An Approach Towards a Service Co-evolution in the Internet of Things

Huu Tam Tran(✉), Harun Baraki, and Kurt Geihs

Distributed Systems Group, University of Kassel, Kassel, Germany
{tran,baraki}@vs.uni-kassel.de, geihs@uni-kassel.de
http://www.vs.uni-kassel.de

Abstract. In the envisioned Internet of Things (IoT), we expect to see the emergence of complex service-based applications that integrate cloud services, connected objects and a wide variety of mobile devices. These applications will be smarter, easier to communicate with and more valuable for enriching our environment. They interact via interfaces and services. However, the interfaces and services can be modified due to updates and amendments. Such modifications require adaptations in all participating parties. Therefore, the aim of this research is to present a vision of service co-evolution in IoT. Moreover, we propose a novel agent architecture which supports the evolution by controlling service versions, updating local service instances and enabling the collaboration of agents. In this way, the service co-evolution can make systems more adaptive, efficient and reduce costs to manage maintenance.

Keywords: Service co-evolution · IoT services · Web services

1 Introduction

In this paper, we address the challenge of coordinated services in the scope of IoT by employing an agent-based approach. Service providers may depend on third party services to deliver quality products to customers and to other service providers as well. To prevent outages and failures by individual service modifications and updates coordinated evolution (hereafter co-evolution) is required in such complex systems, i.e. they need a co-evolution for services in order to ensure that no interruptions occur. A centralized solution would not be realizable due to administrative and technical reasons. It would not be scalable, in particular, in the area of IoT, and security issues would complicate the whole approach. Consequently, service providers have to be responsible for the evolution of their own services. The required actions have to be coordinated with other providers in the IoT environment. The objective is to automate the coordinated evolution as much as possible.

Recently, agent-based models have been suggested for IoT as they can capture autonomy, and proactive and reactive features. Beside that, they can include ontologies for cooperation and different contexts [1,2]. Within the scope of IoT,

R. Giaffreda et al. (Eds.): IoT360 2014, Part I, LNICST 150, pp. 273–280, 2015.
DOI: 10.1007/978-3-319-19656-5_39

agent approaches address application levels and can use services provided by smart objects in order to achieve co-evolution.

Service co-evolution in IoT has received barely attention so far. Thus, there are some needs for detailing the vision of service co-evolution and solutions to provide benefits for IoT users. However, there are many challenges and requirements to tackle to meet an overall tradeoff between aspects like the satisfaction of clients, the resource consumption of provided interface versions and the efforts to update them. Consequently, this paper will analyze the roles of this evolution regarding potential results, challenges and its requirements as well as the solution.

It is not the intention of this paper to present details of Web service evolution as that has been done elsewhere [4,5]. This paper aims at promoting the idea of co-evolution of web services in IoT by (i) illustrating how a service co-evolution is carried out, what should be involved, why it is essential, and what should be prepared in order to meet the co-evolution requirements, (ii) highlighting a novel agent architecture for service providers in the IoT environment and explaining how this agent can be used in service co-evolution environments, (iii) discussing some potential research challenges of service co-evolution. Thus, the main contribution of this paper is to make software engineers aware of the power of service co-evolution and make systems more adaptive, efficient and reliable.

The rest of the paper is structured as follows. Section 2 illustrates an overview of our solution and its key components. Section 3 analyzes the coordination of services and discusses research challenges in service co-evolution. Section 4 introduces a number of existing researches and compares them with our approach, and finally Sect. 5 draws conclusions on our current results and provides an outlook for future work.

2 Solution Overview

Services running on heterogeneous systems and offered by different providers have de-coupled lifecycles, in particular, in IoT. Single services will be updated due to amendments or refinements or to provide further functionalities. Other providers may cut back the functionalities without taking notice of remaining clients that try to apply the removed functions. Business processes and applications that depend on services require appropriate coordination and adaptation by the participating parties. The solution we worked out equips every service with an agent, called EVA (EVolution Agent), that is capable to undertake these tasks. The internal structure and the rough composition of an EVA is depicted in Fig. 1. The next sections introduce the main components of an EVA and their interactions.

2.1 Analysis

The information interaction flow within our model is as follows. When an EVA receives first an Evolution Request, it is analyzed by the Analysis module.

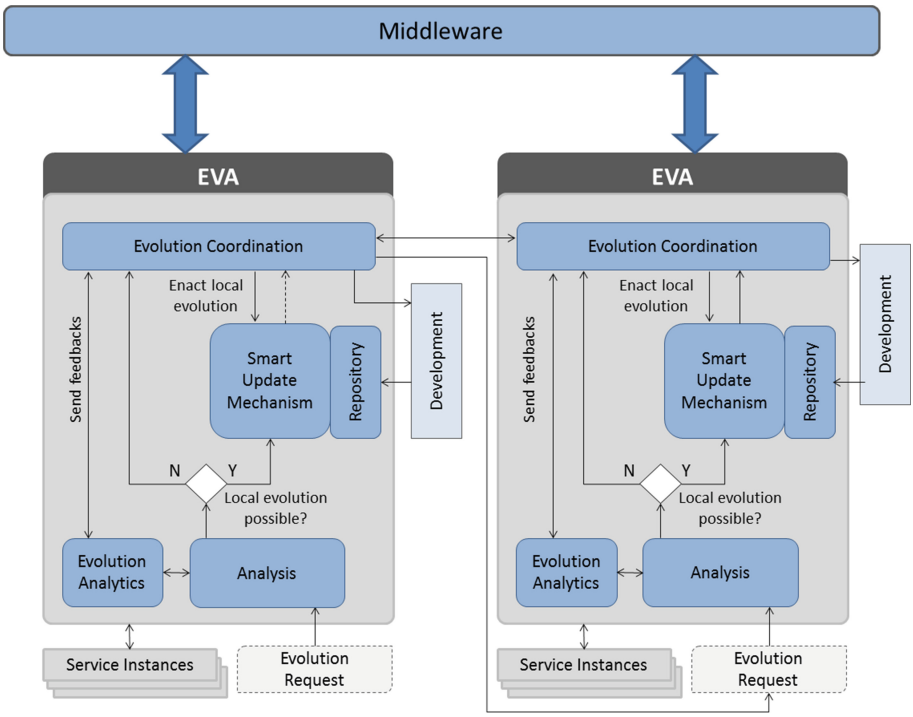

Fig. 1. Architecture of the EVA: global view

An Evolution Request demands for adaptation to be able to take part in future interactions. The Analysis module has to decide whether an evolution should take place and, if so, whether a local evolution is possible or whether the evolution has to be coordinated with other EVAs. For this reason, it assesses firstly the significance of the Evolution Request by evaluating the importance, the reputation and the number of partners who sent the request. The importance of a partner will increase, the more clients are affected by him. The significance will rise too, if the local service strongly depends on the other service and if there are no alternative services available. If either resources are becoming scarce or if it takes high efforts to satisfy the request, then lowly rated Evolution Requests may be rejected. Service instances not requested for a long time can be switched off to free resources for crucial service instances.

To estimate the efforts required for adaptation, the Analysis module considers initially local knowledge that includes information about locally available update mechanisms, the different service instances realizing different versions of the service, and the dependencies that the service versions might have towards other services. In case the Analysis module accepts the Evolution Request and a local update would satisfy the request, it will instruct the Smart Update Mechanism module, as presented below, to execute the local update and to provide eventually

a new service instance. If a pure local update is not available or not sufficient due to an interplay between several services, the Evolution Coordination module has to deal with a coordinated evolution and possibly ask software developers for further configurations.

2.2 Evolution Analytics

As time passes, the Analysis and the Evolution Coordination module can take more sophisticated decisions. The Evolution Analytics module collects runtime data about successful and unsuccessful evolution procedures. These data include information about local and coordinated evolutions since both modules feed the Evolution Analytics module. The goal is to discover promising evolution patterns by fostering successful and proven evolution procedures and preventing unsuccessful ones. Success does not only depend on smooth running in a technical sense, but has to consider the cost-performance ratio, the revenue, the reputation and QoS (Quality of Service) parameters too. Costs comprise, for instance, hardware and human resources which can be estimated hardly in the very beginning. If a new configuration has been implemented, the developer specifies the total man-hours spent. By means of Evolution Analytics EVA will learn to predict worthwile evolutions while minimizing costs and time and maximizing the own revenue and reputation. The reputation of an EVA may decrease if it denies regularly Evolution Requests. Here, Evolution Analytics has to weigh the reputation against other factors like the costs for updates and the future revenue. To estimate reputation, costs and QoS, we will make use of our two prediction algorithms presented in [7].

For reasons of bootstrapping, EVAs are allowed to share parts of their knowledge with other EVAs. Special know how that affects only the service supervised by the EVA, has to be left out.

2.3 Evolution Coordination

In the event that a pure local evolution is not applicable, the Evolution Coordination module will co-operate with other EVAs and possibly interact with software developers. For example, the service is providing a method that depends on data delivered by a third party. To customize the interface for the client sending the Evolution Request, the Evolution Coordination will determine first the involved third parties and send them an Evolution Request. A continuous feedback between the EVAs is required to keep all parties up-to-date and to recognize future developments early. If a third party rejects the Evolution Request or if it is not available anymore, the Evolution Coordination can start a search for suitable services. To this end, we will adopt our service selection algorithms proposed in [8]. If the latter fails due to a lack of matching services, the Evolution Coordination will instruct the service provider or a responsible software developer to adapt the service. For this purpose, the developer may implement a configuration that is subsequently executed by the Smart Update Mechanism.

2.4 Smart Update Mechanism

The Smart Update Mechanism encompasses mainly two types of evolution capabilities. Firstly, it is aware of the different versions of the services running as service instances on the local machine and the versions used in the past. If one of them is fulfilling the conditions required, then it will be assigned to the requesting party. The second approach is a specification of the evolution rules and constraints that represent the possible service re-configurations and adaptations. In MUSIC [6] application developers specified the possible variants of an application and their dependencies on the runtime context; this was exploited by the adaptation manager in the middleware to achieve optimized application adaptation in different situations.

An EVA maintains up-to-date evolution models of its services. The models expose the possible configuration and adaptation paths. The EVA may govern multiple instances and versions of the same service at the same time, in order to accommodate different applications that may have different needs with respect to the service. Eventually, out-dated alternatives will be slowly retired.

The Analysis and Evolution Coordination modules introduced in the previous sections decide which configuration or version will be used for a specific client. In this connection, they do not only consider the possibilities offered by the Smart Update Mechanism, but take also into account the Evolution Analytics to optimize criteria like revenue, reputation, response time and own operability.

2.5 Repository and Middleware

The Smart Update Mechanism makes use of a repository where several configurations were made available by developers. Developers can add new configurations to the repository during the lifetime of a service, for instance, if the Smart Update Mechanism did not find appropriate ones to update the service.

Since objects or mobile devices are free to enter or leave the system, the middleware enables EVAs to communicate with each other in an asynchronous and loosely coupled manner. Beside that, the EVA itself can be divided into its modules such that each module may run on another device. This allows to make use of powerful runtime environments while energy constrained IoT devices that deliver the data offered by the service are spared.

3 Challenges of Service Co-evolution

Service co-evolution needs to be managed in a decentralized fashion since a centralized approach constitutes a bottleneck and would not be scalable. The services have to be responsible for their own evolution and should coordinate their actions with other services according to the knowledge they have of their own capabilities and that of the other services.

Coordinating the evolution of services is a major challenge since it is a complex process that requires multiple interactions, as well as continuous feedback

to understand whether the distributed evolution is proceeding as desired. To prevent never-ending negotiations between service providers about which service has to adapt first or to change at all, we introduce an algorithm that gives a clear path for the evolution. Therefore, we include the number of clients of each concerned service and their overall reputation.

The EVA that is managing an affected service is either interested in an adaptation or rejects it. For this reason, an EVA can vote for (vote = +1) or against (vote = -1) the evolution of a used service. The higher the reputation of a service and the higher its number of clients, the higher the vote of the EVA that is managing the service is weighted. Thus, the overall feedback is comprised of the multiplication of the vote and the weight that consists of the reputation and the number of clients. This means that services that satisfy and affect more clients have a higher impact. A step-wise structure of the proposed algorithm is given in the following:

STEP 1: *An EVA x receives an evolution request from another EVA y.*

STEP 2: *x is asking the EVAs $c \in C$ of its clients whether they would accept or reject the required adaptation.*

STEP 3: *x is summing up the feedbacks of $c \in C$ by considering their vote and their reputation and number of clients that are both scaled into the range* $[0, 1]$.

STEP 4: *x is dividing the summed up feedbacks by the number of clients to obtain f_{agg} and compares f_{agg} with a predefined threshold value ϵ.*

STEP 5: *x is striven for the co-evolution if $f_{agg} \geq \epsilon$. In this case, the aforementioned steps of Sect. 2 will be executed. Otherwise, the evolution requests will be rejected.*

Mostly, evolution cannot be fully automated. In general, it is a multi-step process that a service must go through to transition from a problematic configuration, to a more acceptable new one. This transition may involve adaptation mechanisms that are already in-place, as well as offline activities, such as requirements gathering and software development. Although software evolution mechanisms have been deeply studied in the last decades, service co-evolution offers further research challenges:

- Heterogeneous services in IoT have de-coupled lifecycles, meaning that single services may be updated, or newly developed, while others are still in operation. Any evolution that we perform on a service requires that this action be coordinated with other actions paramount if we want to preserve the applications overall functionality and quality of service.

- The evolution of such complex systems will require that we harness and understand the horizontal and vertical relationships that exist between services, so that we can have them evolve in a coordinated fashion. This can be achieved through modeling and analytics, and through detailed runtime analysis, e.g., runtime testing and formal verification. Given the decentralized nature of the application environment, all these tools need to rely on local knowledge of the service itself and of its surroundings.

4 Related Work

Joo Pimentel et al. [11] outlined the reasoning required in order to support forward and backward co-evolution of service-oriented systems. But this paper only analyzed how to assess the mutual impact of requirements and architecture changes on other service oriented systems and how to react to these changes in order to prevent misalignment between them.

The paper in [12] also provides a theoretical framework and language-independent mechanisms for controlling the evolution of services that deal with structural, behavioral, and QoS level-introduced service changes in a type-safe manner. In particular, the authors distinguished between shallow changes (small-scale, localized) and deep changes (large-scale, cascading). However, the authors only focused to deal with shallow changes. In contrast, our paper tries to deal with vital aspects of deep changes with an adaptive agent approach.

Authors in [10] introduced a change-oriented service life-cycle methodology and described its phases. They discussed when a change in a service is triggered, how to analyze its impact, and the possible implications of the implementation of the change for the service providers and consumers. Nevertheless, a formal model for deep changes based on the one for shallow changes is missing. Additionally, their approach did not mention about the IoT environment which differs from our approach.

The Chain of Adapters technique [13] is an alternative approach for deploying multiple versions of a Web service in the face of independently developed unsupervised clients. The basic idea is to resolve the mismatches between the expected by the consumers and the supported by the implementation interface [12]. It can prove useful in self-configurations. However, it is not clear whether the approach would scale to a high number of Web services.

In fact, there are many agent-based approaches available to support interoperable IoT devices and their services nowadays [1–3,9]. Nonetheless, the adaptation mechanisms and the collaboration characteristics in these agents are not sufficient in order to achieve coordinated service evolution. Furthermore, it needs a global vision which can predict potential effects, challenges and requirements for participating service providers.

5 Conclusion and Future Work

The main goal of this paper is to introduce a new vision of service co-evolution and to provide a common evolution management model and reference architecture for developers. It represents a focused effort to provide a foundation for realizing the full potential of the Internet of Services and other service-based architectures. For this reason, the challenges in the co-evolution of services that cover the wide spectrum from IoT to Cloud Computing, are analyzed too.

This paper also adopts a novel conceptual agent as a solution for service co-evolution. Evolution tasks like the assessment and coordination of evolution requests, updating and versioning the interfaces and selecting matching services

can be performed automatically or semi-automatically by EVAs, inter alia, by exploiting already existing self-adaptation techniques.

In future, first research prototypes and scenarios on the coordination of EVAs shall be delivered to evaluate the prospect of this approach. A further path for future work is to develop an evolution description language for service co-evolution. In this way, systems can be made more adaptive, efficient and reduce costs to manage maintenance.

References

1. Atzori, L., Iera, A., Morabito, G.: The internet of things: a survey. Comput. Netw. **54**(15), 2787–2805 (2010)
2. Ayala, I., Amor, M., Fuentes, L.: An agent platform for self-configuring agents in the internet of things. In: Infrastructures and Tools for Multiagent Systems, pp. 65–78 (2012)
3. Roalter, L., Kranz, M., Möller, A.: A middleware for intelligent environments and the internet of things. In: Yu, Z., Liscano, R., Chen, G., Zhang, D., Zhou, X. (eds.) UIC 2010. LNCS, vol. 6406, pp. 267–281. Springer, Heidelberg (2010)
4. Leitner, P., Michlmayr, A., Rosenberg, F., Dustdar, S.: End-to-end versioning support for web services. In: IEEE International Conference on Services Computing, SCC 2008, vol. 1, pp. 59–66. IEEE (2008)
5. Fokaefs, M., Mikhaiel, R., Tsantalis, N., Stroulia, E., Lau, A.: An empirical study on web service evolution. In: 2011 IEEE International Conference on Web Services (ICWS), pp. 49–56. IEEE (2011)
6. Floch, J., Frà, C., Fricke, R., Geihs, K., Wagner, M., Gallardo, J.L., Cantero, E.S., Mehlhase, S., Paspallis, N., Rahnama, H., Ruiz, P.A., Scholz, U.: Playing music - building context-aware and self-adaptive mobile applications. Softw. Pract. Exper. **43**(3), 359–388 (2013)
7. Baraki, H., Comes, D., Geihs, K.: Context-aware prediction of qos and qoe properties for web services. In: 2013 Conference on Networked Systems (NetSys), pp. 102–109. IEEE (2013)
8. Comes, D., Baraki, H., Reichle, R., Zapf, M., Geihs, K.: Heuristic approaches for QoS-based service selection. In: Maglio, P.P., Weske, M., Yang, J., Fantinato, M. (eds.) ICSOC 2010. LNCS, vol. 6470, pp. 441–455. Springer, Heidelberg (2010)
9. Yu, H., Shen, Z., Leung, C.: From internet of things to internet of agents. In: Green Computing and Communications(GreenCom), 2013 IEEE and Internet of Things, IEEE International Conference on and IEEE Cyber, Physical and Social Computing, (iThings/CPSCom), pp. 1054–1057. IEEE (2013)
10. Papazoglou, M.P., Andrikopoulos, V., Benbernou, S.: Managing evolving services. IEEE **28**(3), 49–55 (2011)
11. Pimentel, J., Castro, J., Santos, E., Finkelstein, A.: Towards requirements and architecture co-evolution. In: Bajec, M., Eder, J. (eds.) CAiSE Workshops 2012. LNBIP, vol. 112, pp. 159–170. Springer, Heidelberg (2012)
12. Andrikopoulos, V., Benbernou, S., Papazoglou, M.: On the evolution of services. IEEE Trans. Softw. Eng. **38**(3), 609–628 (2012)
13. Kaminsi, P., Müller, H., Litoiu, M.: A design for adaptive web service evolution. In: Proceeding of the 2006 International Workshop on Self-Adaptation and Self-Managing Systems, May 21–22, 2006, Shang Hai, China (2006)