M. Gaedke, J. Meinecke, *A. Heil*; FDX - Federating Devices and Web Applications; Proceedings of the Sixth International Conference on Web Engineering (ICWE), Pages 95-102, Palo Alto, USA, 11-14 July, 2006.

FDX – Federating Devices and Web Applications

Martin Gaedke
University of Karlsruhe
Engesserstr. 4
76128 Karlsruhe, Germany
+49 (721) 608-8076
gaedke@tm.uka.de

Johannes Meinecke
University of Karlsruhe
Engesserstr. 4
76128 Karlsruhe, Germany
+49 (721) 608-8072
meinecke@tm.uka.de

Andreas Heil
Microsoft Research Cambridge
7 JJ Thomson Avenue
CB3 0FB Cambridge, United Kingdom
+44 (1223) 479-0
v-aheil@microsoft.com

ABSTRACT
Electronic devices have been used for the support of everyday tasks in domestic and professional environments for some time now. Currently, there is a tendency towards a combined application of individual gadgets that are connected within locally confined environments via a diversity of protocols and technologies like UPnP, WLAN and Bluetooth. As one step further in this direction, there is the vision of devices that are globally and uniformly connected through the WWW, extending the Mobile Web to a Ubiquitous Web. Hence, this allows for scenarios where actions on one device can trigger events on an arbitrary other device, and where third-party Web services from anywhere in the world are involved. Ultimately, this results in federations of devices and Web services belonging to different households, companies, suppliers and service providers, forming new kinds of Web applications that integrate devices as an additional dimension. To fulfill this vision, solutions are required that are able to abstract from the different device implementations and bridge the gap between the device and the Web. In this paper, we present the Federated Device Assembly (FDX) approach that offers an integrated management platform for wrapping and as such connecting arbitrary devices to enable new forms of Web applications along with the means to model such federations. Furthermore, we demonstrate how the required infrastructure systems can be realized by introducing a reference architecture and a generic Web service interface.

Categories and Subject Descriptors
C.2.4 [**Computer Systems Organization**]: Distributed Systems – Distributed applications; D2.12 [**Software Engineering**]: Interoperability - *Interface definition languages*; H.3.4 [**Information Storage and Retrieval**]*:* Systems and Software - *Distributed systems;* H3.5 [**Information Storage and Retrieval**]: Online Information Services - *Web-based services*

General Terms
Design, Security, Management, Standardization

Keywords
Architecture, Device, Federation, Integration, Service Infrastructure Systems, Web Service

ICWE'06, July 11–14, 2006, Palo Alto, California, USA.
ACM 1-59593-352-2/06/0007.

1. INTRODUCTION
The domestic and professional environments of users in today's information era are strongly influenced by a multitude of services and electronic devices. Moreover, we observe, that the trend of interconnecting those devices has increased within the last years [1]. For example, while in the beginning cell phones were stand-alone devices, a modern smart phone can now be integrated into a network by offering a wide range of communication capabilities. But even if devices have the capability to connect to computers through a network and offer services such as synchronization, they still have very limited capabilities in communicating with the environment. In conventional scenarios, it is often the user, who has to map data from the environment to the device manually in accordance to some business process. However, the real potential benefit is reached when the data from the environment transcends domestic and organizational boundaries, supporting business processes and applications on a federated global scale. For example, the control over devices integrated in the user's environment could then be outsourced to companies providing new kinds of business services.

To realize this potential, the systematic integration of a wide variety of heterogeneous hardware devices into an environment builds the necessary foundation. One well-known industry standard for achieving this is Universal Plug and Play (UPnP) [18]. An example for a UPnP standard is the device specification for HVAC systems[1]. Such a HVAC system includes heating and cooling equipment and can be controlled remotely by other UPnP-enabled devices and software. Several systems can be combined and zoned for large buildings, but controlling the system is restricted to the local network by design. Therefore, the combination with services from providers outside the local network to establish federations is difficult or not possible at all. In practice, most electronic gadgets do not follow a standard like UPnP, but still provide some proprietary capabilities for interconnecting. This high degree of diversity poses an obstacle for the integration and combined usage of devices and hinders their systematic integration. Hence, using those devices in a uniform way to establish ubiquitous business processes remains a problem to be solved.

As global communication platform, the WWW offers the necessary standards for heterogeneous environments and as such enables global device interaction. Once the gap between devices

[1] HVAC stands for Heating, Ventilation and Air Conditioning.

M. Gaedke, J. Meinecke, *A. Heil*; FDX - Federating Devices and Web Applications; Proceedings of the Sixth International Conference on Web Engineering (ICWE), Pages 95-102, Palo Alto, USA, 11-14 July, 2006.

and the Web is bridged, they become building blocks for Web applications that scale from domestic to businesses environments. In this paper, we propose a solution to this gap by introducing a systematic approach to model and federate devices as a basis for novel Web Engineering tasks. In Section 2 the challenges to the integration of devices are investigated and the state of the art is discussed. Following this, we present the Federated Device Assembly (FDX) as an infrastructure building block encompassing a way of describing the functionality of arbitrary devices in a generic and federatable manner. Based on this conceptual approach, a reference architecture along with an interface specification for the realization of FDX is given in Section 4. In the last sections we demonstrate an application of the approach by implementing several concrete FDX, based on different hardware devices. We conclude with a summary of the lessons learned.

2. STATE OF THE ART

In order to realize the interconnection of arbitrary devices, several problems have to be addressed. Thus, different approaches can be found, trying to solve these issues. Each approach focuses on a very specific part of the overall problem. Some try to establish a standard for devices, enabling their *dynamic connection*. Others try to establish a standardized infrastructure including special wires and protocols to avoid *protocol diversity* and thus require devices manufactured especially for this infrastructure. All approaches have in common that a diversity of *description languages* is used to enable a consistent way of controlling the devices. Finally, the capability of *building federations* with these devices across organizational boundaries is still an issue that lies outside the scope of most available approaches.

As underlying technology, several software-based approaches are available: CORBA [13], DCOM [12] and RMI [17] are existing standards that enable the communication with the devices but are not interoperable with each other. These de-facto standards rely either on a specific communication infrastructure, on a platform or on a certain programming language, and can be connected to each other only by additional gateways. In contrast, Web services combine many advantages of the previous approaches and in addition allow a platform as well as programming language independent implementation of a service. Common to all approaches mentioned is the capability to expose the software's functionality to its environment. However, this ability is missing for most devices that are capable of connecting to a computer-based network. The previously named approaches and software standards form the base to approach the challenges described in the following.

Challenge: Dynamic Connection

Within a ubiquitous computing scenario, the infrastructure tends to rapidly change and thus modify the original setup. As devices are frequently attached and removed from a network, a static description of the system's model is impractical. Two approaches addressing this issue are Universal Plug and Play (UPnP) [18] as well as the Jini Network Technology [16]. Both approaches allow a dynamic modification of the infrastructure by providing appropriate descriptions and infrastructure services for attached entities. Each approach respectively each de-facto standard restricts the infrastructure to a certain type of technology. Projects like Shaman [15] or Sindrion [6] try to integrate further types of devices into those infrastructures. In this context, wrappers acting as gateways grant either access to a UPnP or a Jini infrastructure. As a result, only a limited subset of the devices' functionality (UPnP subset respectively Jini subset) is provided within the corresponding infrastructure. In addition, both approaches only focus on a specific class of devices to be integrated. However, these approaches do not intend to connect the different infrastructures.

Challenge: Protocol Diversity

Infrastructures built on software-based standards like UPnP or Jini allow a seamless integration of specific devices into existing network technologies like Ethernet. In addition, each standard provides specific communication protocols. Devices that are to be integrated into this network must implement those particular protocols. Besides software-based approaches, a second type of networked device exists: Usually based on a hardware bus, these systems require additional infrastructure, i.e. hard-wired within buildings. Approaches like KNX/EIB [8] or LonWorks [5] provide their own protocols, predominately on lower levels of the ISO/OSI reference model. Some offer the ability to make use of existing infrastructures like twisted-pair, power lines or IP-based networks. Even if integrated into existing infrastructures, the systems still fall back on their own protocols. The main advantage of these standards is that devices can be added with a minimum of effort. The integration of devices not implementing those protocols is not supported.

Challenge: Description Languages

To support a seamless integration of devices into a network, each device must be capable of describing its own functionality in a well-defined way. Providing this information can be accomplished with the help of an Interface Definition Language (IDL), like e.g. in the case of DCOM or CORBA. In contrast to these proprietary approaches, the W3C published the Web Service Description Language [4], which provides an XML-based notation for describing interfaces and communication patterns for Web services. Nevertheless, it focuses on services, not on devices. UPnP provides an approach that covers an XML-based device description, which is however restricted to predefined sets of device types.

Challenge: Building Federations

Even if devices are accessible within a confined environment, the question remains how to expose their functionality to external partners to found a federation. As a first approach towards federation, the Open Service Gateway Initiative (OSGi) [10] allows third-party providers to make their services available within a domestic network. However, the platform provided by the OSGi is very limited, as it does not support exposing local services to the outside or required aspects of establishing secure inter-organizational partnerships, i.e. for example identity and access management.

We have seen that several approaches exist, each with a strong focus on a specific problem. Considering the existing solutions, we realize that a more flexible approach is needed to lead the Mobile Web to a Ubiquitous Web.

M. Gaedke, J. Meinecke, *A. Heil*; FDX - Federating Devices and Web Applications; Proceedings of the Sixth International Conference on Web Engineering (ICWE), Pages 95-102, Palo Alto, USA, 11-14 July, 2006.

3. THE FDX APPROACH

In this section, we describe the Federated Device Assembly (FDX) approach, which focuses on the federation of arbitrary off-the-shelf devices, attached either to the computer or to the network directly. The approach is founded on a device model, abstracting hardware-specific realizations and allows publishing device capabilities in form of Web services as uniform building blocks for federated Web applications and systems.

3.1 FDX Infrastructure Concept

The FDX approach is based on the description of devices on an abstract level. To find an adequate abstraction level for physical devices we performed an evaluation of a wide range of physical devices with nearly 900 functional features. We were able to segment these features into the three building blocks: Methods, properties as well as events, whereas each of the examined devices can be composed of these three groups.

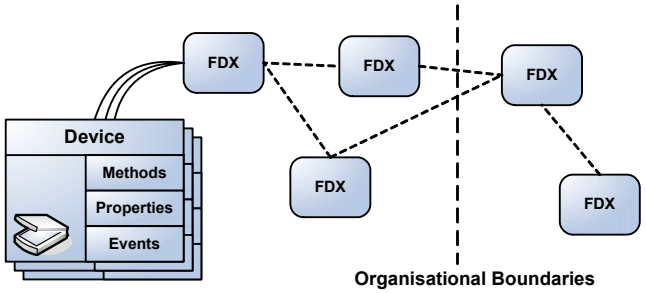

Figure 1. FDX Infrastructure Concept

While this device's functionality usually is bound to its physical environment, a FDX exposes this functionality to the environment of the partners of the federation by wrapping the hardware on a software level. Thus, the FDX must be either located on the computer or on the network connected to the device – acting as mediator between the specific protocols of the device and the standardized Web service protocols. Focusing on this mediation and on Web services based federation protocols, FDX can invoke functionality among each other and even cross the boundaries of organizations, indicated by the dashed lines in Figure 1.

To share the device capabilities among different services, we introduced the concept of the Device Card, a model describing devices based on the building blocks introduced before.

The root element of the model is based on the Dublin Core Metadata Initiative [2]. This de-facto standard for common meta-data attributes provides a wide range of reusable attributes for further use of the model like identifier and title. To allow the modeling of more complex units, parent-child relationship is applicable. Consequently, several sub devices can be part of one device. This also allows the modeling of complex devices such as provided by the UPnP standard. To offer a more flexible approach, the Device Card is extendable with additional information by adding further metadata schemas.

A *status variable* describes a property of a device, providing read-only information. In case of a HVAC system, such as introduced in Section 1, a status variable could be e.g. the current outdoor temperature or the fan speed of the system. Moreover, the uptime or the current error state can be published as status variables. Thus, status variables cannot be manipulated directly from outside of the device. Properties, which can be set as well as methods exposed by the device, affecting internal or external states, are outlined as *functions* in the model above. Within our example, the target temperature could be modeled as such a property. In both cases, a function is invoked directly from outside of the device. *Events* become relevant as soon as state changes within a device occur. They allow to model event channels that enable devices or systems of the federation to subscribe to. In our example, reaching the target temperature could cause the raising of such an event.

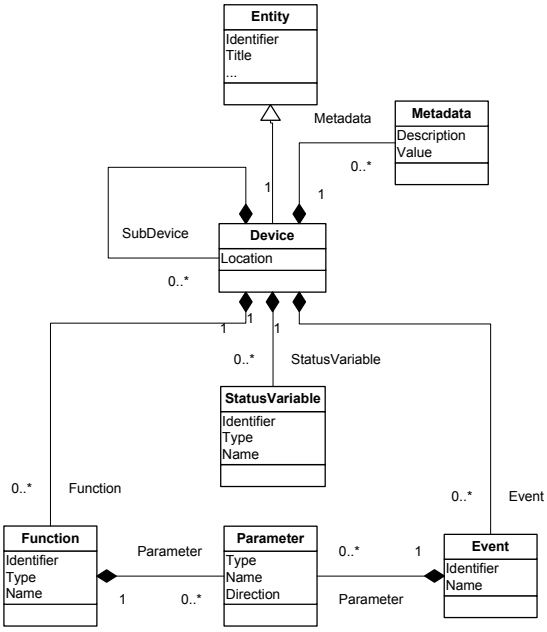

Figure 2. Device Card Model

With an understanding of the abstract concept of devices, we now focus on the integration of such devices in a model towards a Ubiquitous Web.

3.2 Modeling Device Federations

The idea of bringing together Web services and devices supplies the basis for applying already existing Web-based technologies. This includes the area of federated access control, like e.g. WS-Federation [3] or the Liberty Alliance project [9], to support inter-organizational scenarios. An important issue to be addressed in this context is the question of how to model such federations. In particular, descriptions are required that explicitly state, which devices, services, and applications are involved, by whom in the federation they are controlled and how they invoke each other. For this purpose, we have extended our previous work, the WebComposition Architecture Model (WAM) [11], which is an approach to modeling the architecture of inter-organizational Web systems with special focus on federated access control and the mentioned federation specifications. As a main component, WAM comprises an easy to apply graphical notation that we have supplemented with new symbols in order to cover not only Web applications and services, but also devices as subjects of federation. Instead of giving an extensive definition of the model, we limit ourselves to the illustration of the examples introduced in the section before (cf. 1 and 3.1).

M. Gaedke, J. Meinecke, *A. Heil*; FDX - Federating Devices and Web Applications; Proceedings of the Sixth International Conference on Web Engineering (ICWE), Pages 95-102, Palo Alto, USA, 11-14 July, 2006.

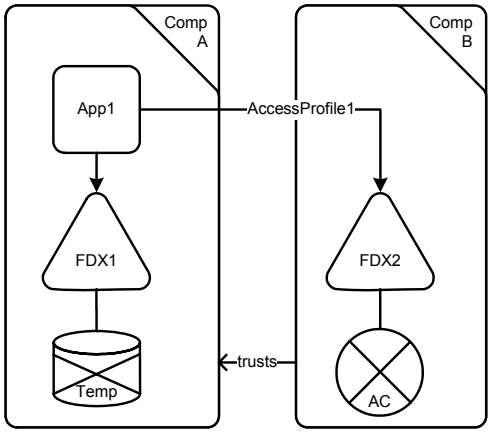

Figure 3. Example WAM Diagram

The example is concerned with a company that has outsourced the control of its air conditioning system to another company that disposes of a network of meteorological stations and can therefore regulate the indoor climate very efficiently. The modeled scenario contains two devices: A temperature sensor *Temp* and the controllable air conditioning system *AC*. WAM distinguishes between sensor devices that only supply information, and actuator devices that also influence their environment. In correspondence to the approach described in the last section, the devices' capabilities are made available with the help of Federated Device Assemblies, modeled as Web services *FDX1* and *FDX2* in Figure 3. These services in turn are integrated into the Web application *App1* that allows the staff of company A to monitor the whole process based on service level agreements (SLA) and configure the climate policy respectively. The technical details about the Web service invocations, e.g. in terms of security specific protocol extensions, remain outside the graphic description. These are described as reusable profiles, which are referred to by labels (*AccessProfile1*). As depicted, the calling application and the called FDX are controlled by separate parties, represented by two different so-called security realms (*CompA* and *CompB*). In order to model the federation of both partners, trust has been established between the two realms of the partners. In this case, realm *CompB* trusts realm *CompA* to use its FDX, probably based on an agreement concluded before. For instance, if a federation-specific protocol like WS-Federation is applied to realize the model, then the trust agreement might include the configuration of a format for security tokens integrated into the Web service messages as well as the exchange of digital certificates so that the tokens can be cryptographically validated. This pre-established trust enables controlled accesses from requestors from *CompA* to the resources at *CompB* without requiring the requestors to have a separate account at *CompB*. One of the design goals of WAM was to abstract from the implementation-specific details in favor of a good overview of the federation structure. For further details about WAM and the underlying semantics, please confer [11].

4. FDX APPLICATION STRUCTURE

After having introduced the basic concepts, we now briefly describe the concrete architectural elements, necessary to build FDX components.

4.1 FDX Reference Architecture

Since devices differ considerably in functionality, we defined a reference architecture for developing FDX (cf. Figure 4). To use the devices functionality, at least direct access to the device driver or access to an adequate remote interface must be available (e.g. UPnP device and service description).

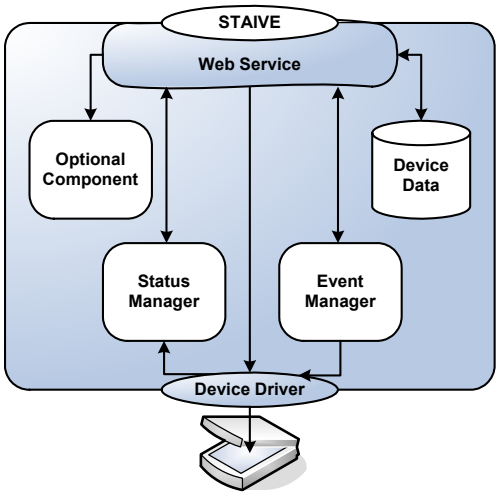

Figure 4. Federated Device Assembly

By design, any device-specific communication technology can be chosen to connect a device to its FDX. Further communication with the device takes place only through the FDX. Thus, the FDX appears as an endpoint for any application accessing the device's functionality. The location of the device is transparent to the calling application, as all communication takes place between the application and the FDX, represented by a Web service.

If the device is not attached to the same platform where the FDX is deployed, e.g. communication takes place over RMI, DCOM or SOAP, the FDX still appears as an endpoint at the same location even if the devices are moved within a system either physically or logically. The necessary mapping between the FDX and the device is the encapsulated secret of the FDX implementation. As such, the FDX can be treated as a black box like any other component or Web service in the context of Web Engineering principles.

As explained before, the basic setup of each device can be reduced to the three elements status variables, events and functions. These different types of information are processe within a FDX using several components (cf. Figure 4). Communication between the device and the so-called *Status Manager* is unidirectional by polling read-only properties of the device. Thereafter, the FDX provides the received data as status variables. Depending on the implementation, the Status Manager can poll a required value on demand or repetitively without external request to cache the values within the FDX. The *Event Manager*, e.g. implemented as stand-alone process, listens to a device's events and hands these over to subscribers of the FDX's event channel.

In order to cache device configurations of attached devices as well as to provide device descriptions through the service, each FDX implements a *Device Data* component. This component realizes database functionality by storing a collection of Device Cards and as such ensures scalability in terms of the number of

M. Gaedke, J. Meinecke, *A. Heil*; FDX - Federating Devices and Web Applications; Proceedings of the Sixth International Conference on Web Engineering (ICWE), Pages 95-102, Palo Alto, USA, 11-14 July, 2006.

devices. For managing only a few devices, the Device Data can be stored in memory. To achieve persistence the information can be stored within data files, e.g. an XML file on the local storage device. For managing many devices, e.g. large-scale sensor networks, the usage of a database management system should be considered.

The FDX Web service offers a generic interface to publish information and receive incoming requests for exposing the functionality of a device or a set of devices. This interface is called *STAIVE* and is further discussed in the following section.

The component-based concept of a FDX allows for adding optional components to handle the attached devices in an appropriate manner. The FDX approach is thus very flexible to wrap certain sets of devices while its composition is not restricted in any way.

4.2 STAIVE Interface

Based on our investigations, we defined a limited set of operations that are necessary to cover the three aspects of a device's functionality. Consequently, the operations within a FDX allow the request and manipulation of status, invocation, and event. In correspondence to this, the proposed interface is called STAIVE. In addition, the FDX supplies the description of each device connected to the FDX, enabling the use of the STAIVE interface. Communication between the client and the FDX is event-driven by design to enable the publish-subscribe pattern.

The complex events that are exchanged in event channels or by invoking methods require a dedicated support for dealing with the magnitude of complexity: This is introduced by the principle of a *context*. The context is used as the single method parameter, allowing for stable method signatures, even though the context may change. It is realized as an XML document enriched with metadata based on the Dublin Core Metadata Initiative (namespace *dc*). Furthermore, FDX-related data is described in a separate namespace (*fdx*). This approach allows for extending the complex events respectively contexts by simply adding new namespaces responsible for transporting additional event information.

Status Variables

The FDX concept allows requesting status variables individually from specific devices. To request a status variable, the operation *GetStatusVariable* as part of the STAIVE interface is called. The context contains an identifier realized as a Uniform Resource Identifier (URI) that allows a distinct identification of the device connected to the FDX and its status variable.

```
<xs:complexType name="VarContext">          (1)
  <xs:sequence>
    <xs:element ref="dc:Identifier"
      minOccurs="1" maxOccurs="1" />
  </xs:sequence>
</xs:complexType>
```

Invocation

Similar to the status variables, the invocation of operations on a device is encoded into the invoke context (2) by a defined identifier handed over to the *InvokeFDX* operation. Again, the identifier allows addressing the device and the corresponding operations to invoke.

```
<xs:complexType name="InvokeContext">       (2)
  <xs:sequence>
    <xs:element ref="dc:Identifier"
      minOccurs="1" maxOccurs="1" />
      <xs:element name="InvokeParameter"
        type="fdx:InvokeParameter"
        minOccurs="0"
        maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
```

To hand over the necessary arguments, each parameter contains a name and a value. The order in which invoke-parameters appear within the context have to correspond to the called operation's signature, which are in turn described in the Device Card. The implementation must verify these values before processing any further actions to ensure the system's robustness against invalid input.

Eventing

The STAIVE interface introduces two operations to *Subscribe* to and *Unsubscribe* from event channels on a given FDX. The subscribe context is made by a pair of two elements: An identifier describing the event to subscribe, as well as the description of a call-back (3).

```
<xs:complexType name="SubscribeContext"> (3)
  <xs:sequence>
    <xs:element name="Event"
      type="fdx:SubscriptionIdentifier"/>
    <xs:element name="Callback"
     type="fdx:Callback"/>
  </xs:sequence>
</xs:complexType>
```

The call-back specifies the operation to invoke after an event occurs. The operation is described using an invoke context for the specified operation (4). Hence, the FDX performing the call-back invokes the operation by sending the context to the invoke operation implemented at the subscribers.

```
<xs:complexType name ="Callback">           (4)
  <xs:sequence>
    <xs:element name="Location"
     type="xs:anyURI"/>
    <xs:element ref ="fdx:InvokeContext"/>
  </xs:sequence>
</xs:complexType>
```

Consequently, the subscriber must implement the STAIVE interface as well, if call-backs are required. In particular, the subscribed FDX can process the call-back easily by sending back the invoke context handed over before. The subscribed FDX must be informed about the address of the call-back endpoint, since this information is not available within the invoke context. Thus, the subscribed FDX does not rely on additional infrastructure services such as UDDI to resolve the subscriber's endpoint.

```
<xs:complexType                             (5)
 name="UnsubscribeContext">
  <xs:sequence>
    <xs:element name="Event"
      type="fdx:SubscriptionIdentifier"/>
    <xs:element name="Callback"
      type="fdx:SubscriptionIdentifier"/>
  </xs:sequence>
</xs:complexType>
```

The unsubscribe context (5) is similar to the one used for subscriptions. Instead of a context for the call-back, only the its

identifier is used. The management of those subscriptions lies within the concern of the FDX and its Status Manager. If a subscriber is not available anymore and the FDX receives an error or timeout due to a call-back, the Status Manager can clear the subscriptions to keep the internal state up to date.

5. FDX Applied

In order to validate the concept of the Federated Device Assembly, two FDX for different classes of devices have been implemented. One proofing the concept of devices directly connected to the network and the other for devices attached directly to a computer. The first covers the UPnP specification and allows connecting remote UPnP devices through the FDX located on one computer on the network. The FDX takes care of detection and management of available remote UPnP devices within the UPnP network and exposes their functionality (by mediation) to non-UPnP devices through the STAIVE interface.

The second one described in the following was implemented for a set of electronic devices attached to a computer directly, so-called Phidgets [14]. Phidgets are basically low cost electronic building blocks connected to a personal computer via the Universal Serial Bus (USB), providing sensing and controlling capabilities. The manufacturer provides an intuitive Application Programming Interface (API) to control the devices. However, the scope of the devices is the connected PC. Communication to other devices is not intended, making them good candidate for the FDX approach.

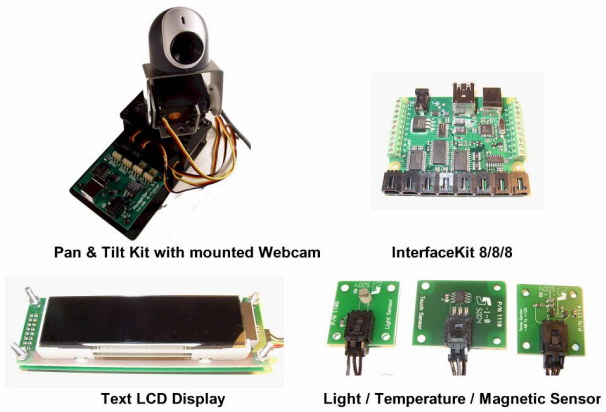

Figure 5. Supported Phidget Devices

Several different types of devices are supported by the Phidget FDX. As an example for actuator devices, the *Phidget Text LCD* was used. This component provides two lines of text, with a maximum of 20 letters per line. Besides displaying text, operating the background light as well as several cursor operations are supported by the device. Another device is the *Pan & Tilt Kit* with a mounted Webcam. The Phidget *InterfaceKit 8/8/8* board was chosen to support sensor input. This board supports up to 8 analog and 8 digital input sensors. In addition to the InterfaceKit, several sensors are supported by the FDX. In Figure 5, several of the supported sensors are displayed.

Based on the recommendations introduced in Section 3, we developed a FDX for Phidget components (cf. Figure 6).

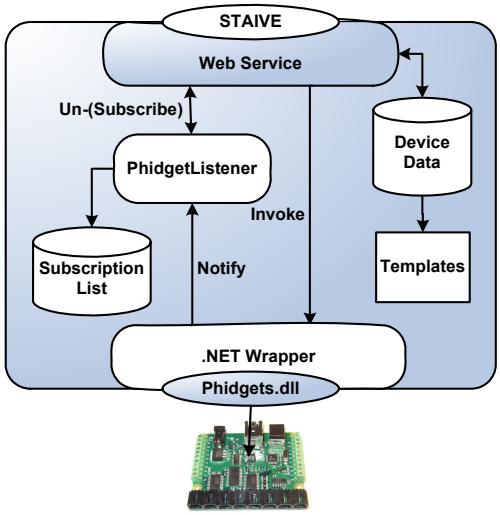

Figure 6. Phidgets FDX

The manufacturer offers a Microsoft Windows 2000/XP driver as well as a Linux driver available as pre-release code. In addition, libraries are available to access the devices using the Microsoft .NET 2.0 Framework. Falling back on the Windows driver, we implemented the Phidget FDX using the Microsoft .NET Framework 2.0. The FDX was designed to support a wide range of different Phidget devices. For each device type, a template based on the Device Card model was created. When a device is attached the corresponding template is loaded, processed and stored within the Device Data component. During this process, the template is filled using the device's name, serial number etc. As a device is removed and attached again, the stored Device Card is identified and used instead of processing a template again.

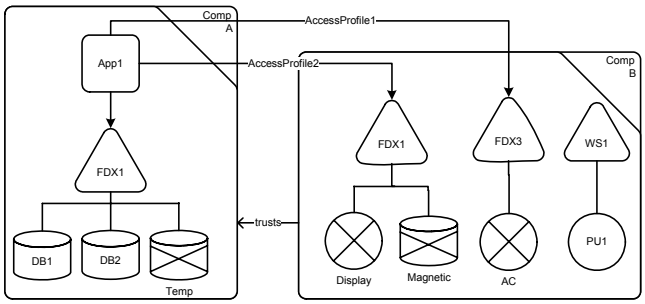

Figure 7. WAM diagram with Phidget Devices

The Event Manager, as described in Section 3, is realized as a separate process (called PhidgetListener in the following). The PhidgetListener reacts on every event fired by the attached Phidget devices, including recently attached and removed devices. After a device raises an event, the PhidgetListener examines the Subscription List, a locally stored mapping of subscribed events and call-backs to perform. Furthermore, the PhidgetListener takes care of subscribing and unsubscribing requests, received through the FDX's STAIVE interface.

Based on the devices supported by the Phidgets FDX, we extend the example from Section 3.2 by adding additional services and devices (cf. Figure 7)

The example shows the service of *CompA* extended by several databases to store and retrieve weather information. In addition,

the provider can send information directly to the display through *FDX1* to inform the customer on *CompB's* side about the current cumulative energy consumption costs. Besides the information collected from the internal meteorological sensors of *CompA*, it is possible for the provider to check the current state of opened windows magnetic switches in *CompB's* estates. Consequently, the system can be controlled more effectively in terms of economic and ecologic factors. Furthermore, the model addresses access restrictions to a dedicated Web service (*WS1*), which cannot be accessed by users at *CompA*.

Figure 7 stresses the analogy of databases and sensor devices within WAM. Both entities, software as well as hardware, provide data to the distributed system. Considering read-only access to a database, no particular difference can be detected if the information beyond the Web service is accessed by another service or application. In addition, the actuator devices, e.g. the display and AC system in the example, can be seen as an analogy to process units (both perform actions). Thus, it is possible to bridge the gap between physical devices and the software services in modeling federated environments.

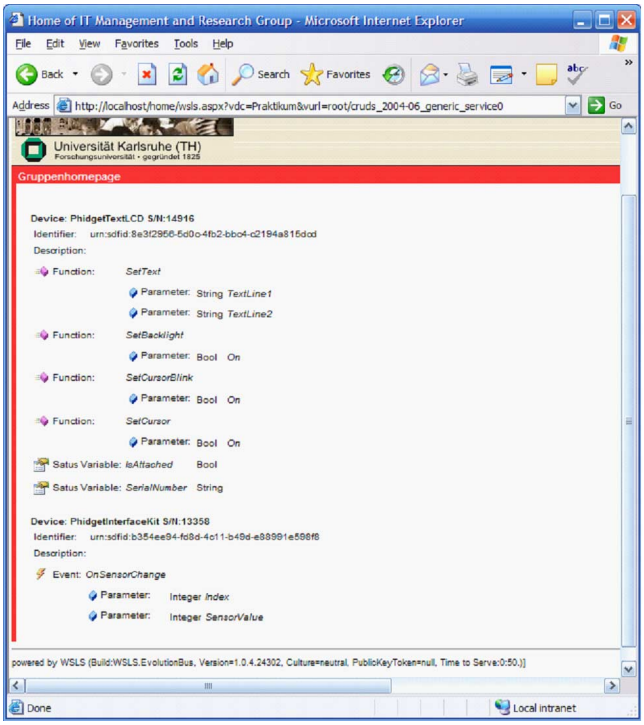

Figure 8. FDX Integration into the WSLS

A further step was to integrate the new FDX into an existing infrastructure to proof the federated approach. For this reason, the Web Service Linking System (WSLS) [7] was used. This service-based framework support the seamless integration of Web services and as such, the integration of FDX. By configuring a presentation component, it was possible to visualize the available functionality of all devices within a federation (cf. Figure 8).

6. CONCLUSION AND FUTURE WORK

In this paper, we presented the Federated Device Assembly (FDX) as an approach to enhance Web applications by extending their interaction dimensions with device capabilities in a federated environment. We introduced a generic interface and a flexible way of describing devices to allow for arbitrary gadgets to be connected to different systems. The service-related approach, together with corresponding federation mechanisms such as WS-Federation, allows offering this functionality as services beyond system or organizational boundaries.

We introduced the basic concept of describing devices as a composition of operations, properties, and events. Based on these descriptions, we developed the STAIVE interface, the key concept for functionality within the FDX. Thus, the functionality can be provided to federation partners. With respect to modeling Web applications, we extended the WebComposition Architecture Model (WAM) to engineer Web applications on an abstract level including applications, services as well as devices. The integration into the Web Service Linking System (WSLS) showed a first proof of concept in integrating devices into existing, federated environments. By developing the FDX approach, we have demonstrated that devices can be integrated into a federated system, even if the devices are initially not designed for this purpose. Furthermore, the FDX approach allows using devices during the process of Web Engineering in the same manner as any other service.

The FDX concept was applied for two very different classes of devices. Thus, it was shown that a wide range of devices is covered by this approach. We are planning to extend the available number of FDX to widen the base for further device integration. Further investigation will also include additional hardware-specific issues, such as modeling accounting for energy consumption, bandwidth and resource usage in general. Finally, the currently established event mechanisms will be extended to allow for applications based on more complex inter-device relationships.

7. REFERENCES

[1] Accenture, Digital Home Solutions: Issues, Trends and Consumer Insights. 2005

[2] Andresen, L., Dublin Core Metadata Element Set, Version 1.1: Reference Description - Recommendation. (2004): http://dublincore.org/documents/dces/ (15.02.2006).

[3] Bajaj, S., et al., Web Services Federation Language (WS-Federation) - IBM Developer Network (2003): http://www-106.ibm.com/developerworks/webservices/library/ws-fed/ (14.10.2004).

[4] Christensen, E., et al., Web Services Description Language (WSDL) 1.1. 2001

[5] Echolon Corporation, Introduction to the LonWorks System. 1999.

[6] Gsottberger, Y., et al., Sindrion: A Prototype System for Low-Power Wireless Control Networks, in 1st IEEE International Conference on Mobile Ad-hoc and Sensor Systems. 2004: Fort Lauderdale, Florida, USA

[7] It-Management and Web Engineering Research Group (Mwrg), WebComposition Service Linking System - Website (2006): http://mw.tm.uni-karlsruhe.de/projects/wsls/ (30.11.2005).

[8] Konnex Association, System-architecture. 2004

M. Gaedke, J. Meinecke, *A. Heil*; FDX - Federating Devices and Web Applications; Proceedings of the Sixth International Conference on Web Engineering (ICWE), Pages 95-102, Palo Alto, USA, 11-14 July, 2006.

[9] Liberty Alliance Group, Liberty Alliance Specifications - Website (2004): http://www.projectliberty.org/ resources/specifications.php (15.02.2006).

[10] Marples, D. and Kriens, P., The Open Services Gateway Initiative: an introductory overview. IEEE Communications Magazine, 2001. 39(12): p. 110-114.

[11] Meinecke, J. and Gaedke, M. Modeling Federations of Web Applications with WAM. in Third Latin American Web Congress (LA-WEB 2005). 2005. Buenos Aires, Argentina: IEEE Computer Society. p. 23 -31.

[12] Microsoft Corporation, DCOM Technical Overview - MSDN Library Article (1996): http://msdn.microsoft.com/library/en-us/dndcom/html/ msdn_dcomtec.asp (15.02.2006).

[13] Omg, Object Management Group - Website (2006): http://www.omg.org/ (15.02.2006).

[14] Phidgets Inc., Phidgets - Website (2003): (15.02.2006).

[15] Schramm, P., et al., A Service Gateway for Networked Sensor Systems, in IEEE Pervasive Computing. 2004. p. 66-74

[16] Sun Microsystems Inc., Jini Architecture Specification. 2003

[17] Sun Microsystems Inc., Java Remote Method Invocation (Java RMI) - Website (2006): http://java.sun.com/rmi/ (15.02.2006).

[18] Upnp Forum, UPnP Forum - Website (2006): http://www.upnp.org (06.02.2006).