DECOM: A framework to support evolution of IoT services

Huu Tam Tran
Distributed Systems Group
University of Kassel
Kassel, Germany
tran@vs.uni-kassel.de

Alexander Jahl
Distributed Systems Group
University of Kassel
Kassel, Germany
jahl@vs.uni-kassel.de

Kurt Geihs
Distributed Systems Group
University of Kassel
Kassel, Germany
geihs@vs.uni-kassel.de

Ramaprasad Kuppili
Distributed Systems Group
University of Kassel
Kassel, Germany
kuppili@vs.uni-kassel.de

Xuan Thang Nguyen
Faculty of Information Technology
Hanoi University
Hanoi, Vietnam
nxthang@hnu.edu.vn

Thi Thanh Binh Huynh
Hanoi University of Science and Technology
Hanoi, Vietnam
binhht@soict.hust.edu.vn

ABSTRACT

In the heterogeneous and dynamic Internet of Things (IoT), applications and services are frequently subject to change for various reasons such as maintaining their functionality, reliability, availability, and performance. Detecting and communicating these changes are still performed manually by responsible developers and administrators. Such a mechanism will not be adequate anymore in the future of large-scale IoT environments. Therefore, we present a comprehensive framework named DECOM for automatic detection and communication of service changes. Here, we assume that capabilities and interfaces of IoT devices are described and provided through REST services. To be able to detect syntactic as well as semantic changes, we transform an extended version of the interface description into a logic program and apply a sequence of analysis steps to detect changes. The feasibility and applicability of the framework are demonstrated in an IoT application scenario.

CCS CONCEPTS
- Software and its engineering → Development frameworks and environments; Software evolution;

KEYWORDS

ACM Reference Format:

1 INTRODUCTION

The envisioned Internet of Things (IoT) foresees a future Internet that incorporates smart physical objects as IoT services [4, 18]. This service-based integration of IoT will be smarter, easier to communicate with, and to be integrated into existing application environments [19]. However, the management of IoT services requires new techniques due to resource constraints on IoT devices regarding processing capacity, communication bandwidth, memory capacity [12, 13, 18].

As all software, IoT services evolve over time to include enhancements and to increase their value to meet the requirements of their service consumers. Often service providers undergo necessary updates like supporting new technologies, discharging obsolete functionalities, bug fixes, and improvement in the quality of the provided service. When a service changes, the client that depends on this service also needs to adapt. However, a service may be modified without notification and updates may lead to service interruptions [19].

Existing approaches like Web Ontology Language (OWL)[3] have some limitations. In particular, OWL does not provide support for general purpose rules, for example, integrity constraints and closed world reasoning, which are seen as an important paradigm in knowledge representation. Furthermore, OWL is monotonic. This means that a referenced ontology can be adjusted by other parties, but the OWL ontologies cannot be used as a common language [2]. Additionally, the efforts for creating and processing semantic service descriptions are time-consuming and require a deep knowledge of the applied logic, descriptions and tools. To deal with these problems, the paper proposes the use of Answer Set Programming (ASP) [7, 8] that has provided many advantages, such as updating existing rules or overwriting initial default assumptions.

Therefore, this paper presents first a comprehensive method to describe services for dynamic and heterogeneous IoT environments. It is based on an extended Web Application
Description Language (WADL) [6] specification for REST services. Our framework is enabled to transform it into a logic program based on ASP. This allows a detailed detection and analysis of compatible and incompatible changes and whether the changes are of semantic or syntactic nature. Besides, ASP is non-monotonic and supports so-called defaults which allows service providers to refer to the same default knowledge. On the other hand, it is also capable of overriding parts and adapting to details without breaching consistency [2, 7].

Secondly, this paper presents and demonstrates automatic detection of syntactic and semantic changes. After a successful detection, the list of changes is transformed into an XML format and sent to affected clients for decision-making purposes. Examples of other works that demonstrate how affected clients could be detected and notified in IoT environments can be seen in [19].

Our evaluation examines the run-time performance and proves the suitability for highly dynamic environments. Thus, the main contributions of this work are: (1) an approach to describe IoT services semantically and syntactically with support for a shared knowledge base; (2) a comprehensive change detection to enhance the reliability of the heterogeneous and dynamic IoT service environments; (3) an evaluation with realistic scenarios to measure the run-time performance and to prove the practicability of our approach.

The rest of the paper is organized as follows. Section 2 introduces an example scenario with service changes. Section 3 presents an overview of the techniques used in our solution. Section 4 explains our proposed framework. In Section 5, we describe our implementation as well as its evaluation. Section 6 discusses related work. Finally, some concluding remarks are given in Section 7.

2 MOTIVATING EXAMPLE

2.1 FitService

Let us consider a Health Care application named FitService which is presented in [18] as a product of HCC [18] and is implemented as a RESTful Web service. The aim of this application is to provide health-related services to its customers. The application provides indicators such as heart rate, basal body temperature and respiratory rate besides some flourishing value-added services like nutrition-related information, health specialists, and training plans provided by third-party services.

Figure 1 shows a flowchart diagram for the FitService application. The green nodes represent the invocation of third-party services. This scenario can be well established by deploying some smart devices providing the functionalities to consumers. In case there is any disturbance in the normal health readings, warnings are displayed to the user.

Let us assume that in order to improve the functionality provided by the third parties, any of these REST services could evolve independently without notifying HCC. In this case, the entire application disrupts, making the application unusable. Hence, changes would indirectly affect the customer applications, making FitService not trustworthy anymore. Thus, a service provider should notify its clients in time and in a precise way about the change from the existing version to the updated one so that the clients can have a grace period to switch over to the new service.

The following section explains by means of the FitService workflow the different types of changes that we will consider. Such changes can also appear in other direct and indirect service layers and affect the customer.

2.2 Change Scenarios

2.2.1 Change the semantics of WeighType: let us consider that the FitService extends its services in an international market and thus it has undergone a few changes in its functionality. Without any prior notification, FitService evolves into its new version. In this version, FitService asks its users to input the weight value in kilo and not in pound anymore. However, this change does not interrupt the clients, but it provides unexpected results and may consider as a simple change of semantics of WeighType. Listing 1 shows the changes in the related WADL file.
2.2.2 Addition of new service: FitService has decided to include a new service called ‘Nutrition’ to its already existing service without any other changes to it. Listing 2 shows the newly added nutrition service in the WADL data. This is to satisfy its clients who could also be health conscious wanting to know the kind of diet they have to maintain. For the new service, the manager of this service also increases the cost of subscription for its premium customers.

Listing 2: Adding Nutrition service

...<resource path="nutrition" method="GET"> <representation mediaType="application/json"> </resource> ...

These scenarios are used in the following to show how a service can undergo changes and subsequently how these changes are described, detected and propagated to the dependent clients.

3 BACKGROUND

3.1 Answer Set Programming

ASP is a declarative approach for solving difficult (primarily NP hard) search problems. It is based on the stable model semantics of logic programming. The idea of ASP is to represent a given computational problem by a logic program whose answer sets correspond to solutions and then use an ASP solver to find a set of minimal answers for this program [8, 14].

The major advantage of ASP is that we only need to define the problem and constraints. In general, processing ASP mainly involves two steps namely Grounding and Solving [8]. Grounding eliminates the variables by constructing smartly the collection of relevant ground rules while Solving exploits search techniques similar to those used by SAT solvers to find the answer sets of the resulting grounded program.

In our approach, the main reason for using ASP instead of OWL is that ASP supports integrity constraints and closed world reasoning. Other frameworks, for example, Web service execution environment [9] combine OWL with rule-based formalisms grounded in logic programming. However, logical conclusions that an OWL reasoner would draw from an ontology differ from those that would be obtained when using a logic program engine [2]. In facts, the code of ASP is logic programming providing defaults for expressing standard representations. This feature of ASP provides various benefits for service vendors and participating parties in dynamic and heterogeneous environments [2].

3.2 RESTful Services

RESTful services are syntactically described using the XML-based WADL that defines the complete interface of a web service. Interface definitions describe resources and all the operations that can be invoked on these resources through HTTP methods (e.g., GET, POST, DELETE, and PUT) [6]. WADL is a simple interface representation. It provides less verbose information for easier reading than the Web Service Description Language (WSDL) [6, 19]. In our scenario, IoT services can be accessed through REST interfaces. The resources and data elements can be arbitrarily represented using various formats and, thus, enabling a lightweight communication compared to Web services. In fact, WADL data is usually rich in syntax but not in semantics. Therefore, for successful service discovery and change detection, we add semantic annotations to the WADL data.

4 DETECTING CHANGES

4.1 DECOM Framework

In the scope of this work, a precise detection means that not only structural changes like the addition or removal of a function are detected but also semantic changes which are not reflected by the structure. For example, a return value may keep its data type Float but change its interpretation from the imperial system to the metric system. This section presents the DECOM framework to detect syntactic and semantic service changes by describing these services before and after the evolution.

Figure 2 shows how to detect changes in our framework involving four stages. The first stage describes RESTful services by using WADL. The second stage annotates semantically RESTful services. This stage adds change information of services into WADL descriptions. The third stage generates an ASP program based on the extended WADL description by XSLT 1. The last stage detects changes by means of ASP Query.

1 https://www.w3.org/TR/xslt/, visited last on 15th September 2018
4.2 Semantic Annotation

For a successful service discovery and the change management, the semantic annotations are to be added to the data. For this, the concept of semantic annotations for WSDL called SAWSDL [16] is considered. Furthermore, our framework uses declarative logic programs formulated in ASP since it may support the demanded unique name assumption which is needed to automatically derive ASP code from the interface description.

```
Listing 3: Example of Semantic annotation in WSDL file
...
<resource path="user">
  <method id="createUser" name="POST">
    <request>
      <ns:representation asp:conceptReference=
"http://.../asp/Fitness#Customer" element=
"Customer" mediaType="application/json">
    </request>
  </method>
</resource>
...
```

```
Listing 4: Semantic annotation for Nutrition service
...
<resource path="nutrition">
  <method id="getNutrition" name="GET">
    <response>
      <representation asp:conceptReference=
"http://.../asp/Fitness#Macro" element=
"Macro" mediaType="application/json">
    </response>
  </method>
</resource>
...
```

Coming back to our example, we assume that the FitService is implemented by using RESTful Web services. In our scenario, the WADL description is annotated before the service evolution of FitService. This stage can be done by using an XSL Transformation for instance. A detailed example can be seen on this URL². Extending WADL with semantic annotations to the service description is performed by adding the attribute asp:conceptReference as seen in Listing 3. The customer resource is annotated because this resource later undergoes a change in WeightType. The prefix asp refers to the ASP description of the Web service, it can be reflected in the namespace definitions of WADL, as shown in Listing 3.

The second change scenario can be observed in Listing 5 and 6 below. It can also be observed in its corresponding XML Schema Definition (XSD). The element Macro is semantically annotated (shown in Listing 4 and 5).

After the addition of semantic annotations to both versions of the FitService's WADL descriptions, the next stage is to generate the ASP description from the extended WADL, as discussed in the following section.

4.3 Generate ASP Description

WADL is XML-based and thus can be considered as a tree structure with nodes representing the 'elements' and edges representing the 'relations'. Thus, XSLT is used as the transformation language to extract the element values.

```
Listing 5: Semantic Annotation for Nutrition service
...
<xs:element name="Customer" type="tns:CustomerType"/>
<xs:element name="CustomerResults" asp:conceptReference=
"http://.../asp/Fitness#CustomerResults" type="tns:CustomerResultsType"/>
<xs:element name="Macro" asp:conceptReference=
"http://.../asp/Fitness#Macro" type="tns:MacroType"/>
...
<xs:complexType name="MacroType">
<xs:sequence>
<xs:element name="carbs" type="xs:float"/>
<xs:element name="fat" type="xs:float"/>
<xs:element name="protein" type="xs:float"/>
</xs:sequence>
</xs:complexType>
</xs:schema>
...
```

The following listings show the basic XSL Transformations applied on the target XML data, besides providing the corresponding output in 'text' format. For generating ASP from the WADL data, it is noted that all datatype attribute values

²http://xslttransform.net/6pS1zDo, visited last on 15th September 2018
are taken as defaults in the Default Knowledge Base (DKB) that is created by the developer.

Listing 6: WADL resource

```xml
...<resource path="/auth"/>
<resource path="/login"/>
<method id="login" name="POST"/>
<request>
<param xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" name="login" style="query" type="xs:string"/>
<param xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" name="password" style="query" type="xs:string"/>
</request>
</response>
<representation mediaType="text/plain"/>
</response>
</method>
</resource>
...
```

Listing 7 consists of various default statements for some of the property values. The rest of the WADL data is realized by using XSLT (see Listing 6 and 8). The result in Listing 9 of the XSLT transformation of the WADL description presents the declaration of facts in ASP.

Listing 7: Example defaults for DKB

```plaintext
...is_a(X,Y) :- property(X),not d(X,Y),not ~is_a(X,Y).
d(X,Y) :- property(X),X=kilo,Y=name(N1).
...is_a(X,Y) :- property(X),not d(X,Y),not ~is_a(X,Y).
d(X,Y) :- property(X),X=number,Y=type(T1).
is_a(X,Y) :- property(X),not d(X,Y),not ~is_a(X,Y).
d(X,Y) :- property(X),X=number,Y=type(T2).
is_a(X,Y) :- property(X),not d(X,Y),not ~is_a(X,Y).
d(X,Y) :- property(X),X=basalbtemp,Y=name(N2).
...```

Listing 8: Example for transformation in XSLT

```xml
<xsl:output encoding="UTF-8" indent="yes" method="text" omit-xml-declaration="yes"/> 
<xsl:template match="/"/> 
<xsl:for-each select="ns:application/ns:resources/ns:resource"> 
  property(xsl:value-of select="substring-after(@path, '/')").
  object(xsl:value-of select="name()")
  is_a(xsl:value-of select="@path")
</xsl:for-each> 
</xsl:stylesheet>
```

4.4 Service Change Definition

Consider that FitService has upgraded its service that includes two scenarios as described in Section 2. These are (i) change in the semantics and (ii) addition of a new service.

Listing 9: XSLT Output in ASP

```plaintext
...object(resource).
is_a(auth.path(P1)).
property(login).
object(resource).
is_a(login.path(P2)).
...
```

(i) Semantic change: Listing 10 shows a snippet of a logical description, realized using XSLT Transformation. It consists of a change in WeightType from pound to kilo. The arguments of every predicate are well described as facts.

Listing 10: Change in WeightType as kilo

```plaintext
...is_a(weight, name(N)).
is_a(WeightType, type(T)).
has(sequence(S), element(E)).
has(element(E), name(B)).
has(element(E), type(T))
...```

By representing the service description using ASP, the KB [8] consisting of unconditional facts cannot be overwritten. In order to change a fact representing a certain element with its attribute value in the knowledge base, defaults are used to add some additional facts to the KB as shown in Listing 10.

In the scope of service evolution, a default statement for name(Y) value is defined as X in the DKB. And the knowledge representation for the WADL data before the service change can import the DKB. This can be represented as shown in Listing 11.

Listing 11: Default statement and change in WeightType - X

```plaintext
// Default statement
is_a(X, name(Y)) :- property(X), not d(X, name(Y)),
not ~is_a(X, name(Y)).
d(X, name(Y)) :- property(X), X=kilo.
...
// Change in WeightType - X
# include "Fitness.lp".
# include "Defaults.lp".
is_a(pound, X(Y)).
...```

Consider that a service provider has updated the service, changing WeightType from pound to kilo, it is then simpler to add this fact to the new ASP program, and by using
include statements to import all the rules to the new ASP program and also the DKB, besides adding the new facts about the service. This aspect of service update can be seen in Listing 11.

(ii) Service addition: The addition of a new service ‘Nutrition’ that can be realized by XSLT as showed in Listing 12.

Listing 12: Addition of new service for element

```asciidoc
... has(sequence(x), element(A..C)).
has(element(A), name(A); element(A), type(A)).
...
has(element(C), name(C); element(C), type(C)).
is_a(macroType, complexType(x)).
is_a(carbs, name(A)).
is_a(float, type(A)).
...
```

5 IMPLEMENTATION AND EVALUATION

This section presents solutions for detecting the changes to the affected clients. It is furnished with an evaluation setup in Section 5.1 required to implement the approach. In Section 5.2, the evaluation results are provided that show the service evolution successfully performed on the WADL data by using the ASP Queries to figure out the differences between the two versions of the WADL before and after the change of services. Section 5.3 provides a performance analysis of the ASP Queries.

5.1 EVALUATION Setup

The following evaluations were performed on a PC, with 2.5 GHz Intel Core i5 processor and 16 GB 1600 MHz DDR3 RAM by utilizing Clingo version 5.2 [7]

5.2 ASP Queries and Results

The setup discussed above forms the basis for the implementation of the prototype for our service change detection. For this evaluation, the following steps are considered to find out the solution. It involves:

- The logical WADL description of the Fitness service (before the change) saved under the filename Fitness.lp is taken into account, having the imported DKB Defaults.lp that consists of all the default statements for the data objects and properties. Fitness.lp also consists of FitnessX.lp, the logical description for FitService XSD.
- The new logical description FitnessNew.lp of the web service description (after the change) has to import all the facts declared in Fitness.lp along with the DKB Defaults.lp and the new FitService XSD saved under the filename FitnessNewX.lp.
- In order to write an ASP Query to find out the service changes, an important step is now to change all the predicates of FitnessNew.lp and FitnessNewX.lp to considering the predicates in Fitness.lp and FitnessX.lp.
- By using the keyword #include which sets all the atoms of Fitness.lp, FitnessX.lp, Defaults.lp, FitnessNew.lp and FitnessNewX.lp together.
- If required, update the DKB Defaults.lp with some additional rules and defaults, which could be the result of newly added atoms in FitnessNew.lp and FitnessNewX.lp.

Listing 13: An example of ASP Query for multiple changes

```asciidoc
... %----FitnessX.lp
property(kilo; liter). object(name(X); name(Y)).
is_a(float, type(X)).
-is_a(P, X) :- is_a(P0, X), property(P), P0 != P.
%----Defaults.lp
is_a(X, name(X)) :- property(X), not d(X, name(X)),
not -is_a(X, name(X)).
d(X, name(X)) :- property(X), X=kilo.
is_a(X, name(Y)) :- property(X), not d(X, name(Y)),
not -is_a(X, name(Y)).
d(X, name(Y)) :- property(X), X=liter.
%----FitnessNewX.lp
property(pound; millil). is_a_n(pound, name(1)).
is_a_n(millil, name(1)).
%----ASP Query
delta(A, B) :- is_a_n(A, B), not d(A, B).
delta(A, B) :- not is_a_n(A, B), d(A, B).
#show delta/2.
```

Before proceeding further with the ASP Queries, the predicate changes in FitnessNew.lp and FitnessNewX.lp are provided. It is achieved with a piece of Java program, which takes every text line in ASP consisting of a predicate as a string.

After creating new predicates of FitnessNew.lp and FitnessNewX.lp, we compare all the atoms of the respective logical descriptions with their corresponding peers of Fitness.lp and FitnessX.lp for both the scenarios. Thereby the Clingo solver is used to execute these queries.

Listing 14: Clingo solver terminal output for QueryX

```asciidoc
... clingo version 5.2.0
Reading from QueryX.lp
Solving...
Answer: 1
delta(pound, name(X)) delta(kilo, name(X))
delta(millil, name(Y)) delta(liter, name(Y))
SATISFIABLE
```

The DKB Defaults.lp can consist of ‘d’ number of defaults corresponding to predicates of either the property or object. This ASP query is more advantageous if it has more defaults.
These queries can be performed by changing multiple numbers of property values in our scenario and find out exactly the differences between the atoms of both FitnessX.lp and FitnessNewX.lp. To depict this, we randomly added two new properties and objects to the logical descriptions and a new default statement in Defaults.lp. The results can be observed in Listing 13 and 14.

5.3 Evaluation

In this section, the test analysis is performed for two kinds of service change scenarios. In the first scenario, the number of service changes has increased from 1 new service to 128 new services. In the second scenario, new semantic changes have been supplemented from one new semantic change to 256 new semantic changes. Then we examine the runtime occurred at both scenarios respectively.

Figure 3 shows the results of average run-time in the first scenario. As can be seen in the figure, with 128 new service changes, there is a slight increment in the average runtime which is 12.67 ms. This is a feasible and acceptable time for managing IoT services.

Figure 4 shows the results of the average run-time in the second scenario. The average run-time increases to around 6.3 ms with 256 new concepts as shown in Figure 4. This result illustrates the significant variation of the standard deviation.

To summarize the performances of the above scenarios, the increment of changes shows an impact in the performance of the Clingo solver. After the successful detection of changes, the ASP Query output consisting of a list of changes is transformed into an XML format that is realized by affected clients later. The transformed list of changes in an XML format is then sent to the affected clients via a notification management like in [19].

6 RELATED WORK

Providing an automated approach for analyzing IoT application changes during the application life-cycle is essential for enhancing the reliability of IoT infrastructures both at design and implementation levels. This section will review some frequently cited works in the literature regarding service change detection.

Fokasif and his colleagues [5] worked on analyzing WSDL interfaces by building a tool called VTracker to find out the differences between WSDL specifications. 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. Besides, this approach of transforming a WSDL interface into a simplified representation can lead to unprecise detection results.

Similarly, D. Romano et al. [17] presented a major contribution with the WSDLDiff tool to find out the fine-grained changes in the WSDL descriptions, by comparing the subsequent versions of the WSDL. Unlike VTracker, the framework depends on the schema used to define the data types in the WSDL, besides the syntax of the description language used to extract the changes. Later in other works, M. Fokasif et al. [6] proposed the WSDarwin tool to recognize the changes in the specification of a service. Unlike VTracker that finds the differences between any pair of XML documents by comparing all elements with each other and thus establishing the mapping based on their structural similarity. In these works, the authors did not consider service changes with respect to the semantic aspects that differs our approach.

Other well-known research results come from M.P. Papa-zoglou and V. Andrikopoulos [1, 15] with analyzing shallow changes and deep changes. Shallow changes refer small-scale, incremental changes that are localized to a service while deep changes are large-scale, transformational changes cascading beyond the consumers of service possible to consumers of an entire end-to-end service chain. In their work, they developed a set of theories and modes that unify different aspects of services such as description, compatibility to assist service developers in controlling and managing service changes. Their approach does not focus on IoT domain; however, many lessons can get from their works.
REFERENCES


7 CONCLUSION AND FUTURE WORK

Awareness of service changes is an essential ingredient of a reliable IoT environment. As changes are unavoidable, appropriate mechanisms are required to detect and to communicate them precisely. In this paper, we present a comprehensive framework called DECOM to describe and detect changes in IoT services by using Answer Set Programming. Two scenarios are investigated in this paper, changes in the semantics at the business layer and the instance at the service layer. Additionally, our evaluation examines the runtime performance and proves the suitability for highly dynamic environments. Thus, this research plays an important role to assist IoT developers in designing and implementing IoT services.

Concerning the future work, we are in the process of adding on-the-fly decision-making and planning of service evolution in the realm of Smart Cities.

8 ACKNOWLEDGMENTS

This work has been supported by a collaboration project funded by DFG/Germany under grant number GE776/20-1 and NAFOSTED/Vietnam under grant number DFG 102.01-2016.03.