An ontology-based dynamic service composition framework for intelligent houses

Matei Popovici, Mihnea Muraru, Alexandru Agache,
Lorina Negreanu, Cristian Giumale and Ciprian Dobre
Computer Science Department
Politehnica University of Bucharest, Romania
{*pdmatei, mmihnea, alexandruag, lorina.negreanu, cristian.giumale, cipsmm*}*@gmail.com*

***Abstract*—In order to improve users ability to interact with devices in an intelligent house, we propose a platform for intelligent device composition, based on Service Oriented Architecture. In such an environment, devices can be controlled and monitored using Service invocations, or they can interoperate in order to fulfil complex tasks, using Service composition. As part of our platform, we define a language for dynamic service composition, called MetaBPEL. It extends the WS-BPEL 2.0 language with semantic information, and acts as an abstract workflow definition mechanism. Unlike BPEL and other conventional composition languages, MetaBPEL does not bind to actual service implementations, but merely describes capabilities that services must have, in order to participate in a workflow. This feature allows both workflow migration between different environments as well as service replacement without modifying the composition scheme. In the article, we describe the MetaBPEL structure and the associated creation, generalization and instantiation mechanisms.**

***Index Terms*—dynamic service composition, semantic matching, ontology, abstract workflow, semantic workflow**

I. Introduction

House intelligence becomes important in our day-to-day lives, reflecting not just an ever increasing need for comfort (easy control and interaction with house appliances) but also the necessity to optimize the consumption of available resources: electricity, water etc. As devices become more specialized, and functionalities more complex, users might look for assistance from intelligent systems in order to take decisions about their home environment. Our approach creates a realistic basis for this target.

Existing intelligent systems for houses [1], [2] only manage to offer a certain degree of device automation: sound, light, blinds and stores systems, motion and light detectors. They offer only a small set of predefined functionalities and adding new, user-defined features is either impossible, or severely restricted. The integration of new devices with existing house systems is another problem insufficiently addressed.

We introduce a flexible Service Oriented Architecture (SOA) platform for controlling and coordinating devices. Each controllable device from a house environment is modeled as a Web Service. Functionalities such as pressing a button are modeled as Web service invocations of particular operations. Such invocations can produce side-effects (such as the actual room cooling or heating), and can return a value (such as the current room temperature).

Fig. 1 presents how the proposed solution differs from traditional device controlling methods. A user typically controls a device using hardware controlling interfaces (remote control, Fig. 1 - left). There are several successful industrial standardization efforts to automate the process. However, such solutions are based on low-level, device-specific communication standards (e.g., BACnet [3]). In our view, creating device web services is an approach that would eventually lead to standardization. This conforms with the vision expressed in existing standards such as BACnet that plan to provide WSDL [4] descriptions for BACnet compliant devices. A high-level approach based on SOA has several advantages. It (1) isolates the actual communication issue to specific adapters incorporated in Web Services. It also (2) allows direct and remote device control and monitoring, by means of Web Service invocations. Also, (3) allows aggregation of multiple house devices based on Web Service composition.

In order to allow users to define complex, context-specific, device behavior (e.g., air conditioning devices and blinds or store systems could be used together, in a coordinated fashion, for house cooling), we propose a dynamic composition mechanism for device services. Existing dynamic composition methods focus on semantic annotation of services, service components (operations, messages, etc.) and on semantic representation of workflows [5]. However, they do not address the problem of workflow migration between different environments. They also do not facilitate simple, standardized

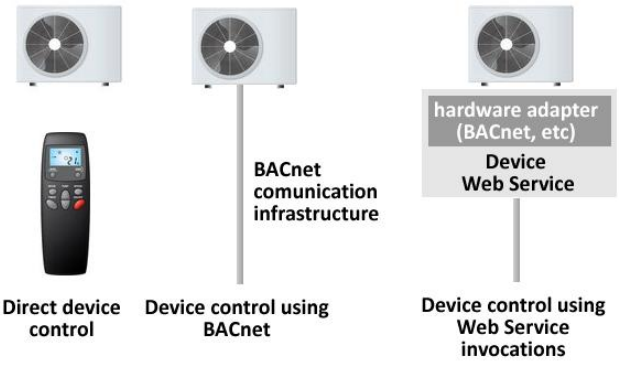

Fig. 1. Controlling devices

deployment of workflows.

The service composition method is based on the WS-BPEL 2.0 standard [6] (in short, BPEL). BPEL composition language has several limitations. By default, workflows refer actual service instances instead of the more general, service interfaces (or service types). This makes the replacement of services within a workflow hard to achieve without rewriting the entire workflow. It also leads to the problem of migrating a BPEL specification among different environments.

Such limitations are addressed by creating a more general composition language, called MetaBPEL. We describe the language and the mechanisms that allow dynamic service selection within a predefined control flow. The mechanism relies on an ontology that describes the services (or devices), service operations, and messages used for service communication. The ontology is used to construct a MetaBPEL specification, by replacing all references to actual services, operations, messages from a BPEL workflow, with their ontological counterparts. The resulting MetaBPEL specification can be created from regular BPEL documents along with appropriate ontological service descriptions, or it can be edited by users through specific graphical tools. The MetaBPEL specification can be easily migrated between different house environments, allowing for smooth device replacement.

The reverse process, generating the actual BPEL workflow from the abstract MetaBPEL description, also relies on the use of the ontology. This process uses an ontology-based matching mechanism to select the available services. An instantiation process creates a BPEL specification by replacing ontological service descriptions with actual service references. Further, the resulting BPEL can be executed on a service execution engine such as Apache ODE [7], ActiveBPEL and others. The approach is fully based on existing technology for composition, and it adds an ontological layer allowing service abstraction.

In Section II we compare our approach to existing dynamic composition techniques and to semantic workflow representation attempts. In Section III, we describe a Service Oriented Architecture that implements MetaBPEL. In Section IV we describe the ontology modeling approach for abstract services. In Section V we present the MetaBPEL structure, the creation and instantiation processes and an algorithm for MetaBPEL generalization that facilitates wider workflow reuse. Finally, in Section VI we describe our prototype implementation and initial results and in Section VII we present our conclusions and future work.

II. Related work

In [5] and [8], authors propose the use of a full ontological description of workflows as well as customized semantic execution engines for dynamic service composition. Workflows are described either as data models or "semantic graphs". In this case, the processes of reasoning and the actual execution of a workflow suffer from a high execution overhead because semantic processing is slow. Also, the workflow cannot be easily migrated because its execution is restricted to environments that run specialized semantic execution engines. In contrast, our approach produces standard BPEL specifications, able to run on existing BPEL execution engines.

Existing approaches that use standard composition languages (such as BPEL) are either limited to annotating BPEL descriptions, for workflow querying [9] or propose rigid matching techniques, where service compatibility is reduced to concept equality [10], [11].

Solutions such as [12] and [13] focus on QoS-based dynamic service composition, but fail to give expressive matching techniques. In an intelligent house scenario, the focus is less on QoS, since all services are locally accessible, but more on matching similar but not necessarily identical services.

Semantic service extensions, such as WSDL-S [14] have limits in extensively describing service concepts. WSDL-S annotates services with semantic information, but does not distinguish between service type (concept) and service implementation (individual). Similarly, in OWL-S [15] there is no distinction between an instantiated process (that refers actual composed services) and a conceptual process (where only service types should be enrolled). Custom, domain specific ontologies for workflows have been attempted in [9], [16] but they do not address the above issues.

Work has been done in ontological representation of devices [17] and on device-populated environments [18], but the issue of dynamic device composition is not fully addressed.

The usefulness of a MetaBPEL specification is twofold: on one hand, it serves as an ontological description of a workflow. From this point of view, it specifies what devices are part of the workflow, and what position they occupy. This is useful for both device and workflow-related queries, as it allows for identification of device types involved in a given composition process and, more importantly, finding certain composition processes that use given device types.

On the other hand, the MetaBPEL is a tool for dynamic service composition, seen as workflow sharing. Using the matching process and selecting adequate service instances, BPEL specifications can be generated and executed. By using BPEL, the execution process is completely independent of the ontological descriptions, and there is no need for an ontology-oriented execution engine, in order to implement dynamic composition.

III. The overall architecture

In Fig. 2, the overall architecture that implements MetaBPEL is shown. The Ontology Repository is a central component: it contains the entire device ontology that describes devices, operations, messages and measurement units. RDF [19] and RDF Schema [20] are used as an ontology description language. RDF has a simple XML structure and vocabulary. It also addresses semantics directly in terms of RDF triples. Several RDF-based query languages, such as SPARQL [21], are already available on the market.

Each published service is bound to an ontological description. The bindings are also defined using RDF. Furthermore,

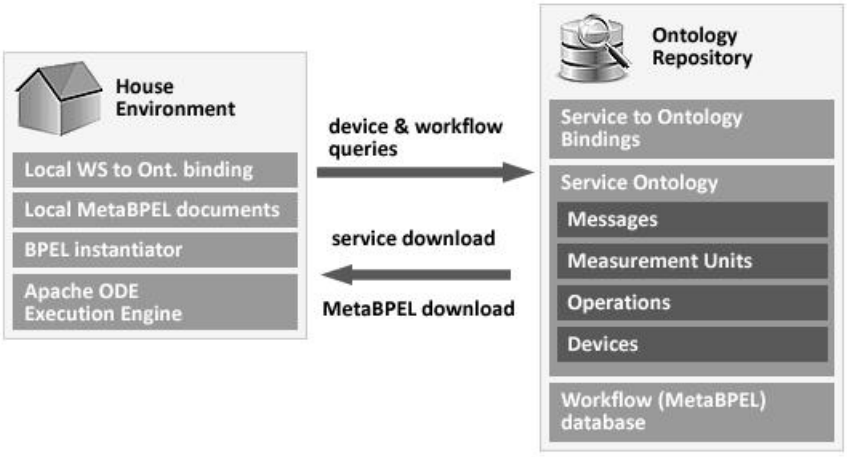

Fig. 2. Overall architecture

the Ontology Repository contains a set of MetaBPEL documents, published (and uploaded) by users.

In the house environment, bindings of locally available services to ontology descriptions are stored. A repository of local MetaBPEL documents, either created by user, or downloaded from the Ontology Repository is maintained as well. The BPEL Instantiator is responsible for: (1) semantic matching of actual (and available) services to a MetaBPEL document and (2) generation of operational code (BPEL), based on the MetaBPEL and the matched services.

The resulted BPEL is deployed and executed automatically using Apache ODE Execution Engine.

IV. Ontology

A. Ontology Description Language

We represent the ontology using RDF and RDF Schema. Using RDF, an example of concept definitions that are used in conjunction with MetaBPEL is shown in Fig. 3.

The concepts and relation defined in Fig. 3 correspond to the visual representation from Fig. 4.

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/
      22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/
      rdf-schema#">

  <rdfs:Class rdf:ID="Object" />
  <rdfs:Class rdf:ID="Device">
    <rdfs:subClassOf
        rdf:resource="#Object" />
  </rdfs:Class>
  <rdfs:Class rdf:ID="Functionality">
    <rdfs:subClassOf
        rdf:resource="#Object" />
  </rdfs:Class>
  <rdfs:Property
      rdf:ID="hasFunctionality">
    <rdfs:domain
        rdf:resource="#Device" />
    <rdfs:range
        rdf:resource="#Functionality" />
  </rdfs:Property>
</rdf:RDF>
```

Fig. 3. Sample RDF description of a device taxonomy

B. Ontology component for message descriptions

It is common for messages used by Services to be described according to a standard WSDL description. An example of such a message is depicted in Fig. 5.

These messages are referred in BPEL specifications when declaring variables, as message types. To connect the input of a service to the output of another, variables or variable parts are copied from source to destination.

Nonetheless, in a dynamic environment, services with similar capabilities might define messages in different ways. For example, different names for messages might be used, message parts may occur in different order. Still, in these cases the inherent semantics might remain the same.

For this reason we propose the creation of ontological message descriptions. Structurally, ontology message descriptions are similar to the ones present in WSDLs. Their role is to offer a common and abstract taxonomy for referring messages with equivalent structure. An example of such an ontological message description is presented in Fig. 6. Concrete WSDL messages must be mapped onto the corresponding ontological message description, as shown in Fig. 7.

C. Ontology component for measurement units

In the case of modeling devices as web services, the communication compatibility between services isn't restricted to message type compatibility. For example, assume we have two temperature sensors, each having a *getTemperature* operation, returning an integer value. Apparently, the operations associated with the two devices are equivalent: they have the

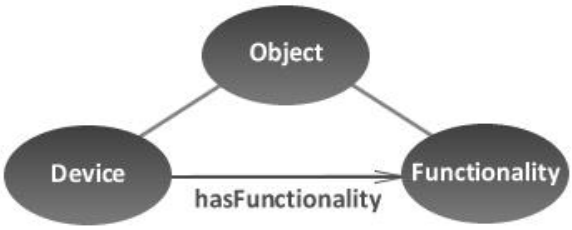

Fig. 4. Visual representation of a sample device taxonomy

```
<wsdl:message name="mess1">
 <wsdl:part name="input1" type="xsd:string" />
 <wsdl:part name="input2" type="xsd:string" />
</wsdl:message>
```

Fig. 5. WSDL message description

same functionality and they return the same message type (integer). But if one device uses Fahrenheit degrees and the other Celsius degrees, the incompatibility is obvious, and replacing one device with another will result in an invalid composition scheme with undesirable functionality.

As such, each ontological description of a message part will have a reference to the measurement unit it uses. The ontology message description will contain these references, as seen in Fig. 6. Message parts that contain abstract information such as device state (on/off), are associated with a special, generic measurement unit.

D. Ontology component for operations

Each service operation description is bound to an input and output message type, as defined in the message ontology component. This is somewhat similar to the approach used in OWL-S.

The main difference resides in the categories involved in the binding. While in OWL-S actual operations are bound to messages, in our approach we bind operation concepts (semantic representations for classes of operations) to ontological message definitions.

```
<MessageType rdf:ID="messType1">
  <hasPart>
    <MessagePart rdf:ID="oinput1">
      <hasUnit rdf:resource="#kWh" />
    </MessagePart>
  </hasPart>
  <hasPart>
    <MessagePart rdf:ID="oinput2">
      <hasUnit rdf:resource="#kWh" />
    </MessagePart>
  </hasPart>
</MesssageType>
```

Fig. 6. Ontological description of a message

```
<Message rdf:ID="mess1">
  <hasType rdf:resource="#messType1" />
  <hasMapping>
    <Mapping rdf:ID="mapInput1">
      <hasRealPartName>
        input1
      </hasRealPartName>
      <hasPart rdf:resource="#oinput1" />
    </Mapping>
  </hasMapping>
  <hasMapping>
    <Mapping rdf:ID="mapInput2">
      <hasRealPartName>
        input2
      </hasRealPartName>
      <hasPart rdf:resource="#oinput2" />
    </Mapping>
  </hasMapping>
</Message>
```

Fig. 7. Mapping between an actual message and an ontological message description

As stated before, Web Service operations model actual device functionalities. Consequently, the operation descriptions must have associated functionalities (such as heating, cooling, temperature measuring etc.). This can be seen in Fig. 8. Since many device operations can produce multiple effects of different types (such as: starting an air conditioning device and setting some predefined temperature, or returning a temperature value and starting an internal revision mechanism), an operation can be associated with multiple functionalities.

E. Ontology component for devices

At the ontological level, devices are seen as a collection of operations. The sum of functionalities of the operations defines the capability description of a device. From this point of view, we can consider a device description as a semantic interface. Devices are classified hierarchically, based on implemented operations and their functionalities. The most general device

```
<Device rdf:ID="airConditioner1140C">
  <hasOperation>
    <Operation rdf:ID="turnOn">
      <hasFunctionality rdf:resource="#on" />
      <hasFunctionality rdf:resource="#cool"/>
      <hasInput rdf:resource="#void" />
      <hasOutput rdf:resource="#void" />
    </Operation>
  </hasOperation>
  <hasOperation>
    <Operation rdf:ID="turnOff">
      <hasFunctionality rdf:resource="#off" />
      <hasInput rdf:resource="#void" />
      <hasOutput rdf:resource="#void" />
    </Operation>
  </hasOperation>
</Device>
```

Fig. 8. RDF description of a device and two of its operations

has only *turnOn* and *turnOff* operations. Subclasses may add new operations, or might redefine existing operations by adding new functionalities. Fig. 8 also serves as an example of a device description.

V. The MetaBPEL composition scheme

A. MetaBPEL representation

Service execution languages such as BPEL do not allow for more abstract references to services. It is also important to note that WSDL descriptions (used for executing a BPEL specification) add the actual service binding to a service interface. In an Object Oriented Programming environment, WSDL is similar to the binding of an interface to the actual class instance that implements it. Thus, we cannot generally define behavior (or dynamic composition), as we would in OOP, because every interface is inherently bound to an actual object. The bindings between BPEL and WSDL documents are shown in Fig. 9.

In the context of home intelligence, whenever a new device (and its associated service) replaces an existing one, the composition scheme must be rewritten. Environments having similar or identical devices cannot share composition schemes.

Thus, we introduce an abstract composition language, MetaBPEL, that addresses both issues mentioned above. MetaBPEL uses standard WS-BPEL 2.0 syntax in order to define the control flow of a composition process. Actual references to services, operations and messages specific to WS-BPEL 2.0 are replaced with ontological descriptions.

More precisely, the following BPEL elements are replaced: partner links, invocations, message references from variable definitions, message part references from copy instructions.

Fig. 10 shows a MetaBPEL specification and its references to the ontology.

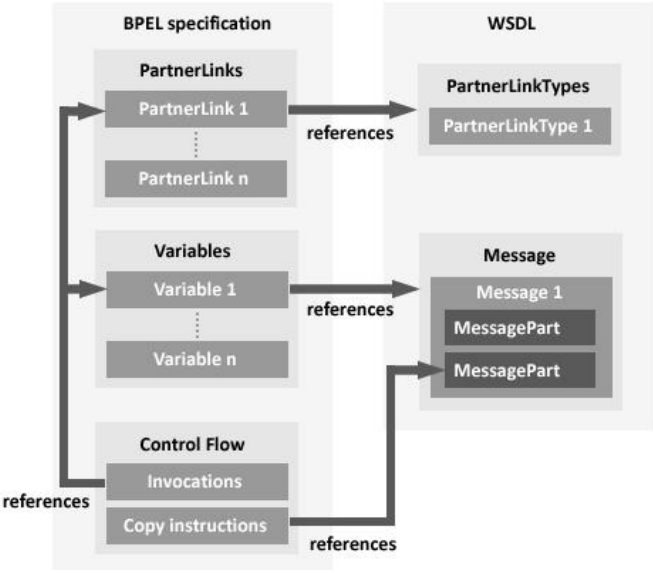

Fig. 9. BPEL specification and its references to WSDL components

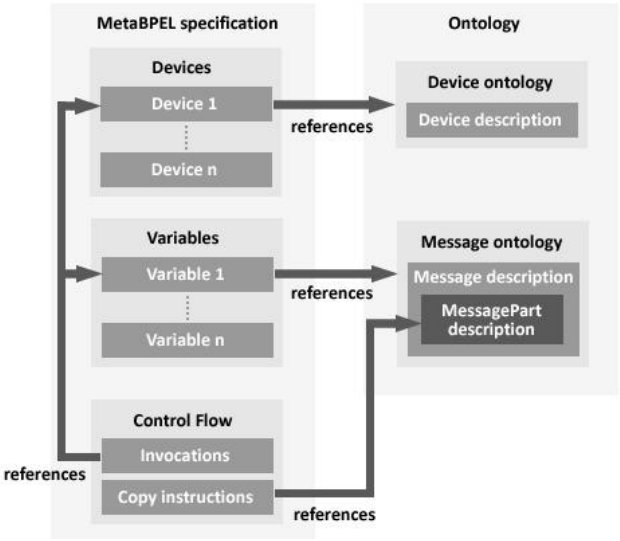

Fig. 10. MetaBPEL representation and its ontology references

B. MetaBPEL creation

MetaBPEL specifications can be created in the following ways:

- **manually**: user-defined, using visual tools. These tools are similar to BPEL composition tools, such as Eclipse BPEL Designer [22], but provide ontological descriptions for available services, operations, etc.;
- **automatically**: generated from BPEL specifications and WSDL descriptions enrolled in the composition process. Actual service references are examined and replaced with their ontology counterparts, as depicted in the previous section. A prerequisite for automatic MetaBPEL creation is the existence of ontological descriptions for each service taking part in the composition. Thus, the ontology performs as a database where all existing device types are registered.

C. MetaBPEL instantiation

MetaBPEL provides an abstraction over composition schemes/workflows defined using BPEL. As such, it allows for workflow migration between different environments. In order to run a migrated MetaBPEL in a new environment, an instantiation process creates a custom BPEL specification which is then deployed on a local BPEL service execution engine.

The instantiation process relies on:

- environment ontologies associated with particular house environments: they are a subset of the main ontology, and contain descriptions only for devices available in particular houses;
- a matching mechanism able to validate the compatibility between existing services and service descriptions from the MetaBPEL.

Assume we have a house environment with several devices that can be controlled using services. For each service, there

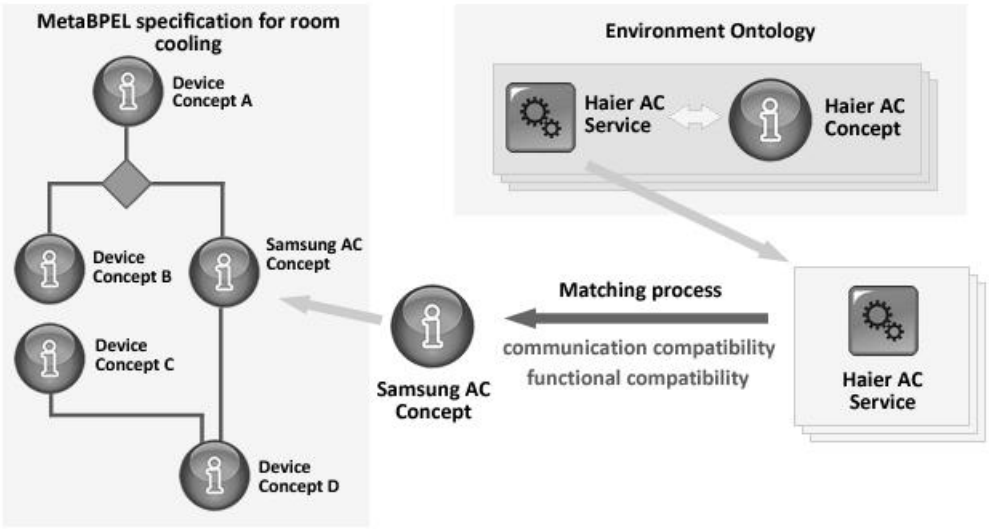

Fig. 11. Matching process

is an associated description in the Environment Ontology, as seen in Fig. 11.

Assume we have downloaded a MetaBPEL specification for intelligent room cooling that enrols five device concepts (*A*, *B*, *C*, *D* and *SamsungAC*)[1]. The following question arises: "Which implemented services can match device concept definitions from the MetaBPEL?". The matching process is responsible for selecting adequate services to replace concepts. In Fig. 11, the *HaierACService* is such a candidate. If no services are found for a device concept, then the MetaBPEL cannot be instantiated. On the other hand, if several compatible services exist, the user will manually select the one he prefers.

The matching process involves two steps:

- **functional** compatibility: establishes if the services operations produce the desired effects required in the composition process (for example: modifying room temperature);
- **communication** compatibility: establishes if a service is using messages compatible to other services it interacts with.

1) Functional compatibility: In order to establish if a service is functionally compatible with a certain device concept, the following must be done:

- the service ontological description, i.e. the device concept, must be retrieved;
- a functionality-based compatibility between used operations must be established.

For example, in Fig. 12, the operations *turnOn*, *turnOff* and *setTemperature* are used in the MetaBPEL for intelligent room cooling.

[1]References *A*, *B*, *C*, *D* and *SamsungAC* need not be all different. For example, *A* and *D* can refer to the same device that is used several times in the workflow. For the sake of simplicity, we omit such a situation here.

In order to check if the *HaierACService* can replace *SamsungACConcept* during the instantiation process, each operation from *HaierACConcept* must be found compatible with *SamsungACConcept*'s operations invoked in the MetaBPEL.

Operation compatibility is established using functionalities. Functionalities represent a way of identifying device operations that have the same physical effect. Two perfectly compatible physical devices may not necessarily be represented by two identical services. While two operations from two different services may stand for the same physical operation, they may also follow different naming conventions (for example *someoperation* and *SomeOperation*).

Functionalities can be seen as labels, and each operation is associated with a set of functionalities. For example, the operation *turnOn* implemented by an air conditioner can be seen as having two functionalities: *start device* and *modify temperature*. The compatibilization process for operations distinguishes among the following cases:

- **full** compatibility: occurs when two operations have the exact set of functionalities. In Fig. 13 there is an example of two operations having a single identical functionality;
- **partial** compatibility: occurs when an operation has additional functionalities. Fig. 13 shows such a case;
- **incompatibility**: occurs when an operation does not have the required functionalities.

The compatibility relation between concepts is not symmetric. An example can be viewed in Fig. 13: operation *On* can replace *turnOn* in a MetaBPEL, since it has all the functionalities of *turnOn*. On the other hand, *turnOn* cannot replace *On*, because it has fewer functionalities.

2) Communication compatibility: Communication compatibility refers to the ability of a service to understand messages from other services it interacts with. It is established using ontology message descriptions introduced in Section IV-B, and it is more strict than functional compatibility. A service imple-

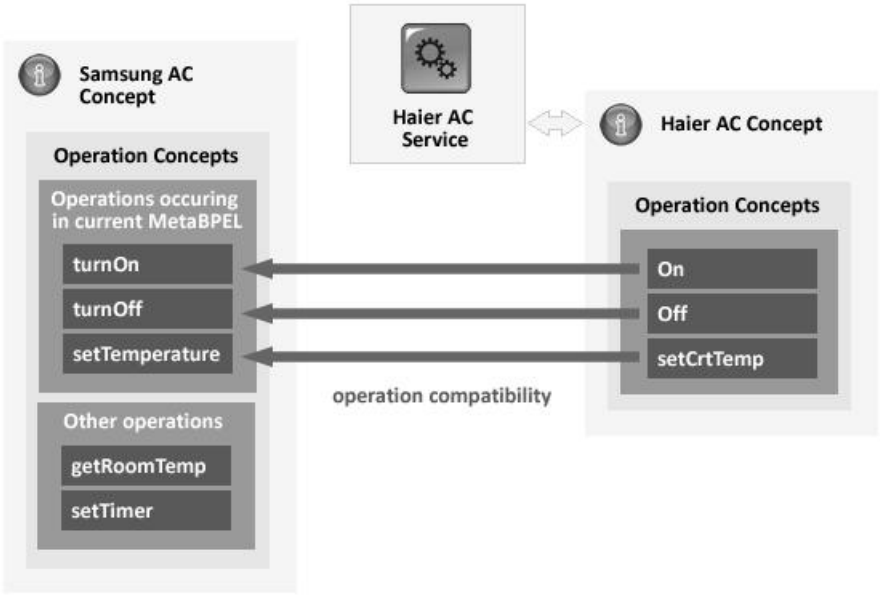

Fig. 12. Device compatibility

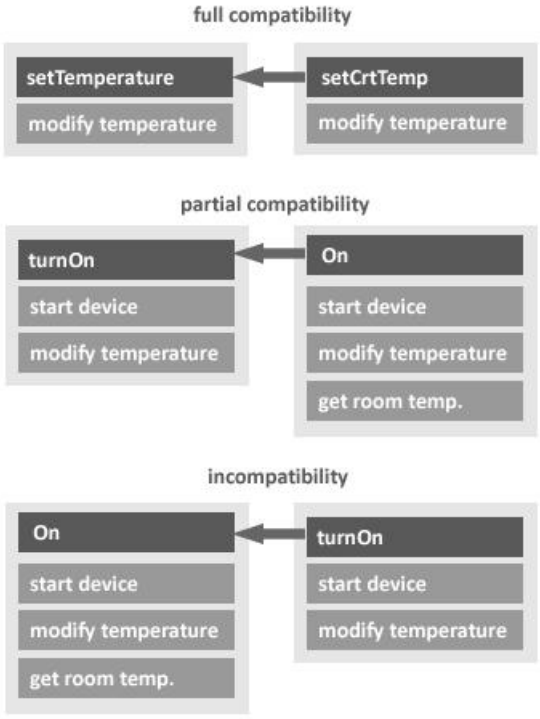

Fig. 13. Operation compatibility

mentation is compatible with a MetaBPEL abstract service if all input and output message types are bound to the ontological message descriptions referred by the abstract service.

When the MetaBPEL is instantiated, all references to ontological message parts will be replaced by the actual message parts of the matching service implementation. This process is done for each of the matching services that participate in the composition process.

D. MetaBPEL generalization

MetaBPEL documents contain abstractions of actual services (and their groundings). In order to increase portability, descriptions from MetaBPELs can be further generalized.

We illustrate this in Fig. 14: a user wants to create a composition scheme that includes his Haier air conditioning device. Either by means of a graphical interface, or from an existing BPEL specification, a MetaBPEL is produced, containing the Haier device description. Assume that, in the MetaBPEL, only operations *turnOn* and *setTemperature* of the Haier device are used. In this scenario, the MetaBPEL can only be used in environments that contain Haier devices.

But, since *setTemperature* is a very general operation, its functionality might be provided by other air conditioning devices as well. It is desirable to replace the *Haier* device concept with a more general one such as *AirConditioner*. The generalization process takes a MetaBPEL as input and produces another MetaBPEL where all concepts are generalized. The first step is to take all occurrences of a concept from the MetaBPEL, and identify the set of operations used in that MetaBPEL (as opposed to the entire set of operations). Each operation is associated with a set of functionalities, and these functionalities might be satisfied by a more abstract device from the ontology. For example, it is possible that some functionalities of a certain air conditioning device can be satisfied by the entire class of air conditioners, or at least by the class of air conditioners belonging to a specific manufacturer.

The generalization algorithm for a device concept works in the following manner: it starts from a device-concept and the set of functionalities used in an abstract workflow. In each

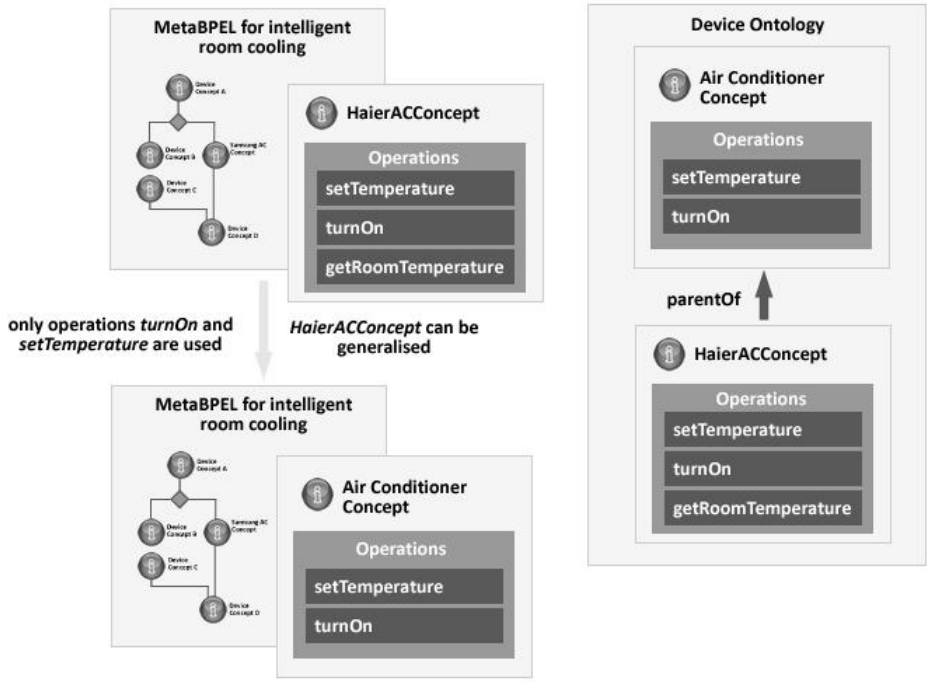

Fig. 14. Generalization example

iteration, it will attempt to move up the device hierarchy and check if the higher concept can still satisfy the set of functionalities. The process is repeated until the current concept is unable to satisfy the set of functionalities, or the top concept is reached. The result is the most general concept able to satisfy a given set of functionalities. A generalized MetaBPEL is created by replacing each device concept with its generalized counterpart, that is determined via the generalization algorithm, which is described using the pseudocode from Fig. 15.

E. Examples

By using MetaBPEL, generic workflows can be defined, that invoke generic, ontological descriptions of device operations. For example, the excerpt in Fig. 16 reads: *Find an air conditioner that has the model 1140C, and invoke its* setTemperature *operation, using the contents of the variable* temp *as input and storing the output within the variable* result. Here, *airConditioner1140C* is an individual from the ontology, that is an instance of the *HaierAirConditioner* class, while *setTemperature* is an instance of the *Operation* class.

WSDL messages have ontological counterparts (*Message* instances), that serve as a means to infer the compatibility between output messages that are fed as inputs to other services. For this reason, variables defined within the MetaBPEL specification also have ontological descriptions, such as the one shown in Fig. 17.

Here, *celsiusTemperature* and *successIndicator* are individuals from the ontology, that map concrete WSDL part names to certain ontological parts within the ontological message type. For example, *celsiusTemperature* is defined in Fig. 18.

```
CheckFunctionalities(
    Concept x,
    Operation [] ops)
  foreach Operation o in ops
    Functionality [] f =
        o.getFunctionalities()
    if ! x.canSatisfy(f)
      return false
  return true

GeneralizeDeviceConcept(
    DeviceConcept x,
    Operation [] ops)
  if ! x.hasParent
    return x
  if CheckFunctionalities(x.parent, ops)
    return GeneralizeDeviceConcept(
        x.parent,
        ops)
  return x

GeneraliseMetaBPEL(MetaBPEL m)
  foreach DeviceConcept c in
      m.getDeviceConcepts()
    Operation [] ops = {}
    foreach Occurrence x in c.getOccurrences()
      ops.add(x.getOperation())
    m.replace(
        c,
        GeneralizeDeviceConcept(c, ops))
  return m
```

Fig. 15. Pseudocode for generalization

```
<invoke
    device="@airConditioner1140C"
    operation="@setTemperature"
    inputVariable="temp"
    outputVariable="result" />
```

Fig. 16. Abstract invocation

```
<variable name="temp"
    messageType="@celsiusTemperature" />

<variable name="result"
    messageType="@successIndicator" />
```

Fig. 17. References to ontological message types

Thus, this message type has a single ontological part, called *temp*, that uses Celsius degrees as a measurement unit. The WSDL message of a concrete service shown in Fig. 19 must be mapped to the ontological type using a mapping such as the one described in Fig. 20, thus establishing a correspondence between concrete WSDL parts and ontological parts. Ontological message descriptions represent the common denominator when trying to check the communication compatibility of directly interacting services.

VI. Implementation

A prototype implementation of the proposed platform has been successfully deployed. It contains (1) services for controlling simple devices such as an air conditioner, (2) an experimental ontology for describing these services, as well as apropriate bindings, (3) a set of predefined MetaBPEL documents and (4) an experimental BPEL instantiator.

The instantiator uses Apache ODE BPEL parser in order to create MetaBPEL documents from BPEL. It was also used for the reverse process.

MetaBPEL documents were instantiated and the resulting BPEL documents were successfully deployed on Apache ODE. Experiments showed that MetaBPEL creation is not a computationally expensive process, due to its simple, BPEL-related structure.

```
<MessageType rdf:ID="celsiusTemperature">
  <hasPart>
    <MessagePart rdf:ID="temp">
      <hasUnit rdf:resource="#uDegC" />
    </MessagePart>
  </hasPart>
</MesssageType>
```

Fig. 18. Ontological message type definition

```
<wsdl:message name="setTemperatureRequest">
  <wsdl:part name="temp" type="xsd:float"/>
</wsdl:message>
```

Fig. 19. WSDL message

```
<!-- setTemperatureRequest is the name
    of the WSDL message -->
<Message rdf:ID="setTemperatureRequest">
  <hasType
     rdf:resource="#celsiusTemperature" />
  <hasMapping>
    <Mapping rdf:ID="mapTemp">
<!-- temp is the field name
    in the WSDL message -->
      <hasRealPartName>temp</hasRealPartName>
<!-- reference to the message part
    defined in the message type -->
      <hasPart rdf:resource="#temp" />
    </Mapping>
  </hasMapping>
</Message>
```

Fig. 20. Mapping between WSDL message and its associated ontological message type

The prototype implementation was created as part of research work for FCINT project (*Framework for service composition based on ontologies for the aggregation of knowledge and information for intelligent buildings*).

VII. Future work

Our main focus is to extend (and relax) the semantic service matching technique such that a broader class of services could be made compatible with a MetaBPEL. As part of such relaxation, a more flexible communication compatibility is desirable.

Also, we are exploring possible disadvantages resulted from partial functional compatibility: replacing an operation with another one having additional functionalities than required, might produce undesired side effects. In this case, partial compatibility should be considered a special case of incompatibility.

We are planning to create graphical tools for the creation of MetaBPEL documents. Such tools will allow users to create composition schemes in a visual environment, as well as exploring the compatibility of different available services within a workflow.

Acknowledgment

The research presented in this paper is supported by national project “TRANSYS Models and Techniques for Traffic Optimizing in Urban Environments”, Project “CNCSIS-PD” ID: 238.

References

[1] (2010, Sep.) Ashley Intelligent Houses. [Online]. Available: http://www.ashley-s.com/
[2] (2010, Sep.) EIB KNX Intelligent Building Control. [Online]. Available: http://www.intelligenthouse.net/
[3] *BACnet*, ASHRAE Std., 2004. [Online]. Available: http://www.bacnet.org/

[4] *Web Services Description Language (WSDL) Version 2.0*, W3C Recommendation, Jun. 2007. [Online]. Available: http://www.w3.org/TR/wsdl20/

[5] D. A. D'Mello and V. S. Ananthanarayana, "Dynamic web service composition based on operation flow semantics," in *Information Systems, Technology and Management*, ser. Communications in Computer and Information Science, S. K. Prasad, H. M. Vin, S. Sahni, M. P. Jaiswal, and B. Thipakorn, Eds. Springer Berlin Heidelberg, 2010, vol. 54, pp. 111–122, 10.1007/978-3-642-12035-0_12. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-12035-0_12

[6] *Web Services Business Process Execution Language Version 2.0*, OASIS Public Review Draft, Aug. 2006. [Online]. Available: http://docs.oasis-open.org/wsbpel/2.0/wsbpel-specification-draft.html

[7] (2010, Sep.) Apache ODE. [Online]. Available: http://ode.apache.org/

[8] K. Fujii and T. Suda, "Dynamic service composition using semantic information," in *Proceedings of the 2nd international conference on Service oriented computing*, ser. ICSOC '04. New York, NY, USA: ACM, 2004, pp. 39–48. [Online]. Available: http://doi.acm.org/10.1145/1035167.1035174

[9] J. Nitzsche, D. Wutke, and T. Van Lessen, "An ontology for executable business processes," in *Workshop on Semantic Business Process and Product Lifecycle Management (SBPM), volume 251 of CEUR Workshop Proceedings*, 2007.

[10] A. B. Amare and R. Siebes. (2010, Dec.) The BPool system: a scalable abstract WS-BPEL instantiation mechanism using semantic constraints. Faculty of Sciences, VU University Amsterdam. The Netherlands. [Online]. Available: www.cisa.inf.ed.ac.uk/OK/Deliverables/D7.5/bpel-for-lcc.pdf

[11] V. Issarny, D. Sacchetti, F. Tartanoglu, F. Sailhan, R. Chibout, N. Levy, and A. Talamona, "Developing ambient intelligence systems: A solution based on web services," *Automated Software Engineering*, vol. 12, pp. 101–137, 2005, 10.1023/B:AUSE.0000049210.42738.00. [Online]. Available: http://dx.doi.org/10.1023/B:AUSE.0000049210.42738.00

[12] S. B. Mokhtar, J. Liu, N. Georgantas, and V. Issarny, "QoS-aware dynamic service composition in ambient intelligence environments," in *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, ser. ASE '05. New York, NY, USA: ACM, 2005, pp. 317–320. [Online]. Available: http://doi.acm.org/10.1145/1101908.1101959

[13] V. Prinz, F. Fuchs, P. Ruppel, C. Gerdes, and A. Southall, "Adaptive and fault-tolerant service composition in peer-to-peer systems," in *Distributed Applications and Interoperable Systems*, ser. Lecture Notes in Computer Science, R. Meier and S. Terzis, Eds. Springer Berlin / Heidelberg, 2008, vol. 5053, pp. 30–43, 10.1007/978-3-540-68642-2_3. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-68642-2_3

[14] *Web Service Semantics — WSDL-S*, W3C Member Submission, Nov. 2005. [Online]. Available: http://www.w3.org/Submission/WSDL-S/

[15] *OWL-S: Semantic Markup for Web Services*, W3C Member Submission, Nov. 2004. [Online]. Available: http://www.w3.org/Submission/OWL-S/

[16] M. Herrmann, M. A. Aslam, and O. Dalferth, "Applying semantics (WSDL, WSDL-S, OWL) in service oriented architectures (SOA)," in *Proceedings of 10th Intl. Protégé Conference*, 2007.

[17] D. Bonino and F. Corno, "DogOnt — Ontology modeling for intelligent domotic environments," in *The Semantic Web - ISWC 2008*, ser. Lecture Notes in Computer Science, A. Sheth, S. Staab, M. Dean, M. Paolucci, D. Maynard, T. Finin, and K. Thirunarayan, Eds. Springer Berlin / Heidelberg, 2008, vol. 5318, pp. 790–803, 10.1007/978-3-540-88564-1_51. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-88564-1_51

[18] M. R. Huq, N. T. T. Tuyen, Y.-K. Lee, B.-S. Jeong, and S. Lee, "Modeling an ontology for managing contexts in smart meeting space." in *Proceedings of the International Conference on Semantic Web and Web Services*, ser. SWWS '07, 2007, pp. 96–102.

[19] *RDF Primer*, W3C Recommendation, Feb. 2004. [Online]. Available: http://www.w3.org/TR/2004/REC-rdf-primer-20040210/

[20] *RDF Vocabulary Description Language 1.0: RDF Schema*, W3C Recommendation, Feb. 2004. [Online]. Available: http://www.w3.org/TR/rdf-schema/

[21] *SPARQL Query Language for RDF*, W3C Recommendation, Jan. 2008. [Online]. Available: http://www.w3.org/TR/rdf-sparql-query/

[22] (2010, Sep.) Eclipse BPEL Designer. [Online]. Available: http://www.eclipse.org/bpel/