EAI Endorsed Transactions
on Cloud Systems

Research Article ICST.ORG

# Service Co-evolution in the Internet of Things

Huu Tam Tran*, Harun Baraki and Kurt Geihs

University of Kassel, Distributed Systems Group, Wilhelmshöher Allee 73, Germany

## Abstract

The envisioned Internet of Things (IoT) foresees a future Internet incorporating smart physical objects that offer hosted functionality as IoT services. This service-based integration of IoT will be smarter, easier to communicate with and more valuable for enriching our environment. 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, Coordinating agents, IoT services, Web services, Adaptive services

Received on 01 December 2014, accepted on 20 January 2015, published on 23 February 2015



doi: 10.4108/cs.1.1.e5

## 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.

With the emergence of Internet Protocol-based IoT devices [9] and the concept of embedded Web services [16, 19], new and highly interconnected IoT-aware applications can be created. Nonetheless, changes can happen at any stage in the service life cycle and have unpredictable impact on the service stake-holders [23]. Though Web services bring more flexibility, they also create new challenges for change management in the Web service lifecycle. How to handle those changes for each Web service consumer as well as facilitate the client application updates on the consumer side? This question has become an emerging concern for Web service providers and Web service consumers. Being therefore able to control how changes manifest in the service life cycle is essential for both service providers and service consumers [23]. In fact, changes of Web service can occur in three aspects of services: change in the functional behavior of the service; change in the non-functional behavior of the service; and syntactical changes in the service interface.

Recently, agent-based models have been suggested for IoT as they can capture autonomy, and proactive and reactive features. Besides that, they can include ontologies for cooperation and different contexts [2, 3]. Within the scope of IoT, 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 trade-off between aspects like the satisfaction

*Corresponding author. Email: tran@vs.uni-kassel.de

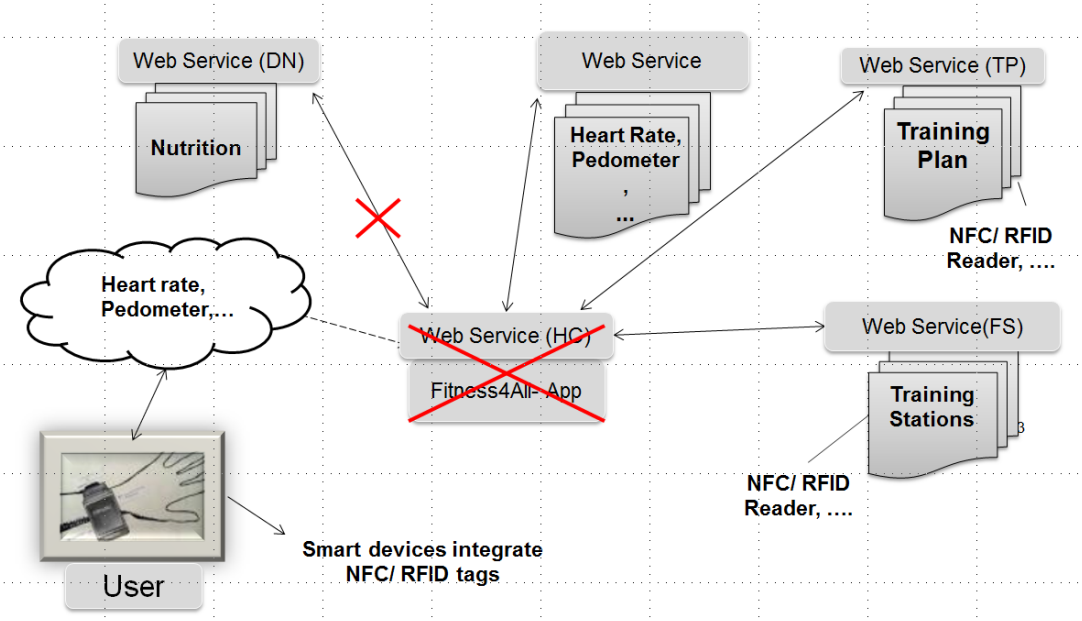


**Figure 1.** Fitness4All disrupts due to service updates.

of clients, the resource consumption of provided interface versions and the efforts to update them. Consequently, this paper will analyse 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 [11–13, 15]. 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 these agents 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 provides a motivating example in the healthcare area. Section 3 illustrates an overview of our solution and its key components. Section 4 analyses the coordination of services. Section 5 introduces a number of existing researches and compares them with our approach, and finally section 6 draws conclusions on our current results and provides an outlook for future work.

## 2. A motivating example

To clarify and illustrate the idea of co-evolution, we first present a scenario set on Health Care applications, and some important concerns for application development in the IoT environment. HealthCareCo (HCC) is a company specializing in providing healthcare services for customers. Through its Web service, the company provides a service named Fitness4All to customers (users). Users can employ Fitness4All to obtain information about health indicators such as heart rate, number of daily steps, insulin level, arterial blood pressure as well as daily calorie consumption. Fitness4All can also provide diagnosis of the illness as well as consulting users by offering a user guide on nutrition and exercises to promote health. Users will be advised to practise in health centers like Fitness Stations Co (FSC) or Training Plan Co (TPC).

The company's customers are provided with a smart device which can access remotely Fitness4All services. This smart device has an integrated RFID tag. When customers go into FSC or TPC, local RFID readers will identify the user. At the same time, the smart device will connect to the center's Web service. After that, FSC or TPC will provide the appropriate training for all users through personal health information based on the Fitness4All. During training, the device will signal an alert warning to the users if there are any health issues. In addition, Fitness4All can advise on nutrition information to the customers. This service is supported from another company named Daily Nutrition Co (DNC) which provides various nutritious foods to people. Suddenly one day, the Web services from service providers like FSC, TPC and DNC updated to new version without notifying HCC. This leads to the disruption of Fitness4All. Users can not access the

information from the web service. It created many claims and the company had to return money back to users. HCC had to invest many financial resources and human efforts to continue providing Fitness4All. HCC Board of Directors understood that the emerging social Internet sector was still immature because of its lack of efficacy in dealing with changes while ensuring the quality of the composite. After some time, customers no longer trust the Fitness4All service as several interruptions occurred 1. As a consequence, the company had to stop the service and its activities.

After HCC Board's initial problems they decided to contact a consultancy firm called IoT-SU(an IoT-support Company), specialized in supporting companies that use third party internet services to implement robust systems. IoT-SU knows that when they publish a certain service through an interface, they have to be ready to maintain different versions of that interface over time, in order to not deny service to hundreds of thousands of subscribers.

However, service providers would like to publish new interfaces and forget about maintaining old ones. This would allow them to evolve the services faster, and continuously offer improved services for their subscribers. The solution to face this problem is called service co-evolution. Service Co-evolution artifacts are also installed on the consumer's side so that changes in the interfaces can be managed mostly automatically. If it cannot be adjusted automatically, then the development team of HCC will be informed. In this case, the service provider may offer the old version in parallel to the new version for a specified period of time to avoid an abrupt interruption of HCC.

IoT-SU helped the HCC service to adopt the service co-evolution approach. The sustainability of the selected solution allowed them to continue the provision of their services and to expand their user base.

Figure 2 illustrates a possible service co-evolution scenario with support by Evolution Agents (EVAs).

Third party service providers usually update interfaces in order to make their services more reliable and faster or to provide further functionality. In the introductory example the developers forgot about maintaining the previous versions for the Fitness4All service. In that case, for a short-term solution, the technical group of HCC should contact soon the third party service providers to get support in updating the interfaces. In the longer term, the technical group should deploy EVA for their Web services. When an update appears from the Web service providers, the Fitness4All Web service will be update almost simultaneously if appropriate update mechanisms are available or it will inform developers immediately.

It is imperative that an application has to cope with such evolutions, so that its business continuity is not compromised. Unfortunately, we cannot rely on a centralized evolution manager, since it would represent a traffic bottleneck and a central point of failure. Having a centralized manager might not even be feasible, for technical or administrative reasons.

Evolution also 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 many new research challenges:

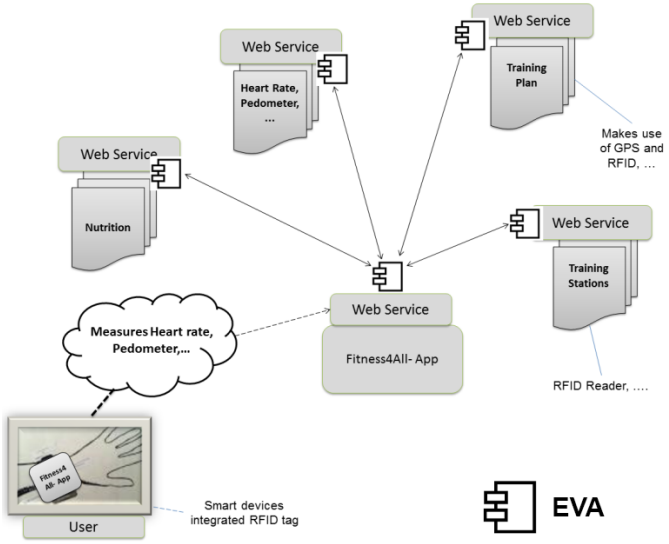


**Figure 2.** The service co-evolution with EVAs in the scenario.

(i) 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.

(ii) 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 modelling and analytics, and through detailed runtime analysis, e.g., runtime testing and formal verification. Given the decentralized nature of the application environment, all these

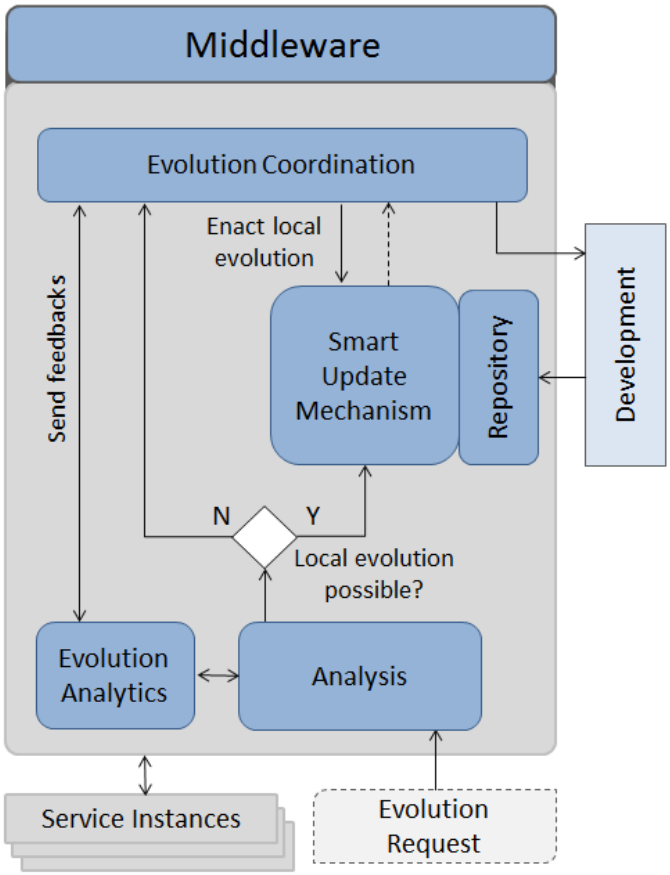


**Figure 3.** Architecture of the EVA

tools need to rely on local knowledge of the service itself and of its surroundings.

## 3. 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 are depicted in figure 3. The next sections introduce the main components of an EVA and their interactions.

### 3.1. Analysis

The information interaction flow within our model is as follows. When an EVA receives first an Evolution Request, it is analysed by the Analysis module. An Evolution Request demands for adaptation to be able to take part in future interactions. In case a service provider wants to update his service, the Evolution Request can be sent by its EVA to the EVAs of the clients. This scenario is discussed in section 4. A further scenario is that the EVA of a service that is composed of other services and depends on them, demands one of his service providers to evolve to be able to update his own service.

In the latter case, the analysis module of the receiving EVA 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 interplay between several services, the Evolution Coordination module has to deal with a coordinated evolution and possibly ask software developers for further configurations.

### 3.2. 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 worthwhile 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 [4].

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.

### 3.3. 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 [7].

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.

### 3.4. 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 [10] application developers specified the possible variants of an application and

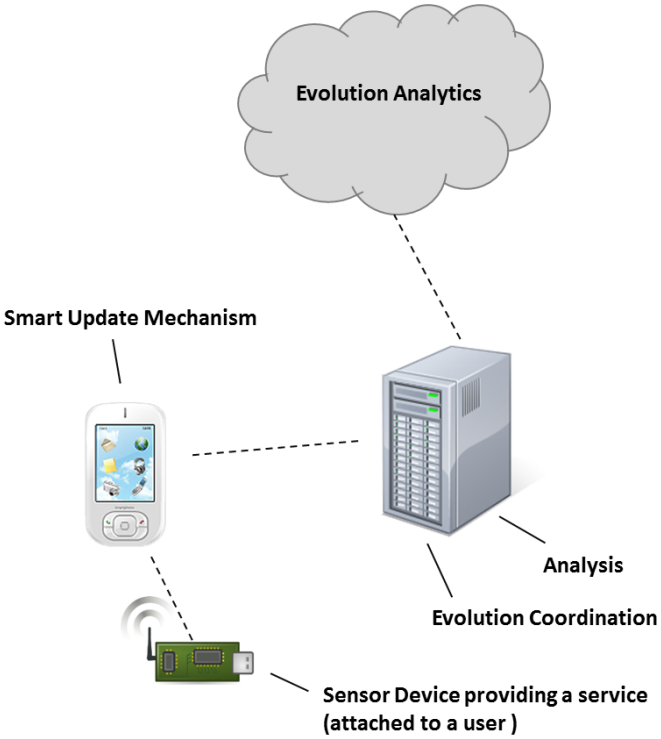


**Figure 4.** Deployment of an EVA in an IoT scenario

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.

### 3.5. Repository

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.

### 3.6. Middleware

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. Besides 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. Small services, such as an object in IoT, will not have the processing power or storage needed to implement a full EVA. Figure 4 shows a low-level object that has outsourced its EVA components. This is a conservative deployment scenario since we assume that the on-site server and the cloud are always available.

However, more opportunistic approaches are also conceivable. For example, an object might rely on the availability of mobile devices that can enter or leave the system freely, to provide the resources it needs to communicate with the cloud.

## 4. Coordination of EVAs

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. Figure 5 shows the process of taking into account the feedback received in response to an evolution request, particular from EVA x which sends the requests to its clients.

To tackle the aforementioned challenge, we define the following terms to coordinate and adapt EVAs.

Weight of a service (w):

The weight of a service is defined as the product of the reputation r of the service (range [0,1]) and its number of clients $n_c$ (scaled into the range [0,1] by incorporating the max and min values of all services considered). The reputation r of the service and the weight w of a service can be defined respectively as follows:

$$r = \frac{\sum_1^n rating_i}{n} \tag{1}$$

$$w = r \times n_c \tag{2}$$

Vote of a service (v): 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 or against the evolution of a used service. Hence, we adopt the values of votes:

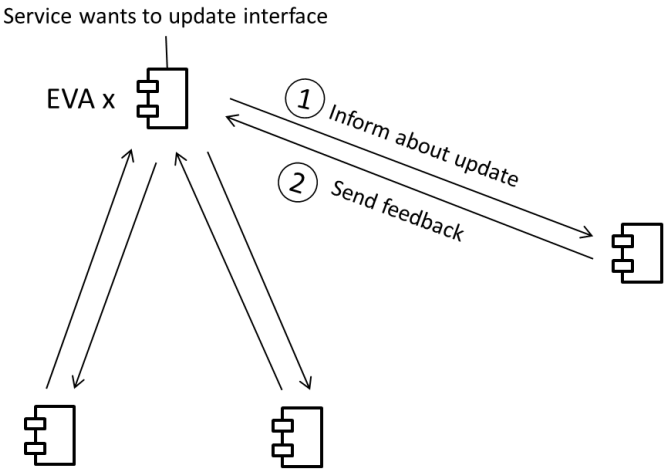


**Figure 5.** Coordination of EVAs based on client's feedback

- Vote $(v_i) = +1$ votes for accepting the new service version.
- Vote $(v_i) = -1$ votes for not updating the interface or do update but keep the old version.

Feedback (f): 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.

Feedback of one client $(f_i)$:

$$f_i(w_i, v_i) = w_i \times v_i \tag{3}$$

Feedback of all $n$ clients $(f_{agg})$:

$$f_{agg} = \sum_i^n w_i \times v_i \tag{4}$$

### 4.1. Coordination algorithm

The co-evolution will be executed if

$$f_{agg} \geq \epsilon(thresholdvalue) \tag{5}$$

A step-wise structure of the proposed algorithm that encompasses the equations from (1) to (5), is given in the following:

**Input:** Evolution request of EVA x to EVAs $c \in C$; number of clients and the reputation of the EVAs $c \in C$.

**Step 1:** The service managed by EVA x will be updated by the provider or EVA x received 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 $\epsilon$.

**Step 5:** The co-evolution will be executed if $f_{agg} \geq \epsilon$. In this case, the update mechanisms will be executed.

**Step 6:** Otherwise, the evolution requests will be rejected.

**Output:** Accept or reject the evolution requests

### 4.2. Optimization Problem

Web services may be composed of several other Web services and, hence, stay in contact to different EVAs. This scenario is similar to that of business processes in SOA (Service-oriented Architectures), where a central process orchestrates Web services to realize a certain functionality. The orchestration is formulated as a process that may start with the arrival of a message like a receive or a pick activity. It may send an answer back to the requester with a reply activity. Activities and service invocations may be grounded inside a sequence or a flow structure.

In service co-evolution, we are searching for the optimal interface. A service may be a composition of multiple EVAs where each EVA represents a Web service. The orchestration itself is monitored and managed by an EVA. It tries to optimize the own revenue and the rate of satisfaction of clients by providing a suitable interface for a given functionality. This interface is built by a composition of the interfaces of other EVAs (services). The goal of an selection algorithm is to find first an optimal choice of interfaces that realizes the most preferred interface for the orchestration. This is done by the aforementioned coordination algorithm. With EVA x, for instance, we define the set I that contains the set of requested service interfaces. The set I depends on the clients who vote whether they would accept or reject the interface of a composition of certain interfaces.

After finding the set I of matching interfaces, the challenge is to find those services that optimize the overall quality of the orchestration for the clients and the revenue for the provider. Finding the optimal solution means now to maximize the overall satisfaction (S) of the own clients and to select those EVAs that will provide the required interfaces and that maximizes the own revenue (R).

We consider the following objective function for realizing the EVA orchestration:

$$MaximizeF_{obj}(I) = F_{obj1}(R) \times F_{obj2}(S) \quad (6)$$

$F_{obj1}(R)$ and $F_{obj2}(S)$ are explained below. $F_{obj1}(R)$ is an objective function for maximizing the provider's revenue, $F_{obj2}(S)$ has the goal to maximize the clients' satisfaction.

We incorporate the following quality dimensions to compute the overall satisfaction for the clients:

- $throughput(t)$ : number of service invocations per time unit.
- $reliability(l)$ : the probability that the service executes successfully.
- $executiontime(e)$ : the time it takes to execute the service.
- $availability(a)$ : the percentage of time during which the service is available.

The vector Q (t, l, e, a) contains the quality of service (QoS) dimensions.

The value of functions $F_{obj1}$ and $F_{obj2}$ are weighted and combined to make the final decision. To maximize $F_{obj}(I)$ in (6) both objective functions below has to be maximized:

$$F_{obj1}(R) = \sum_{i=1}^{n} r_i, r_i \in R \quad (7)$$

$$F_{obj2}(S) = k_1 \times t + k_2 \times l + k_3 \times \frac{1}{e} + k_4 \times a \quad (8)$$

Therefore:

$$F_{obj}(I) = (\sum_{i=1}^{n} r_i) \times (k_1 \times t + k_2 \times l + k_3 \times \frac{1}{e} + k_4 \times a) \quad (9)$$

To keep the description as simple and clear as possible, the normalization steps to the range of [0,1] are not included in the formula. The factors $k_i, i = 1...4$ represent the weights for the quality dimensions depending on the preferences of the EVA. Let us assume that the current interface of EVA x can provide a revenue of $R_0$ and an average satisfaction of $(t_0, l_0, e_0, a_0)$. The goal of interface selection would be to find a pair $(R_i, S_i)$ better than the pair $(R_0, S_0)$.

In the event of pure Web service selection for business processes, this issue can be solved by our approach in the paper [7] or by other popular approaches like integer linear programming [24] and genetic algorithms [6].

## 5. Related Work

Over the last decades, the service evolution raised to a more and more important topic that brings many new challenges to software engineering. This section will present some frequently cited works related to our research.

One of first works handling the problem of service evolution is developed by Forkaefs et al. [11]. Their

tool VTracker is designed to analyse the evolution of WSDL interfaces. The idea of Vtracker is based on the Zhang-Shashas tree-edit distance [25] which calculates the minimum edit distance between to trees. In this study the WSDL interfaces are considered as usual XML files. Specifically the authors created an intermediate XML representation to reduce the verbosity of the WSDL specification. In this simplified XML representation, among other transformations, the authors trace the references between message parameters and data types and replace the references with the data types themselves. However, VTracker does not take into account the syntax of WSDL interfaces. As consequence, their approach outputs only the percentage of differences between XML elements. In addition, this approach of transforming a WSDL interface into a simplified representation can lead to the detection of multiple changes while there has been only one change.

Similarly, Romano and M. Pinzger [18] presented an outstanding work called WSDLiff that compares subsequent versions of WSDL interfaces to automatically extract the changes. This approach takes into account the syntax of the WSDL file and the schema file XSD [18] that is used to define the data types of the WSDL interface. In particular, WSDLDiff extracts the types of the elements affected by changes (e.g., Operation, Message) and the types of changes (e.g., removal, addition, move, attribute value update). Romano at al. refer to these changes as fine-grained changes. The fine-grained changes extraction process of WSDLDiff is based on the UMLDiff algorithm [21]. This proposed tool is a useful means to understand how a particular Web service evolves over time. This approach is relevant for Web service subscribers who want to compare the evolution of different Web services with similar features or to analyze the most frequent changes affecting a WSDL interface. By applying this approach, Web service subscribers can estimate the risk associated to the usage of a certain Web service. Nonetheless, the authors did not investigate the co-evolution of different Web service which differs from our approach.

Other well-known research results come from M.P. Papazoglou and V. Andrikopoulos [1, 15] with analyzing shallow changes and deep changes. In their papers, they developed a set of theories and modes that unify different aspects of services (description, versioning, and compatibility) to assist service developers in controlling and managing service changes. They distinguished between shallow changes (small-scale, localized) and deep changes (large-scale, cascading) for service compatibility and reasoning mechanisms for delimiting the effect of changes which can keep local to and consistent with a service description. They discussed when a change in a service is triggered, how to analyse its impact, and the possible implications of the implementation of the change for the service providers and consumers. However, the authors only focused to deal with shallow changes. Additionally, their approach did not mention about the IoT environment and its services. But some lessons can be learned from their formal principles and theories for the description of the coordination of EVAs, for instance.

Design patterns have been widely used for software development for structuring solutions [8, 20]. S. Wang et al [20] focuses on a common evolution scenario where a single service, provided by a single provider, is used by many different and possibly unknown consumers, as is the case of most current Web services, such as Google Maps, eBay Trading, and Amazon E-Commerce. In the scenario of [20], the services usually face large and frequent changes as a result of an increasing need to conform to changing business and technological requirements [20]. In particular, the paper proposed four patterns involving compatibility, transition, split-map, and merge-map. These patterns provide generic and reusable strategies for service evolution. These patterns can be involved to deploy and support our agents as they can be used to derive the set of changes.

Another important service evolution approach is the analysis of service dependencies. Basu et al. [5] introduced a tool that can extract dependencies from log files. Their technique could be adapted in order to infer a set of dependent service consumers. One the dependencies are understood, it is also important to infer the impact of service changes on the dependent applications. The Chain of Adapters technique [12] 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 expectations of the consumers and the supported versions and configurations of the services. This can prove useful in self-configurations.

It is worth mentioning the work on service compatibility with the WS2JADE tool [14] which is based on an agent approach. Xuan Thang Nguyen and Ryszard Kowalczyk proposed the WS2JSADE toolkit for integrating Web services and the Jade agent platform. This tool provides facilities to deploy and control Web services as agent services at run time for deployment flexibility and active service discovery. The authors also discuss different ways how Web services can be visible to agents and how they can be accessed and used by agents. Although WS2JADE offers many advantages over other existing tool, it still lacks substantial theoretical work with respect to agent to Web service integration like, for instance, service co-evolution.

In fact, there are many agent-based approaches available to support interoperable IoT devices and their services nowadays [2, 3, 17, 22]. 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.

## 6. Conclusion and Future Work

This paper introduces a new vision of service co-evolution in IoT. It provides developers a common evolution management model and reference architecture and represents a focused effort to provide a foundation for realizing the full potential of 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 as well.

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.

Furthermore, the paper also proposes an approach for coordinating EVAs in service co-evolution. Besides that, it proposes a first approach for the selection of interfaces in orchestrations that is required to satisfy revenue and satisfaction requirements. In this way, systems can be made more adaptive, efficient and reduce costs to manage maintenance.

In summary, the paper is a step forward to service co-evolution in IoT. The results of this research will provide support to professional service providers and business process engineers. In future, first research prototypes for the coordination of EVAs shall be delivered to evaluate the prospect of this approach.

## Acknowledgment

The authors would like to acknowledge the generous support of DAAD (Deutscher Akademischer Austauschdienst).

## References

[1] Andrikopoulos, V., Benbernou, S., Papazoglou, M. P., 2012. On the evolution of services. Software Engineering, IEEE Transactions on 38 (3), 609–628.

[2] Atzori, L., Iera, A., Morabito, G., 2010. The internet of things: A survey. Computer Networks 54 (15), 2787–2805.

[3] Ayala, I., Amor, M., Fuentes, L., 2012. An agent platform for self-configuring agents in the internet of things. INFRASTRUCTURES AND TOOLS FOR MULTIAGENT SYSTEMS, 65–78.

[4] Baraki, H., Comes, D., Geihs, K., 2013. Context-aware prediction of qos and qoe properties for web services. In: Networked Systems (NetSys), 2013 Conference on. IEEE, pp. 102–109.

[5] Basu, S., Casati, F., Daniel, F., 2008. Toward web service dependency discovery for soa management. In: Services Computing, 2008. SCC'08. IEEE International Conference on. Vol. 2. IEEE, pp. 422–429.

[6] Canfora, G., Di Penta, M., Esposito, R., Villani, M. L., 2005. An approach for qos-aware service composition based on genetic algorithms. In: Proceedings of the 7th annual conference on Genetic and evolutionary computation. ACM, pp. 1069–1075.

[7] Comes, D., Baraki, H., Reichle, R., Zapf, M., Geihs, K., 2010. Heuristic approaches for qos-based service selection. In: Service-Oriented Computing. Springer, pp. 441–455.

[8] Daigneau, R., 2012. Service Design Patterns: fundamental design solutions for SOAP/WSDL and restful Web Services. Addison-Wesley.

[9] Dunkels, A., et al., 2009. Efficient application integration in ip-based sensor networks. In: Proceedings of the First ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings. ACM, pp. 43–48.

[10] 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., 2013. Playing music - building context-aware and self-adaptive mobile applications. Softw., Pract. Exper. 43 (3), 359–388.

[11] Fokaefs, M., Mikhaiel, R., Tsantalis, N., Stroulia, E., Lau, A., 2011. An empirical study on web service evolution. In: Web Services (ICWS), 2011 IEEE International Conference on. IEEE, pp. 49–56.

[12] Kaminski, P., Müller, H., Litoiu, M., 2006. A design for adaptive web service evolution. In: Proceedings of the 2006 international workshop on Self-adaptation and self-managing systems. ACM, pp. 86–92.

[13] Leitner, P., Michlmayr, A., Rosenberg, F., Dustdar, S., 2008. End-to-end versioning support for web services. In: Services Computing, 2008. SCC'08. IEEE International Conference on. Vol. 1. IEEE, pp. 59–66.

[14] Nguyen, X. T., Kowalczyk, R., 2007. Ws2jade: Integrating web service with jade agents. Springer.

[15] Papazoglou, M. P., Andrikopoulos, V., Benbernou, S., 2011. Managing evolving services. Software, IEEE 28 (3), 49–55.

[16] Priyantha, N. B., Kansal, A., Goraczko, M., Zhao, F., 2008. Tiny web services: design and implementation of interoperable and evolvable sensor networks. In: Proceedings of the 6th ACM conference on Embedded network sensor systems. ACM, pp. 253–266.

[17] Roalter, L., Kranz, M., Möller, A., 2010. A middleware for intelligent environments and the internet of things. In: Ubiquitous Intelligence and Computing. Springer, pp. 267–281.

[18] Romano, D., Pinzger, M., 2012. Analyzing the evolution of web services using fine-grained changes. In: Web Services (ICWS), 2012 IEEE 19th International Conference on. IEEE, pp. 392–399.

[19] Shelby, Z., 2010. Embedded web services. Wireless Communications, IEEE 17 (6), 52–57.

[20] Wang, S., Higashino, W., Hayes, M., Capretz, M. A., 2014. Service evolution patterns. Proceedings of the 21st IEEE International Conference on Web Services.

[21] Xing, Z., Stroulia, E., 2005. Umldiff: an algorithm for object-oriented design differencing. In: Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering. ACM, pp. 54–65.

[22] Yu, H., Shen, Z., Leung, C., 2013. From internet of things to internet of agents. In: Green Computing and Communications (GreenCom), 2013 IEEE and Internet of Things (iThings/CPSCom), IEEE International Conference on and IEEE Cyber, Physical and Social Computing. IEEE, pp. 1054–1057.

[23] Yu, Q., Liu, X., Bouguettaya, A., Medjahed, B., 2008. Deploying and managing web services: issues, solutions, and directions. The VLDB JournalâĂŤThe International Journal on Very Large Data Bases 17 (3), 537–572.

[24] Zeng, L., Benatallah, B., Ngu, A. H., Dumas, M., Kalagnanam, J., Chang, H., 2004. Qos-aware middleware for web services composition. Software Engineering, IEEE Transactions on 30 (5), 311–327.

[25] Zhang, K., Shasha, D., 1989. Simple fast algorithms for the editing distance between trees and related problems. SIAM journal on computing 18 (6), 1245–1262.