proceedings of the 26th International Conference on World Wide Web Companion. pp. 1431–1436. WWW '17 Companion (2017), https://doi.org/10.1145/3041021.3053895

Interoperability as Basis for Novel Interaction Concepts in the Sterile Supply Process

Veronika Krauß[1] and René Reiners[1]

Fraunhofer Institute for Applied Information Technology FIT, Schloss Birlinghoven, 53754 Sankt Augustin, Germany http://fit.fraunhofer.de
{firstname.lastname}@fit.fraunhofer.de

Abstract. The emerging topic of Internet of Things (IoT) allows novel kinds of applications and process support in many domains leading to radical innovations but also reveal problems to currently existing systems, infrastructures and working processes. This work describes a twostep approach to investigate possibilities for interaction concepts in combination with display technologies for the sterile supply process in clinics. First, it describes an underlying central application platform for integrating existing devices, services and data sources. Building on top of this interoperability layer, new application and interaction concepts are defined that are iteratively engineered with end-users to ensure appropriateness and acceptance for the final system design.

Keywords: Interoperability · Smart Glasses · Sterile Supply Process · Digital Hospital

1 Objective and Challenges

In the sterile supply process, it is important for multiple data sources, devices, and user roles to work hand in hand. Establishing interoperability is challenging due to proprietary APIs, data formats as well as established, partially legacy, software systems that are in productive use. In order to support working processes by providing currently needed and relevant information, a common communication layer needs to be created, which is open and flexible enough to allow the integration of new systems and data formats. After this communication layer is set, new concepts for supporting the working processes can be investigated. Apart from the technology-driven requirements, established ways of working which foremost affect humans and their work at hand need to be understood and taken into account for new designs and ideas. Only an iterative, user-centered design approach gives room for understanding user needs and practices. Based on gained application knowledge, new rapidly prototyped solutions can emerge. This way, the final design has very high chances for user acceptance and improvement in everyday work.

2 Step One: The Underlying Information Layer

The overall system's architecture makes use of the LinkSmart® [1] Middleware, which allows for rapidly connecting devices and data by offering REST and MQTT APIs. Hardware used for cleaning and sterilizing surgery goods in combination with hospital management systems can easily be integrated by offering a corresponding interface to LinkSmart®. Any application that allows visualization and interaction will now be able to access information by either subscribing indirectly and/or publishing to an MQTT broker, or directly by accessing the responsible resource via REST. In the latter case, the resource's address will be obtained using a service catalog offered by LinkSmart®. A general overview of the described information layer is presented in Fig. 1.

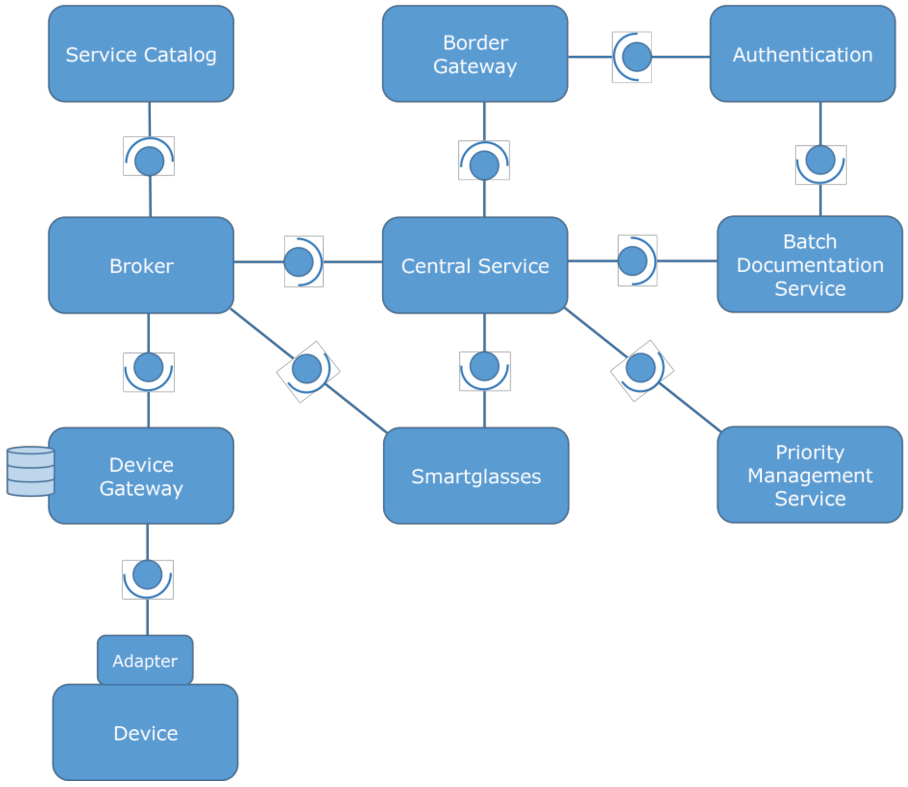


Fig.1. Sample architecture demonstrating the integration of devices, services and software using LinkSmart® as an information layer and application platform. Devices are integrated using a dedicated adapter. Following this approach, also various interaction devices can get access to the system.

3 Step 2: User-Centered Application Engineering

In parallel to the completion of step one, relevant use cases were identified by following a user-centered design approach. With the help of end users, visiting facilities and given the possibility to watch and learn from ”living” processes in a normal working environment, different user roles, tasks, process steps and current obstacles could be identified. The identified use cases span the process steps of cleaning, packing and sterilizing surgery goods alongside with first-level support mechanisms for workers. For the sake of brevity in this work, details of the use cases will be omitted.

Now that the domain, task models and user roles are better understood, an iterative prototyping process starts exploring new ways of supporting the personnel working in the sterilization process for surgery goods. The focus lies on increasing the overall process quality and employees’ safety. Every iteration ends with the evaluation of the current design, be it a concept, storyboard, cognitive walkthrough or, in later stages, a hands-on prototype. The approach ensures the integration of new requirements as well as findings that usually come up after communicating ideas [2]. The final prototype will be implemented on a Microsoft Hololens [3] and demonstrate the newly gained flexibility and quality of information that can be delivered on user demand.

4 Future Work

The overall goal of this work is to show the benefits of loosely coupled services in the sterile supply process that allow an easy interchange of visualizations and interaction technologies. For this purpose, a demonstrator is currently under development that supports the packing tasks for surgery packages that are delivered to the operating room.

As irst interaction device, smart glasses are used to enable hands-free interaction, augment existing systems and therewith overcome current obstacles as well as to increase the overall quality of the process. A comparison between side displays as well as fully-ledged augmented reality systems such as the Microsoft HoloLens [3] will provide deeper insights regarding suited visual support.

In order to better support hands-free interaction with the system, other devices for recognizing gestures or voice interfaces will be explored. LinkSmart® as information layer offers the open basis for exploring the design space.

References

1. Linksmart, https://www.linksmart.eu/. Last accessed 17 Jul 2018
2. Alan Dix, Janet Finlay, Gregory Abowd, and Russel Beale. Human-Computer Interaction. Prentice Hall Inc., Upper Saddle River, NJ, USA, 2nd edition, 1998
3. Microsoft Hololens Homepage, https://www.microsoft.com/de-de/hololens. Last accessed 10 May 2018