A Notification Management Architecture for Service Co-evolution in the Internet of Things

Huu Tam Tran*, Harun Baraki*, Ramaprasad Kuppili*, Amir Taherkordi†, and Kurt Geihs*
* *Distributed Systems Group*
University of Kassel
Kassel, Germany
Email: {tran, baraki, kuppili}@vs.uni-kassel.de, geihs@uni-kassel.de
† *Networks and Distributed Systems Group*
University of Oslo
Oslo, Norway
Email: amirhost@ifi.uio.no

***Abstract*—IoT applications are essentially characterized by their highly dynamic nature, in which the configurations of both network and software systems may change. In the context of IoT software services, modifications can occur during application lifespan due to updates and amendments. Hence, third-party applications and services depending on the changed services need to take appropriate coordination and adaptation actions. Existing solutions do not focus on coordinated co-evolution within resource constrained environments. Addressing this issue in a resource-efficient manner is the objective of our proposed notification management mechanism. It detects and informs dependent third-party applications automatically about service changes and categorizes them into crucial and minor changes. Furthermore, it maintains a list of possibly affected clients without consuming further resources on the IoT device the service is running on. Special attention is also paid to the usability and integrability of our mechanism for clients as well as for providers.**

***Keywords*-IoT Services, Service Evolution, Internet of Things, Service Co-evolution, Semantic Web, Ontology**

I. Introduction

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 easier to communicate with and integrate into existing application environments [1], [2], [3], [4]. However, the management of IoT services and their interfaces will require new techniques due to resource constraints on IoT devices in terms of processing capacity, communication bandwidth, battery lifetime, and memory capacity. A critical research problem, in this context, is how to handle service changes for each service consumer and how to enable and facilitate end-user application updates in case the dependent clients provide services to other clients. Addressing the challenge of service changes means that service interface modifications require adaptations in participating parties to prevent outages and failures due to individual service modifications.

To address unpredictable side-effects in IoT service environments, we provide a notification management architecture that supports a coordinated co-evolution of services in order to reduce downtimes. In this research, the co-evolution paradigm in service-oriented computing stands for a coherent process of evolving and maintaining a service and its interdependent services through a series of explicit changes [5]. Such kind of service co-evolution requires adaptation steps taken by both service providers and service consumers.

In the scope of the PROSECCO[1] research, we develop a solution for coordinated service co-evolution through a design technique that equips every service with an agent, called EVA (Evolution Agent) that performs the service evolution in collaboration with other EVAs [6], [7], [8]. This work extends our Evolution Agents with change detection and notification capabilities and implements a service registry that cooperates with them.

The main contributions of this work are: 1) A change detection approach based on the analysis of modified service descriptions; 2) a notification management architecture supporting the co-evolution of IoT services; 3) a service registry based on ontologies; 4) a case study that shows the application of the proposed solution in a real word scenario.

The rest of the paper is organized as follows. In Section II, we depict a motivating example for change management in IoT environments. Section III presents the background of service co-evolution and an overview of our semantic service registry. In Section IV, we analyze service descriptions to detect the kind of changes. In Section V we present our proposed notification management. Section VI provides a case study for our approach. Section VII reviews the state of the art. Finally, some concluding remarks are given in Section VIII.

[1]Provisions for Service Co-Evolution: http://www.uni-kassel.de/eecs/fachgebiete/vs/research/prosecco.html

II. A Motivating Example

To illustrate the motivation for service co-evolution, we presented the EVA architecture in our previous work [6]. In the following scenario, we emphasize the important role of communication protocols when IoT service providers need to inform their own clients about changes.

The example in Figure 1 illustrates the problem in the context of an IoT scenario. A sensor node located at A (S1) is providing the temperature and humidity values via a REST service. Its clients may be servers, laptops, android phones or other devices. These different clients can access this service to use the data directly, like e.g. C3 and C4, or to process it and provide it to other clients, like e.g. client S2.

Someday, the temperature provider updates the service so that it does not deliver temperatures in Celsius but in Fahrenheit. Hence, it is necessary to check the effect of updates on clients in the network. The clients cannot replace the service by integrating other services since the other sensor nodes are deployed in other rooms and consequently measuring other locations. A solution could be provided by agents, namely EVAs. They emulate the old version of the service by combining the updated service S1 with another service S2 that converts Fahrenheit to Celsius. The client applications continue to work with the old version provided by the agent. This step circumvents interruptions and gains time for manual adaptions by developers. Nonetheless, affected clients still have to be informed about the significance and kind of changes automatically. A major challenge for service providers is to find the target clients to send the service change information to and to store and maintain the list of clients. The first idea would be to keep the list of clients on the IoT device exposing the changed service. Then, the clients will be informed by the device whenever a change takes place. This means that service providers need more storage and bandwidth, especially in case of a growing number of clients in the network as their addresses have to be stored so that change information can be sent to them. Furthermore, the kind of change and the affected clients have to be detected on the IoT device. However, with the resource limitations of typical IoT devices, it is not feasible to maintain and analyze such a growing list. The following sections present our proposed architecture that handles such scenarios.

III. Semantic Service Registry for Service Co-evolution

A. Coordinated Service Co-evolution

Services are bound to be updated due to amendments or refinements or to provide additional functionalities, with or without decommissioning the older versions of services. To prevent outages and failures by individual service modifications and updates, coordinated evolution is required in IoT systems. The term *service evolution* involves the deployment

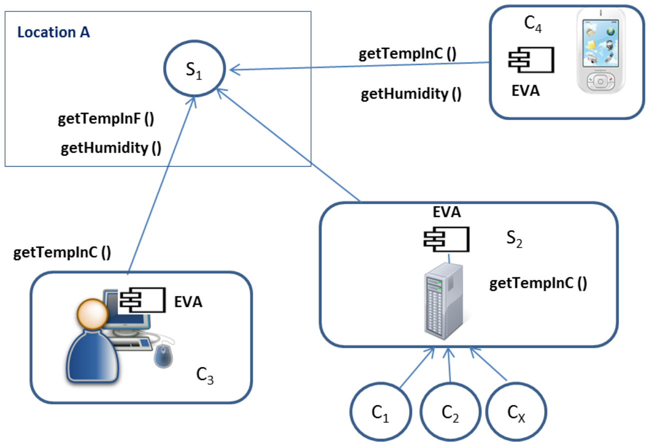

Figure 1. A motivation scenario in IoT environment

of a new service version which is caused by necessary changes in the interface structure, functionality, usage protocol, usage policies, business rules and regulations, etc. The objective is to automate the coordinated evolution as much as possible. For instance, if service S1 is used by client S2 and client S2 is providing a service to client Cx then it has to be verified if client Cx is also affected if service S1 is changed.

B. Evolution Agent

Sensors play a pivotal role in flooding the fast-paced developing world with their latest advancements and applications. Yet, many typical sensor devices, such as Zolertia Z1 motes, Arduino BT motes or Waspmotes possess severe memory constraints, featuring less than 10KB of RAM and 130 KB of ROM, resulting in limited computing power [4]. Thus, change management in IoT environments has to be endowed with resource-efficient mechanisms that relieve service providing IoT devices. For this reason, the idea of IoT devices being managed by an externally hosted Evolution Agent is considered. The EVA can be deployed to a powerful gateway node or in the Cloud. An EVA needs memory and storage to create, to distribute and to process incoming update messages. Therefore, it consists of components for Analysis, Evolution Analytics, and Evolution Coordination. Only the so-called Smart Update component [6] runs on the IoT devices and ensures that the external EVA can access and manage them. Hence, in the context of service co-evolution, each IoT system will base on the Cloud or other local computers to be able to cater for a growing number of clients in the network. One EVA may monitor and manage several IoT services belonging to one provider. The communication between clients and EVAs is performed via REST, which is explained in detail in Section V.

C. Ontology

In [9], an ontology is defined as an *explicit specification of conceptualization* that gives the relation between concepts, by strictly obeying a taxonomy, which is the hierarchy of those concepts. An ontology is based on a natural language by creating the nodes and edges, thus describing the taxonomical hierarchy of 'Classes' as nouns and their respective 'Properties' as verbs. Depending on the amount of data, ontologies can be created manually or automatically by using eXtensible Mark-up Language (XML), Resource Description Framework (RDF) and Web Ontology Language (OWL), which are rich in resource description. The primary components of an ontology include Classes (concepts), Individuals (objects or instances), Attributes (object description by relating them to other aspects) and Relations (relationship between objects of same or different classes). The ontology construction plays a pivotal role, taking different aspects into consideration like, acquiring the domain knowledge, designing the conceptual structure (hierarchy) and finally developing every bit and piece of each concept. In our work, we apply an extended version of the SSN (Semantic Sensor Network) ontology to describe and to search for suitable IoT services. In particular, we add the concept of communication protocols into the ontology to gain a loose link between the syntactical description of a service, e.g. WADL (Web Application Description Language) file, and its semantic description. The latter is specifically used to find appropriate services with respect to their semantic meaning. For instance, a sensing device for temperature (SSN ontology) can be represented as a resource in the WADL file that offers operations to read the current temperature and the average temperature of the last hour. Since the WADL file is linked in the semantic description, our service registry would also return the WADL file of this service, if a client would search through a SPARQL query for temperature sensing devices. Using the WADL file, the client can identify the concrete operations that are related to the temperature sensing device and filter out the operations of other sensing devices of the IoT device as these operations can be found under the according resource node in the WADL file.

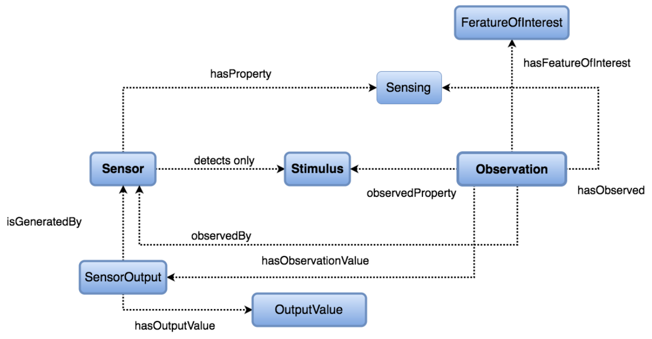

Figure 2. Stimulus-Sensor-Observation Path

Using the SSN Ontology aids the developer to access the information related to the sensor, e.g. accuracy, temperature range, etc. It is built with a pattern based on Ontology Design Pattern (ODP), which describes a Stimulus-Sensor-Observation (SSO) pattern. It is conceptually organized into 10 modules, consisting of 41 concepts and 39 objects inheriting from 11 DUL (DOLCE-UltraLite) concepts and 14 DUL object properties [9].

A SSO pattern is the central part of an SSO ontology, which connects sensors to their observations obeying the sensor perspectives. It consists of three primary nodes in a graph; namely Stimuli, Sensors and Observations.

- **Stimuli**: Stimulus is the change in the state of an environment, which is detected by a sensing activity.
- **Sensors**: A ssn:Sensor that senses or observes an environment and transforming (ssn:detects) it to ssn:SensorOutput.
- **Observations**: Observations are of primary importance, for which the sensors are activated with either the external (unnatural) or internal (natural) stimulus. The observations are the combination of the *act of sensing*, the *event* which is the stimulus (dul:includesEvent), the *sensor* (ssn:observedBy), a *method* (ssn:sensing MethodUsed), a *result* (ssn:observationResult), an *observed feature* (ssn:featureOfInterest) and an *observed property* (ssn:observedProperty) [9], as shown in Figure 2. Figure 3 shows the main excerpt of the ontology.

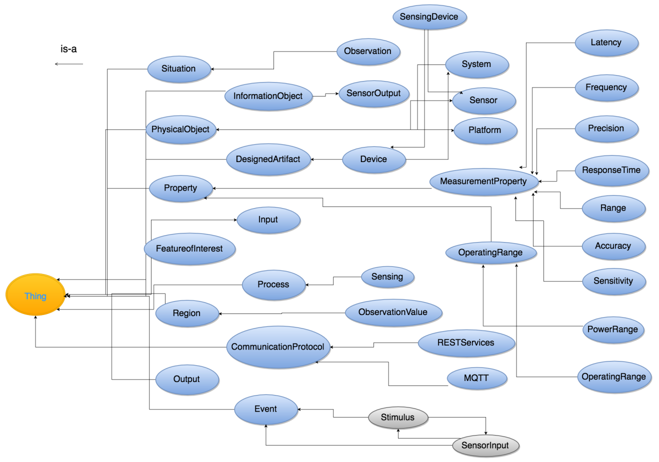

Figure 3. Ontology for IoT

D. Service Registry

The IoT provider node registers its services in the registry by submitting a semantic description and a link to the description of the communication protocol or concrete interface to access it (e.g. through REST). The registry is deployed in the Cloud. When a client wants to find and use a service, it formulates its search conditions as a SPARQL query and invokes the search operation of the registry's

REST interface. The registry checks its data set for all services that fulfill the search conditions. Then, the registry returns the relevant list of services and their descriptions to the client. However, they do not contain the real service addresses but the addresses of the responsible EVAs. This mechanism ensures that a client is contacting first the EVA of a service so that the EVA can register the client into its list before redirecting the client to the service endpoint. The IoT device only provides the service. We expect that clients have also an EVA installed. If not, they have to install a component that receives and depicts the messages sent by a remote EVA. In case of any service updates, the EVA of a provider can inform the clients about the relevant service changes from now on, as it has the required information about its clients.

IV. Change detection based on service descriptions

A. Analyzing service descriptions

In the further course of this paper, we assume that services are provided through REST and that in addition to the semantic description the WADL file is linked in the semantic description. WADL files are XML-based and specify the complete interface of a web service. They describe resources and all the operations that can be invoked on these resources through HTTP methods (e.g. GET, POST, PUT, and DELETE).

To inform about syntactical changes, the notification management detects changes based on comparing the WADL files of two consecutive service versions. In case it finds out differences between the WADL files, the clients that are registered for the affected operation by the provider's EVA will be informed.

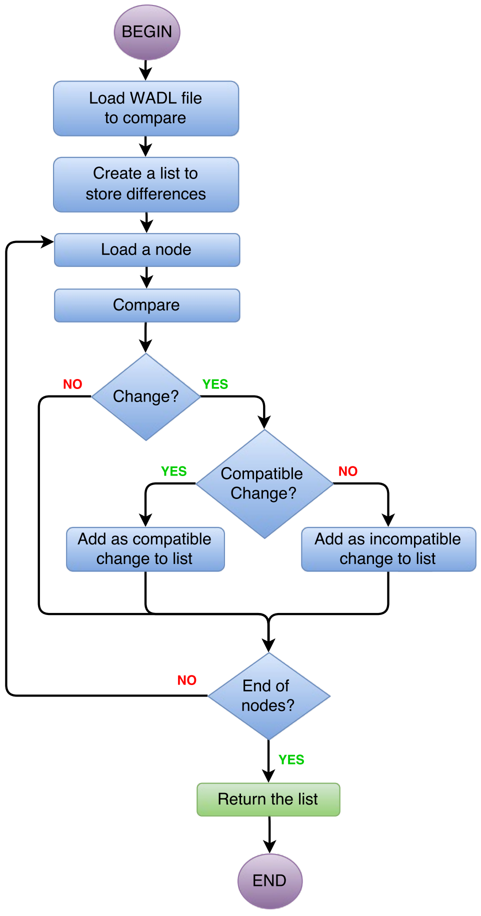
BEGIN
Load WADL file to compare
Create a list to store differences
Load a node
Compare
Change?
NO
YES
Compatible Change?
YES
NO
Add as compatible change to list
Add as incompatible change to list
End of nodes?
NO
YES
Return the list
END

Figure 4. Detection of changes based on service description

B. Change Detection

Figure 4 shows a flowchart explaining how the changes in two different WADL files of a given service are detected by parsing and analyzing them. The workflow is as follows: after a parser is created, both versions of the WADL description are loaded to compare the differences. Each entity in the description (e.g., ressources, data types) is considered as a *node*. To store the differences, a *list* is created beforehand to add the changes at the end. Both WADL files are provided to the StAX parser that successively reads each node from the first version and tries to detect it at the same position in the new version of the service. If any change is detected, the whole path to the node is stored in the list. Additionally, it will be noted whether the node was added, removed or updated. In case a data type was changed, the affected clients will be found out through the operation that uses the changed data type. Moreover, the detected change will be categorized into a compatible or incompatible change. According to Papazoglou et al. [10], compatible changes that do not affect a client's execution are the addition of new WSDL operations and the addition of new XML schema types that are not contained within previous types. Papazoglou et al. referred to WSDL files. However, it also holds true for WADL files. A further compatible change is the addition of new HTTP methods. Other kinds of changes are categorized as *incompatible changes* and are crucial for clients. In that case, the developer of the client application has to adapt the application manually or if available and possible, the EVA of the client will emulate the old version by combining the new service version with services for transforming back the changed elements.

After the final *node* is processed, the *list* is sent to the potentially affected clients. In case of changes in the semantic description of an IoT service, the same procedure is executed, except that all changes are categorized as incompatible changes. If a certain resource (e.g. sensing device) is affected by the change, the operations of that resource and thus the affected clients can be detected through the WADL file.

V. Proposed Notification Management

In this section we present the notification management architecture and, in particular, how the service registry, the service provider and clients are interacting with each other to keep clients informed about any relevant updates on the service side.

In our scenario the registry and IoT services can be accessed through REST interfaces. One of the major advantages of using a REST interface is that resources and data elements can be arbitrarily represented by means of various formats and, thus, enabling a lightweight communication compared to Web services. Further reasons for using REST interfaces are the uniform interface and the addressability which refer to the accessibility of addressable resources via uniform interfaces to clients, and statelessness which assures the action to requests based on the information available in the request, i.e. the server does not rely on the data from previous requests to perform the action for a new request [11], [3]. Thus, the REST interface can be considered as a possible solution for communication between IoT services in the coordinated service evolution.

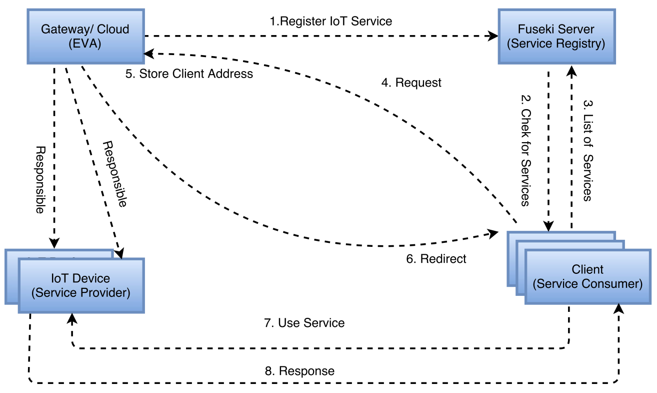

Figure 5. Proposed Notification Management for Service Co-evolution

In Figure 5, an EVA deployed on a gateway node and responsible for a group of IoT devices is depicted on the left-hand side. The services are provided by the IoT devices. However, the notification system is not deployed on the resource-constrained IoT devices but on an external host. This EVA is responsible for the services and service updates on the sensor nodes it is managing, but does not need to comply with their resource limitations in terms of memory, storage or processing power. Hence, it can handle a growing number of clients. The EVA has the information of the services provided by the IoT devices and registers these services through a REST interface at the Service Registry together with their semantic description files (step 1). This enables potential clients to search for concrete services and get a list of suitable IoT services (step 2). However, in our case, the Service Registry is not returning the address of the actual IoT service but the address of the EVA responsible for that IoT service (step 3). This enables the responsible EVA to take note of the client and register the client's address and the invoked operation in a list created for the actual IoT service (step 4 and 5). In step 6, the EVA on the server will redirect the client to the real IoT service so that the service is used directly (step 7 and 8). In this way, the IoT device does not need to maintain the list of clients.

The provider's EVA only informs those clients about updates on IoT services that are indeed affected. Furthermore, clients may also have clients which are using their services. As soon as a client is indirectly affected by a service update, it will be informed through this mechanism as well. Update notifications will encompass the whole chain of affected parties, so that in case of cycles the initial causing service provider recognizes that.

VI. Case Study

We have implemented the notification management architecture using EVAs on Apache Tomcat on Raspberry Pi 3, Apache Fuseki on a server, REST services on Zolertia Z1 motes and Android devices as clients. This section will explain details of our implementation.

In Figure 5, it can be seen that both the service provider and the client implement a REST interface in order to communicate with each other. The service is registered at the registry and hence it can be accessed by the clients. Whenever a client (e.g Android device in Figure 6) requests a service by specifying, e.g. the sensor type and region, the registry gives a list of providers offering a suitable service.

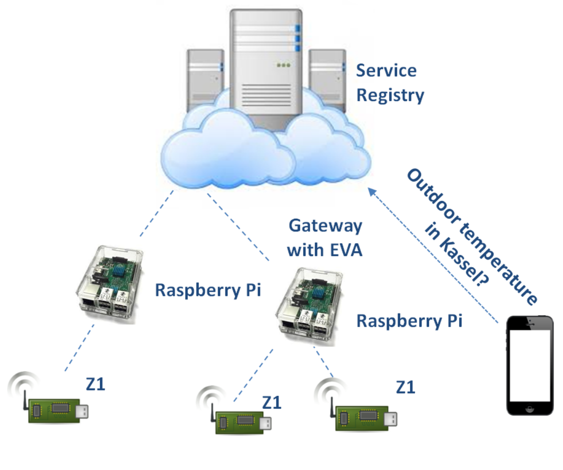

Figure 6. Implementation Scenario

Upon reception of response, the client requests the provider for the service. But, as the current provider is an EVA and not the real IoT device, the EVA adds the client to its list of clients for notifying it about any changes of the requested IoT service. The provider redirects the client to the real IoT service with a redirection message which is the HTTP redirection. Through HTTP redirection the client is notified about the address of the real IoT service and the client can finally use the requested service. In this way, the built-in constraints of IoT devices are faced with grouping several devices under one external EVA.

In step 5 of Figure 5, it can be seen that the client is redirected to the real IoT device by the EVA of the service provider. Upon receiving the permanent redirection message, the client has to replace the IP address of the service. Our scenario is depicted in Figure 6. Two Raspberry Pi 3 work as border routers, but have also EVAs installed. Zolertia Z1 motes provide different IoT services for measuring the temperature, humidity, and brightness. We changed during the experiment the return type of the temperature reading operation from float to String and passed the new service implementation to the responsible EVA of the IoT device. The EVA took as additional inputs the updated descriptions and executed the algorithm presented in section IV. The change detection identified the new return type as an incompatible change and noted down the operation and resource affected by the update. Then, the client on the Android device was informed about the incompatible change. Finally, the EVA deployed the new service implementation on the Zolertia Z1 motes.

The implementation has shown that our approach is a viable solution for IoT environments. Storing the list of clients and finding the affected target clients to send the service change information to were executed externally on powerful hosts. This notification management architecture plays an important part in coordinating services in IoT environments and will also be used in our future work when further ontologies and the implementation of the services are connected with each other.

VII. Related work

Service co-evolution can be seen as a special case of service evolution. It raised to a more and more important topic that brings many new challenges to software evolution. In the scope of Web services, researchers have spent significant effort on investigating methods and techniques for the management of service changes. Some researchers, for instance, Andrikopoulos and Papazoglou [12], [10] investigate service changes; S.Wang et al. [13] and A. Cicchetti et al. [14] perform compatibility and impact analysis; M. Romano et al. [15] and M. Fokaefs [16] et al. design tools which may detect or extract multiple changes based on WSDL files; P. Kaminski et al.[17] develop service adaptation techniques. However, there has been limited research addressing the issues of service co-evolution as well as its notification mechanisms in IoT systems. This section will present some frequently cited works related to our research.

One of the initial works handling the problem of service evolution is developed by Fokaefs et al. [16]. Their tool VTracker is designed to analyze the evolution of WSDL interfaces. 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. However, VTracker does not take into account the syntax of WSDL interfaces. 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 [15] presented an outstanding work called WSDLDiff 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 that is used to define the data types of the WSDL interface. Romano at al. refer to these changes as fine-grained changes. The results of this study showed that this approach is a useful means to understand how a particular Web service evolves over time. Nonetheless, the authors neither investigate WADL files nor addressed the coordinated evolution of different Web services that distinguishes from our approach.

Other well-known research results come from M.P. Papazoglou and V. Andrikopoulos [12], [10] 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 and deep changes for service compatibility and reasoning mechanisms for delimiting the effect of changes which can be kept local and consistent with a service description. However, the authors only deal with shallow changes. Additionally, their approach does not mention about the IoT environment and its services. Fokaefs et al. [18] provide an approach to deal with deep evolutionary changes since it automates the adaptations of the service clients across the co-evolution chain. Nonetheless, WSDarwin is a maintenance support tool that operates exclusively at development time.

In fact, there are many agent-based approaches available to support interoperable IoT devices and their services nowadays [19], [20]. 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 and requirements for participating service providers.

VIII. Conclusions and Future Work

This paper proposes a novel notification management architecture that detects service description changes and notifies all affected participating parties in the network. This is the first step to support affected peers to adapt to changes in coordinated service co-evolution scenarios.

The design of this architecture has taken into account the combination of service registries and the semantic and syntactical description and changes of services. The notification management architecture is implemented to cater growing dynamic networks in IoT, particularly for services exposed by resource-constrained devices.

The paper is a step forward to service co-evolution in IoT. In the future, we will further extend and improve in many different aspects such as extending the semantic service discovery and the notification management to more sensor devices and various kinds of IoT applications. In the current work, we also focused on the description files. However, including an abstract model of the implementation may lead in combination with the semantic and syntactic description to more valuable and precise statements regarding the significance of a change and the affected clients.

Acknowledgement

This is a work in progress supported by the German Research Foundation (DFG) under the project PROSECCO, grant number 5534111. The authors would like to thank the DFG for supporting their participation in worldwide research networks.

References

[1] G. Fortino, D. Parisi, V. Pirrone, and G. Di Fatta, "Bodycloud: A saas approach for community body sensor networks," *Future Generation Computer Systems*, vol. 35, pp. 62–79, 2014.

[2] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos, "Sensing as a service model for smart cities supported by internet of things," *Transactions on Emerging Telecommunications Technologies*, vol. 25, no. 1, pp. 81–93, 2014.

[3] K. Dar, A. Taherkordi, H. Baraki, F. Eliassen, and K. Geihs, "A resource oriented integration architecture for the internet of things: A business process perspective," *Pervasive and Mobile Computing*, vol. 20, pp. 145–159, 2015.

[4] X. T. Nguyen, H. T. Tran, H. Baraki, and K. Geihs, "Frasad: A framework for model-driven iot application development," in *Internet of Things (WF-IoT), 2015 IEEE 2nd World Forum on*. IEEE, 2015, pp. 387–392.

[5] M. P. Papazoglou, "The challenges of service evolution," in *Advanced Information Systems Engineering*. Springer, 2008, pp. 1–15.

[6] H. T. Tran, H. Baraki, and K. Geihs, "Service co-evolution in the internet of things," *EAI Endorsed Transactions on Cloud Systems*, vol. 15, no. 1, 2 2015. [Online]. Available: http://eudl.eu/doi/10.4108/cs.1.1.e5

[7] M. De Sanctis, K. Geihs, A. Bucchiarone, G. Valetto, A. Marconi, and M. Pistore, "Distributed service co-evolution based on domain objects," in *11th Int. Workshop on Engineering Service-Oriented Applications (WESOA'15), colocated with ICSOC 2015*. Springer Verlag, 2015.

[8] H. T. Tran, H. Baraki, and K. Geihs, "An approach towards a service co-evolution in the internet of things," in *Internet of Things. User-Centric IoT*. Springer, 2014, pp. 273–280.

[9] M. Compton, P. Barnaghi, L. Bermudez, R. GarcíA-Castro, O. Corcho, S. Cox, J. Graybeal, M. Hauswirth, C. Henson, A. Herzog *et al.*, "The ssn ontology of the w3c semantic sensor network incubator group," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 17, pp. 25–32, 2012.

[10] M. P. Papazoglou, V. Andrikopoulos, and S. Benbernou, "Managing evolving services," *Software, IEEE*, vol. 28, no. 3, pp. 49–55, 2011.

[11] D. Pfisterer, K. Römer, D. Bimschas, O. Kleine, R. Mietz, C. Truong, H. Hasemann, A. Kröller, M. Pagel, M. Hauswirth *et al.*, "Spitfire: toward a semantic web of things." *IEEE Communications Magazine*, vol. 49, no. 11, pp. 40–48, 2011.

[12] V. Andrikopoulos, S. Benbernou, and M. P. Papazoglou, "On the evolution of services," *Software Engineering, IEEE Transactions on*, vol. 38, no. 3, pp. 609–628, 2012.

[13] S. Wang and M. A. Capretz, "A dependency impact analysis model for web services evolution," in *Web Services, 2009. ICWS 2009. IEEE International Conference on*. IEEE, 2009, pp. 359–365.

[14] A. Cicchetti, D. Di Ruscio, and A. Pierantonio, "Managing dependent changes in coupled evolution," in *Theory and Practice of Model Transformations*. Springer, 2009, pp. 35–51.

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

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

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

[18] M. Fokaefs and E. Stroulia, "Wsdarwin: Studying the evolution of web service systems," in *Advanced Web Services*. Springer, 2014, pp. 199–223.

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

[20] H. Yu, Z. Shen, and C. Leung, "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, 2013, pp. 1054–1057.