Supporting Composite Smart Home Services with Semantic Fault Management

Zohar Etzioni, John Keeney, Rob Brennan and David Lewis
FAME and Knowledge and Data Engineering Group
Trinity College Dublin
Ireland
e-mail: {etzioniz|John.Keeney|Rob.Brennan|Dave.Lewis}@cs.tcd.ie

***Abstract*— The proliferation of smart home technologies providing home users with digital services presents an interoperability problem. This paper asserts that the value of these services can be greatly extended by enabling technology-neutral compositions of these home services, and furthermore that the reliability of such composite services will be of paramount importance to ensuring widespread deployment. Therefore fault management capabilities must accompany the composition capabilities in order to increase the robustness and reliability of such services. This paper proposes a web services-based abstraction layer for home area network service composition and a semantically informed fault management system for composed services, which can assist in diagnosis and correction of problems with composite services in smart homes of the future. Prototyping work and use cases are also described and initial metrics are presented that investigate the overhead of introducing the technology neutral abstraction layer.**

Keywords - component fault management, composite services, UPnP, home area network.

I. MOTIVATION

Applications in the smart home of the near future will be constructed as service compositions or mash-ups that combine capabilities offered by a wide range of device vendors and service providers. This is because no individual vendor or provider can ensure control of a personalized consumer-created environment based on hardware and software components, even where this is enabled by interoperability standards such as UPnP[6] and HTTP/SOAP web services. Thus smart home applications will increasingly cut across traditionally segmented domains such as white goods manufacturers, consumer electronics, domotics, telecommunications providers and internet-based services. Thus diversity is guaranteed.

Given this wide array of components and the emerging ubiquity of service-oriented computing solutions, device and service compositions will play a key role in leveraging the value of individual service components in an ever expanding array of new and useful applications. This leverage of value includes activities such as support for chaining of components into composite services in order to provide richer user experience by introducing new applications from existing device services. It can also be a force multiplier in terms of effectiveness, for example a standalone a database service is of limited utility to most consumers but when coupled with social networking, accounting and appointment book applications it provides a vital part of many workflows that could be used to support home businesses or even weekly shopping trips. Thus the inherent value in the underlying services, normally only available to specialist technical users, is unlocked for end-users.

For device vendors and service providers the value offered by their products will shift from being solely concerned with their use in isolation, to their ability to be used in concert with the rest of the digital home ecosystem of devices and services. Until now, such compositions have only been available to users through service compositions continuously being offered by innovative third parties. The developers of these components can therefore increase their attractiveness to consumers by maximizing their ability to be integrated with other components. However, this must be balanced against the integration costs incurred by third party service composers and the management costs incurred by composite service operators, which may deter them from actually including the component in future composite offerings. Support for recursive composition of atomic services means that service providers can expand their service offerings by re-using their own atomic resources and services as parts of composite or specialized services that they then offer directly to consumers. Finally the ability to compose local services with other service providers facilitates the building and maintenance of strategic partnerships with other service providers in order to maximize the value (and hence demand) for the atomic services.

Therefore if such compositions are going to be practical then it is insufficient to enable just the initial stage of integration, instead the composed service must be supported throughout its lifecycle i.e. from composition, through deployment, maintenance or modification and finally retirement. This is especially important for services deployed in a smart home as it is necessary to enable troubleshooting and corrective actions by non-technical users, who may themselves be the actor who initially composed the services, or to enable self-healing or corrective actions by intelligent services deployed in the home.

There have been several promising technologies proposed for smart home device and service integration, these include UPnP[6], Jini [7], HAVi[8], OSGi[2] and DPWS[11]. All of these technologies enable, to a greater

or lesser extent, interconnectivity between heterogeneous devices and services. While they generally provide useful abstractions for interaction with a particular device, these technologies are non-interoperable. This leaves vendors and consumers locked into spot solutions and ultimately not delivering on the promise of universal smart home device and service integration. Creating services with a richer user experience for home area networks requires underlying services that are composable in a way that enable flexible compositions of the services exposed by networked devices in the smart home.

This paper presents a modular approach for enabling the composition of smart home services. An implementation of this approach is presented whereby UPnP services are automatically wrapped as web services to take part in a declaratively defined service compositions based on the BPEL[10] standard for specifying executable composite processes. However, to support runtime fault management of the composite service we enhance this infrastructure using a semantics-based approach. A knowledge model of UPnP events and faults expressed as an OWL[9] ontology was created. This facilitates machine-based reasoning about faults from atomic UPnP services in order to derive composite faults for the composed services. The semantics of a fault in a composite service are not simply the aggregation of the semantics of atomic faults but in fact need to take into account the interplay of atomic services in a composite service. For example the failure of a specific atomic service does not necessarily imply the failure of the whole composite service. Therefore, this approach significantly increases the manageability and hence reliability and suitability for self-healing and diagnosis of composite services for smart home applications.

In section II we provide an overview of related work on smart home services composition and management. Section III presents the architecture approach for enabling composition of smart home services and our current implementation in terms of BPEL-based integration of UPnP-based smart home services wrapped in web service interfaces. Next, in section IV we describe a use case for this work based on an example composite smart home service. Our ongoing work on enhanced fault-management for composition of semantically annotated events is described in section V. Finally we present some conclusions and a summary of the status and applicability of our work.

II. RELATED WORK

In general while there has been a lot of work on enabling composition of smart home services, most of this has focused on the use of a single integration technology such as UPnP, here we discuss relevant related research that has specifically addressed the likely technological heterogeneity in the smart home.

A middleware approach for enabling service composition for smart homes with BPEL is presented in [1]. The approach is based on marrying an OSGi platform [2] with the BPEL specification and a runtime engine. In this architecture a composite service is expressed as a BPEL process that orchestrates a set of OSGi services. The composite service is deployed in the OSGi platform as a virtual bundle and then can be accessed by other services as an ordinary OSGi service. This is equivalent to the way BPEL services have an ordinary WSDL based interface and can be accessed as web services. When a composite service is located and invoked, the implementation bundle calls the BPEL engine, which interprets the process specification and executes the process and can call back OSGi services. Smart home services are discovered and registered with the OSGi platform as services via technology related bundles (e.g. a UPnP bundle) and thereby can be invoked from composite services running on this platform. In contrast to our approach the OSGi platform rather than web services are used to abstract the individual technologies and there is no formal knowledge model used to capture cross-technology service event or fault semantics.

Wen et al. [5] tackle the problem of interoperability and heterogeneity in smart homes. They discuss the different layers of interoperability and for each layer suggest their preferred solution. For connectivity interoperability they suggest an Ethernet cloud; for network interoperability – TCP; and for syntactic interoperability - XML/SOAP. The main difference from our approach is that we describe an open driver-based middleware that can be extended with “drivers” per technology as an adaptation layer from a protocol specific to protocol neutral web services interface.

In [3] the problem of dynamic smart home services composition is discussed. They suggest an ontology-based approach for smart home service composition, specifically in the audio/video domain. A system architecture that allows dynamic service composition based on semantic technology is presented. They define a device and a capability ontology, which are used to assist with service composition to enable determining which device can perform which semantic task.

uMiddle [13] is a universal middleware for interoperability in pervasive environments. uMiddle uses a universal description language as a canonical service description thereby enabling developers to write applications that interact with devices with a technology-neutral interface. uMiddle does not address declarative service composition and does not support fault management features.

FedNet [14] is an intermediary based solution for transparent integration of applications with diverse smart objects by wrapping the interaction with the smart object and placing the FedNet intermediary between the applications and the smart devices. Smart devices are described via proprietary description documents. The interaction between the application and the smart device representation is made through a RESTful interface. The application description is declarative and proprietary and does not support specific fault management.

Finally, in previous work on policy-based integration of multi-provider digital home services [12], the authors

have addressed the issue of the federated distribution of management authority in a domain where device and service providers may wish to retain some control of their offerings, despite the composed service residing or executing largely within the smart home.

III. ARCHITECTURE

A. Background

In order to facilitate composition of services in the smart home, we propose that an abstraction layer should be introduced between the low-level device/service technology, and the service composition layer. This abstraction layer should hide the differences in format, protocol, and network related details from the consumer and expose the device services in a common standard way. We adopt a web-service based approach, such that devices in the home area network are discovered and their services are exposed to the user as web based services. In addition to enabling the broadest method of standard access to all types of services, this approach enables the composability of such services using existing service composition standards, which is an important attribute desired for smart home applications.

Business Process Execution Language (WS-BPEL or BPEL in short) [10] is an OASIS[1] standard language for the orchestration of web services described via the web service description language (WSDL) to be called and controlled via control flow structural activities. An important attribute of BPEL is that the composite service is also rendered as a web service thereby making it interoperable and easily integrated with other web services.

B. Proposal

In this architecture we propose a framework to allow smart home service compositions to be defined using BPEL. In addition we suggest the dynamic and automatic creation of a web service proxy for each digital service, e.g. the capability offered by a UPnP-enabled device discovered in the HAN. This approach provides a common web service-based interoperable layer which can then be used in BPEL-based composition along with actual web services. BPEL enables declarative service composition and allows for seamless integration with web based services. Both BPEL and web services provide structured and declarative fault handling as part of the language which is considered essential for management of composite home services. In our future work we will consider more lightweight approaches both for service composition as well as for proxy service representation, e.g. RESTful services.

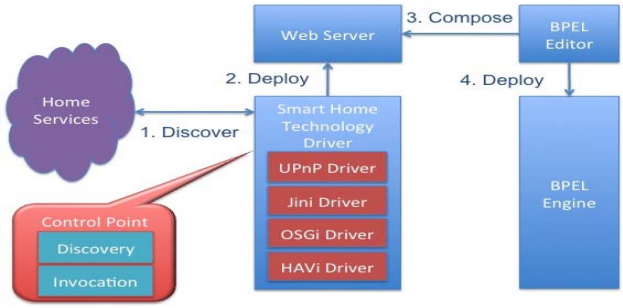

Figure 1 Abstract Architecture

In this abstract architecture (Figure 1) a "driver" module is loaded for each smart home service technology. Such a driver is required to abstract the interaction with the devices supporting the particular technology. It is expected to discover the technology-compliant devices, extract their services description, generate corresponding technology neutral web services and deploy them to the local web server. Finally it should be able to invoke operations on the device on-demand. By creating a proxy interoperable web interface, different smart home technologies could be used and composed in a uniform manner ignoring the underlying differences in protocol, data structure, platform and network. This architecture could ideally be deployed as part of a home gateway.

Figure 1 shows the architecture with a driver for UPnP. This driver is implemented as a UPnP control point application that interacts with the UPnP devices in the HAN. The control point application has two main sub-modules:

(i) Discovery sub-module – responsible for discovering UPnP devices and services in the home are network using the UPnP discovery protocol. Once a device is discovered, its services are investigated. For each service, its XML description is fetched, and based on this description, a corresponding web service proxy is generated on the fly using template-based code generation. For each service, a single web service is created, such that for each action there is a corresponding web service operation. The parameters of the operation match the parameters of the UPnP action. The web service returns a string and may throw an exception in case an error was returned from the UPnP service. Figure 2 shows a snippet from the AVTransport service of a UPnP media renderer, while

```
<action>
   <name>Play</name>
   <argumentList>
      <argument>
         <name>InstanceID</name>
         <direction>in</direction>
         <relatedStateVariable>
               A_ARG_TYPE_InstanceID
         </relatedStateVariable>
      </argument>
      <argument>
         <name>Speed</name>
         <direction>in</direction>
         <relatedStateVariable>
              TransportPlaySpeed
         </relatedStateVariable>
      </argument>
   </argumentList>
</action>
```

Figure 2 UPnP AVTransport Service Description

[1] Organization for the Advancement of Structured Information Standards (OASIS) - http://www.oasis-open.org

```
<xs:element name="Play" type="tns:Play">
</xs:element>
<xs:element name="PlayResponse" type="tns:PlayResponse">
</xs:element>
<xs:complexType name="Play">
    <xs:sequence>
        <xs:element name="InstanceID" type="xs:string" minOccurs="0">
        </xs:element>
        <xs:element name="Speed" type="xs:string" minOccurs="0">
        </xs:element>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="PlayResponse">
    <xs:sequence>
        <xs:element name="return" type="xs:string" minOccurs="0">
        </xs:element>
    </xs:sequence>
</xs:complexType>
<message name="Play">
    <part name="parameters" element="tns:Play">
    </part>
</message>
<message name="PlayResponse">
    <part name="parameters" element="tns:PlayResponse">
    </part>
</message>
<portType name="MediaRenderer_AVTransport">
    <operation name="Play">
        <input message="tns:Play">
        </input>
        <output message="tns:PlayResponse">
        </output>
        <fault message="tns:UPnPException" name="UPnPException">
        </fault>
    </operation>
```

Figure 3 Media Renderer AVTransport Service WSDL

figure 3 shows an extract from the automatically generated WSDL file generated for the proxy corresponding that UPnP service.

Once a proxy web service has been generated it is built and packaged as a web application and deployed to a local web server. The discovery application is dynamic and as soon as a UPnP device or service is added to the network it is automatically detected, and its corresponding web service proxies are generated and deployed to the web server. As soon as the device or service disappears from the network, its corresponding web service proxies will be immediately undeployed from the web server and removed. The implementation of each web service operation delegates the call to the corresponding UPnP device or service via the control point execution module described below.

(ii) Invocation sub-module – responsible for invoking home service actions and extracting the results, errors and events. When an action is invoked through the execution service, the corresponding device action is called and the result or error code is returned to the caller. In case an error occurs, the UPnP error code generates a UPnP exception which is thrown back from the web service to the caller. An evaluation of the performance overhead for the web service proxy approach for UPnP services is given in section V.

Once a proxy has been deployed to the web server, a composite service can then be defined in BPEL involving the proxies, and indeed may include other arbitrary web services. Such a composite service may be deployed in a BPEL runtime engine and be executed in the home environment, either as part of a client application, or indeed as part of another composite service.

The advantage of a web service proxy based approach is that it enables standard seamless composition of web-based services with home services with no additional effort. Another advantage is that a similar approach can also be applied on types of home services other than UPnP.

IV. USE CASE

In this section we describe a sample service composition use-case where generic UPnP services, wrapped as web services are composed into a composite service for use within the smart home. In this case we envision a scenario where a smart home user may wish to stream video from a UPnP media server to a UPnP media renderer (display device), and when the user changes rooms, to have the media stream displayed on another UPnP media renderer in the other room if appropriate. Figure 3 shows a snippet from the WSDL conversion for a UPnP media renderer AVTransport service, shown previously in figure 2. A similar automatic conversion can be created for other media devices and services available in the smart home.

The composition of the service can then be easily performed, making extensive use of easily procured flexible web service composition tools. Figure 4 shows a screen shot of a composition defined graphically using the Netbeans IDE[2] service composition tool. Once assembled,

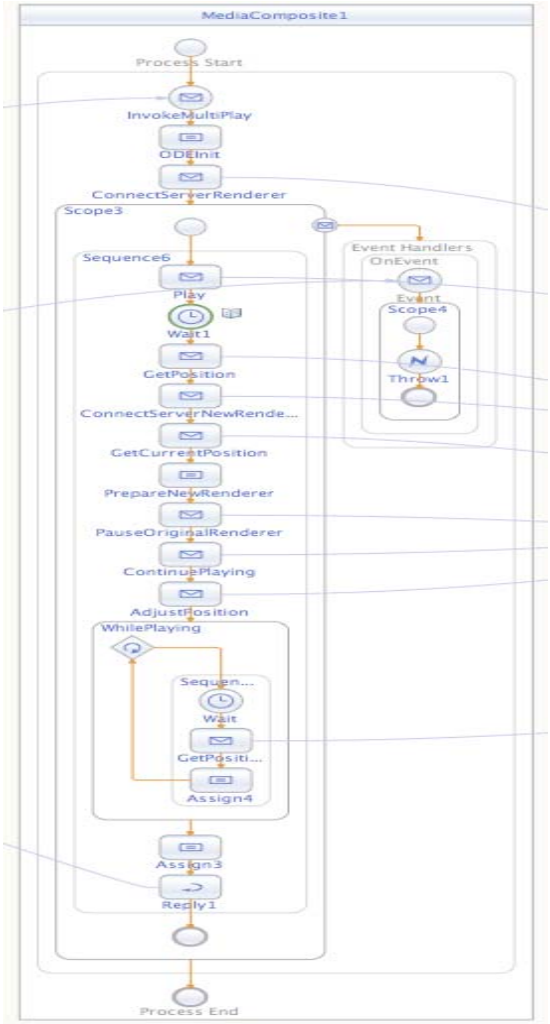

Figure 4 Multi Renderer Composite Service

[2] http://www.netbeans.org

the WSDL of the two constituent UPnP services, and a service to provide contextual location information about the user, are composed together to automatically produce a standards-compliant BPEL process. This is then deployed to a BPEL engine, and made available as a WSDL compliant web service to be directly used by a client application, or reused in an even more sophisticated composite service.

This composition has been implemented. In this case a UPnP compliant XBMC [3] media server was used with two XBMC UPnP media renderers, while the location tracking information was provided via a simulation service. The composite service executes as expected, with the video stream smoothly transitioning from one media renderer instance to another.

However, a more challenging issue arises if an error occurs at one of the constituent services in the composition, for example, where a malformed media location URL is passed into a media renderer. UPnP devices and services can be considered state machine based, where state changes are signaled via status updates, which can be modeled as events. Where an operation causes an error, rather than have the faulting operation fail instead it succeeds, with the state transitioned to an error state and signaled with a state change event. When this model is mapped to the synchronous request/response web services approach, all operations appear to succeed. Out of band error checking becomes more complicated, especially where some events signal normal operation, some signal warnings, some signal resource-level issues that may be independent of any operation being performed, and finally only some events signal errors.

If the process of wrapping UPnP services as web services to support their agile integration with other web services is to be semi automated, and the composition process made easier, this raises a number of further concerns. Firstly, there is a requirement to capture the semantics or meaning of different events, so they can be mapped to the individual operations and underlying resources of individual UPnP devices and services. While this information can be determined by reading the UPnP specifications, they are not formally encoded. Secondly, given the focus on event-based feedback, the composition process needs to be extended to support not just the synchronous data-flow approach, but also support the composition-level combination of asynchronous events in a holistic manner. These issues and a proposed approach to address them will be further discussed in the next section.

V. ONGOING WORK

In this section we provide more details on our ongoing work on semantics-based composite fault management and current prototyping experiences.

A. Composite Service Fault Management

Fault management is a sub area of service management that aims to enable the user or service provider to identify a problem with a service as soon as it occurs. Traditionally in telecommunications networks, implementing fault management functions is a very high priority for any new deployments since service outages directly affect revenue. Improving fault management is also an essential step towards enabling automation in composite service management. Composite service fault management in general deals with detection, isolation, correlation and correction of events that affect the composite service. In composite services, failures can occur in a number of ways: within the services themselves – application related faults, during the interaction with services, e.g. communication problems or versioning problems; or composition related faults – such as incompatible parameters. Identifying faults enables a user or manager to respond quickly and appropriately to failures, thereby increasing the reliability of the composite service. Holistic composite service management should consider event information coming from atomic services in addition to other available context information such as the model of the composition and its execution state. This leads us to the next topic.

B. Semantic Modeling For Composite Events

Comprehensive end-to-end aware composite service fault management requires composing management information coming from multiple sources to report accurately about the state of the composite service and how it is affected by events or faults in constituent services. A gap exists between the level of a fault management interface that is exposed by atomic services and the fault management interface we desire to have for the composite service. For example, atomic service events may be batched, suppressed, or mapped to composite service faults depending on the execution state and the model of the composite service.

The event interface of atomic services can be enriched with formal semantic models, with such a semantic fault interface can support composite fault management. In this paper the authors propose a framework that can be used by the composite service designer. By using declarative rules, enriched with formal semantic models, the faults of constituent atomic services can be mapped to those of a composite service, aligning the fault management of the composite service with the end-to-end management goals of the composition.

C. A Framework For Composite Service Fault Management

Figure 5 describes a conceptual framework for a composite service fault/event detector. The purpose of this architecture is to describe rule-based composite service fault management, deploy this management interface, and identify composite faults and events during the composite service runtime. The framework has three main subsystems:

[3] Xbox media centre - http://xbmc.org/

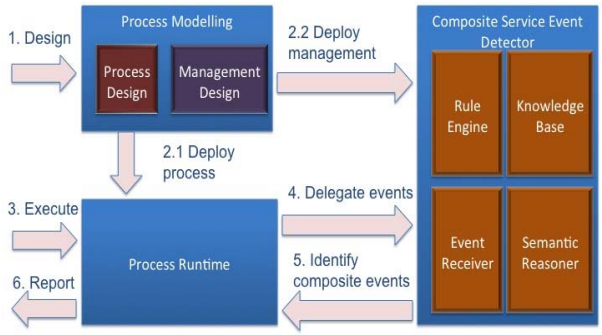

Figure 5 Composite Service Fault Management Abstract Architecture

(i) *Fault composer* – This tool allows the fault management designer for the composite service to leverage the knowledge present in the composite service model. It is proposed to implement the fault composer tool as a plug-in extension to an existing BPEL design tool. Fault management interface composition will be done using an IDE GUI (e.g. an Eclipse or Netbeans plug-in) that would enable the user to compose fault information from the various available sources and in addition it will be able to automatically suggest to the user composite event patterns based on an analysis of the composite service model and constituent atomic services fault management interfaces. In order to enable such an automatic suggestion mechanism, the composite service model is inspected in conjunction with the atomic services fault management interfaces. For example, if there is no alternative path, all atomic service faults are promoted by default to be composite service faults. Another option is to merge semantically equivalent faults to their common parent concept.

(ii) Composite service runtime – there are three interaction points between the composite service runtime and the fault management system:

- When a composite service is deployed to the composite service runtime engine, its corresponding management interface (i.e. the composite events/fault rules) is deployed to the composite service event detector. However this is an incremental process and management can be deployed at a later time, or upgraded after the composite service has been deployed.
- The composite service event detector needs to receive from the composite service runtime engine information about runtime events/faults coming from atomic services, process models, process execution state changes, input/outputs, composition related faults
- When an event/fault is identified, the composite service event detector must be able to send the event/fault back to the composite service runtime engine so it can be relayed as a fault/event coming from the composite process that raised it.

(iii) *Composite service event detector* – this is the main runtime component of the system and its role is to identify patterns matching composite services faults/events. It is based on the following main concepts:

- Semantic modeling – enrich the atomic services fault management interfaces with semantic annotations from a fault management ontology. This requires a high-level, domain independent fault ontology, a composition fault ontology, and a domain-specific fault ontology, e.g. a UPnP fault ontology (see Figure 8 for an example based on our current work).
- Semantic reasoning – given the ontologies described above, every instance of event/fault received from the process runtime is inserted to the knowledge base and can potentially trigger semantic reasoning rules associated with the fault management interface of the related composite service. Such reasoning can involve type based reasoning – e.g. checking if the fault type is a subclass of another ontology type. Instance based reasoning can be used to check the relation between two instances of faults and infer the existence of a third fault with a different or similar type.
- Rule engine – expresses the fault management interface of the composite service as a set of rules mapping between atomic services semantic event information, composition related events (execution state and model), temporal constraints, and additional available context. Additional context can be anything that is known to the composite service fault management designer to be available in the deployment environment including network events/faults, application events/faults, and device events/faults.

D. *Prototype Implementation*

In order to demonstrate and evaluate the above approach, a prototype of the system has been implemented, tested, and has been shown to perform as expected for a small number of UPnP devices and services. The current prototype was implemented support for only one technology, UPnP, but as the architecture is open and modular, we would extend this as part of our future work. The architecture of this prototype, targeting composite service fault management in a smart home HAN, is given in Figure 6. During runtime, the UPnP control point application discovers UPnP devices, generates corresponding web service proxies, allowing

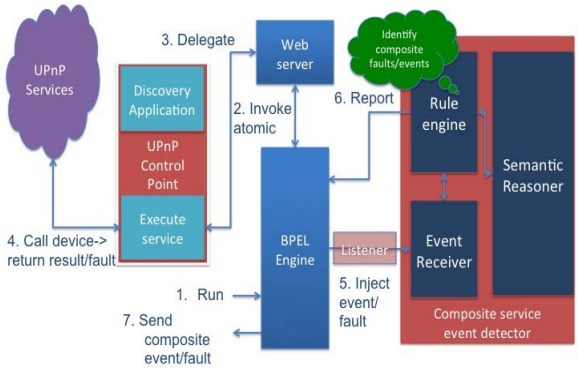

Figure 6 Prototype Architecture

services to be composed using a BPEL editor, and have these composite applications deployed and executed. The prototype uses Apache Tomcat[4] as web server to host the proxy web services and Apache ODE[5] BPEL engine as a web application deployed in it for hosting and executing composite services.

In parallel the composite service fault management interface can be designed as a set of rules mapping atomic service events and faults, composite service execution state changes and temporal constraints into the fault management of the composite service. The prototype uses JBoss Rules 5 (formerly known as Drools) as a rule engine and rules are defined using the Drools DRL format (See figure 7). These management rules are specified in a *when-then* format, such that the *when* part defines a composite fault pattern, and the *then* part defines the type of composite fault that correspond to the pattern. The fault management patterns can refer to event semantics (as triples), composite service life cycle events, state changes and temporal constraints. A rule file may have multiple patterns of composite faults that correspond to a certain composite service. Once the rule file has been defined it is placed in the predefined rules directory and picked up by the rule engine. During runtime events and faults coming from atomic services, BPEL runtime, and UPnP devices are inserted to the rule engine as event/fault objects, as well as triples. Triple object are extracted from the ontology representation of the event/fault instance. Figure 7 shows an example of such a composite fault pattern corresponding to the composite media renderer described in section IV. This rule identifies a fault (f1) whose type is one of the semantic subclasses of UPnPArgumentFault according to the ontology that was defined for this prototype. In case a match is found, a composite fault having a type "UPnPArgumentFault" is sent back to the composite service and from there to the caller.

```
rule "Semantic-Illegal-Argument"
when
        e1: ProcessInstanceLifeCycleEvent(
            processName=="MediaComposite1"
        )
        f1: Fault(
            id==e1.id
        )
        t1: Triple(
            subject==f1.id,
            predicate=="hasType"
        )
        t2: Triple(
            subject==t1.object,
            predicate=="subClassOf",
            object=="UPnPArgumentFault"
        )
then
        sendCompositeFault(e1.getProcessId(),
        e1.getProcessName(),
        "UPnPArgumentFault");
```

Figure 7 Semantic Composite Fault Example

For this prototype a high level fault ontology was created along with a UPnP Fault management ontology. The High level fault ontology defines the main generic types of events and faults, such as physical fault, communication fault, application fault, security fault, etc. It also defines a set of composition related events, such as process life cycle event, and activity state change event. For UPnP it defines a UPnP fault which is mapped from the error code that can be returned by a service action, and a UPnP event that corresponds to a device state variable change of state. For the prototype purposes only a small subset of the errors were modeled, specifically the focus has been on the general UPnP errors and on specific errors in media renderers. Events and faults in the UPnP fault ontology have an associated resource, which represents the device, and a fault type, which is the root of a hierarchy of fault/event types. Figure 8 shows a snippet from this ontology.

During runtime, events and faults coming from the atomic constituent services are delegated through the web service proxy to the event receiver of the composite event detector. Additional events representing process life cycle and execution state changes are received though a listener implemented in the BPEL engine. The control point application listens to UPnP events and forwards them to the composite event detector.

Once events and faults are inserted to the rule engine, the relevant composite service's rules are triggered as appropriate whereby the rule engine attempts to identify composite fault/event patterns, informed by the semantics of the events, combined with semantic information modeling generic composite services. In case such a pattern is identified, an appropriate predefined composite fault template is instantiated and sent back to the composite service and from there sent back to the caller of the service.

For testing the prototype a composite service as illustrated in section IV was used. Multiple XBMC clients were used as media renders and media servers on multiple desktop and laptops. The prototype was tested with a small set of

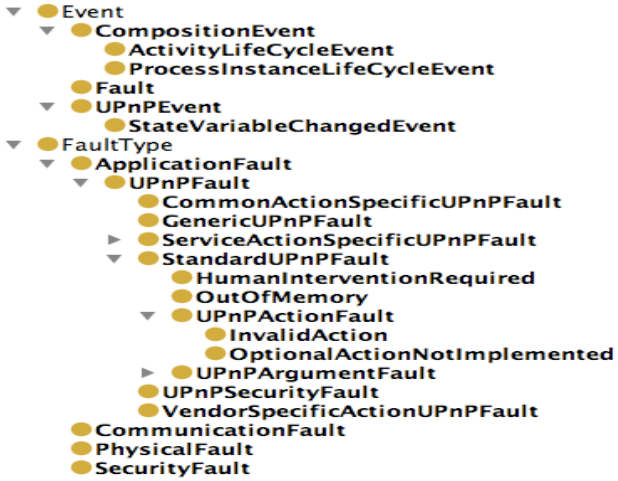

Figure 8 UPnP Fault Ontology

[4] http://tomcat.apache.org/
[5] http://ode.apache.org/

rules corresponding to this service.

Table 1 shows indicative performance overheads introduced by the prototype implementation for 3 media renderer AVTransport service operations ("Play", "Pause" and "Stop"). The first column (UPnP) shows the time taken to directly invoke the operation using SOAP messaging. The second column shows the server-side time required to perform the same operation when it is wrapped in a web service interacting with the UPnP control point application, where the additional redirection overhead introduced by the server-side processing is approx. 5 milliseconds. Column three (Roundtrip) shows the time taken to invoke the operation from a Java web service client on the same host as the web service and demonstrates the additional overhead introduced by the client side processing to form the web service request, pass it to the web server, and then wait for the web service response. These metrics demonstrate the minimal overhead introduced by this approach, especially when offset against the added capabilities to easily compose these services, not just with other UPnP services, but potentially with any web service.

Table 1 Performance Overhead

	UPnP	*Webservice wrapped UPnP*	*Roundtrip (Client, Webservice & UPnP)*
Play	112.36ms	117.86ms	387.52ms
Pause	365.46ms	370.98ms	614.85ms
Stop	342.35ms	347.44ms	585.0ms

VI. CONCLUDING REMARKS

In this paper the authors envision that smart home functionality will be constructed as service compositions or mash-ups based on device and service offerings from a wide range of manufacturers and providers. In this paper an approach is presented where UPnP devices and services are wrapped as web services which then can be easily reused and composed with a wider set of devices and services. One of the major side-effects of this approach is a requirement to perform more sophisticated composite fault management in compositions containing UPnP devices and services. A design and prototype implementation of such a composite fault management framework, informed by formal semantic models of the services, faults and the application domain is also presented. Early experimental results indicate a low performance overhead for the approach. In future implementations we are going to look at RESTFul web services as a lightweight common interoperable infrastructure.

ACKNOWLEDGMENT

This material is based upon works supported by the Science Foundation Ireland under Grant No.08/SRC/I1403 as part of the Federated, Autonomic End to End Communications Services Strategic Research Cluster (www.fame.ie).

REFERENCES

[1] Redondo, R.P.D., Vilas, A.F., Cabrer, M.R., Arias, J.J.P., Duque, J.G., Gil-Solla, A.: "Enhancing Residential Gateways: A Semantic OSGi Platform". IEEE Intelligent Systems (2008) pp. 32-40, January/February, 2008

[2] OSGi Service Platform, Core Specification r4, The OSGi Alliance, 2005. Available: http://www.osgi.org

[3] Bottaro, A., Gérodolle, A., Lalanda, P., "Pervasive Service Composition in the Home Network", 21st International IEEE Conference on Advanced Information Networking and Applications (AINA-07), Niagara Falls, Canada, May 2007.

[4] Perumal, T., Ramli, A.r., Leong, C.y., Mansor, S., Samsudin, K., "Interoperability for Smart Home Environment Using Web Services", International Journal of Smart Home, vol. 2, no. 4, Oct. 2008, pp. 1-16.

[5] Wen, C.C., Wei, T.S., Peng W.S., Mohamad, N., "Supporting service composition with ontology-based UPnP AV architecture in AV environment," In Proceedings IEEE Conference on Innovative Technologies in Intelligent Systems and Industrial Applications, (CITISIA 2008) pp. 104-109, Cyberjaya, Malaysia, July 12-13, 2008.

[6] Universal Plug and Play Forum, www.upnp.org

[7] JINI, www.jini.org

[8] HAVi, "HAVi, the A/V digital network revolution", http://www.havi.org/pdf/white.pdf, 1999.

[9] Web Ontology Language (OWL): "Submission Request to W3C: OWL 1.1 Web Ontology Language". W3C. 2006-12-19. http://www.w3.org/Submission/2006/10/.

[10] OASIS. Business Process Execution Language v2.0 http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html, 2007.

[11] Devices Profile for Web Services (DPWS): http://docs.oasis-open.org/ws-dd/ns/dpws/2009/01

[12] Brennan, R., Lewis, D., Keeney, J., Etzioni, Z., Feeney, K., O'Sullivan, D., Lozano, J., Jennings, B., "Policy-based integration of multi-provider digital home services", in IEEE Network Magazine, Vol 23, No. 6, Nov/Dec 2009.

[13] Nakazawa, J., Tokuda, H., Edwards, W. K., and Ramachandran, U. 2006. "A Bridging Framework for Universal Interoperability in Pervasive Systems". In Proceedings of the 26th IEEE international Conference on Distributed Computing Systems (ICDCS) July 04 - 07, 2006.

[14] Kawsar, F., Nakajima, T., and Fujinami, K. "A document centric approach for supporting incremental deployment of pervasive applications". In Proceedings of the 5th Annual international Conference on Mobile and Ubiquitous Systems: Computing, Networking, and Services, (MOBIQUITOUS2008) Dublin, Ireland, July 21-25, 2008