A Framework for Advanced Home Service Design and Management

Félix Jesús Villanueva, David Villa, María José Santofimia, Francisco Moya, and Juan Carlos López, *Member*, IEEE

Abstract — *In this paper a distributed object-oriented framework (DOBS, Distributed Object Based Services) for home service design is presented. This framework eases the development of advanced services able to run in a variety of devices ranging from tiny wireless sensors to powerful multimedia servers. Special emphasis will be made on DOBS main features, its core components, the offered common services and the supporting tools developed to allow users and manufacturers to easily build advanced DOBS compliant services[1].*

Index Terms — Service architecture, distributed services, integration architecture, home services.

I. Introduction

The lack of a common framework for home service development has been largely discussed in the last years. One of the main problems identified is the heterogeneity and diversity of devices and services found in those environments.

Our proposal is built on the well-known Distributed Object Oriented (DOO) paradigm as the cornerstone for modeling, in a more efficient way than existing solutions, any kind of service one can find (or envision) in home networks. Special care has been taken in the way required resources and network bandwidth can be optimized. In DOBS, all services are distributed objects that may run on devices with very little resources as well as on personal computers. The manufacturers may also provide efficient ad-hoc hardware implementations of those objects. Additionally, a complete set of tools has been created, that allows manufacturers and even users to create advanced services hiding most middleware particularities.

The remaining of this paper is organized as follows. In the next section, we will shortly introduce the related work that deals with frameworks for home services. In section III, we will analyze the key requirements which guide the design of the framework. We then introduce the core of the DOBS framework in section IV, and the DOBS common services in section V. Section VI outlines the development process of DOBS compliant services. Section VII describes our prototype implementation. Finally we draw some conclusions and highlight relevant future work.

II. Previous Work

A quick analysis of the market of networked residential services leads to the following types of end products:

- Services in a box: This type of solution include the devices, software and all necessary elements for service operation. They are closed solutions with dedicated devices and software that may rarely be used for different purposes. This is probably the simplest solution and also the most widespread in the current market.
- Based on a residential gateway: A residential gateway is a central device which interconnects the different devices and data networks in the home and provides external connectivity. Service providers, and specially telecom operators, are mostly interested in this approach since it allows a centralized control of the service distribution points. Unfortunately the residential gateway constitutes a single point of failure, which makes it inadequate for many services. Besides, this solution scales badly when the number of technologies involved increases.
- Customized installations: These are generally the most expensive solutions and are typically represented by full custom projects for large buildings (e.g. intelligent buildings).

A common problem for all these solutions is that they link users to a single provider or technology. This fact makes it difficult to expand and reuse services and devices even in the same environment. A proper solution resides in an intermediate middleware that isolates services from devices and technology.

Several research efforts have been made in order to integrate services. One of the most used solutions is the OSGi architecture [1] and related works [2][3]. OSGi is a Java based platform for service management centered in the residential gateway model. It follows a centralized model where the bundles (OSGi components) may either use general purpose services (e.g. log, configuration, etc.), share modules, or manage security aspects, for example. Although OSGi provides a solid ground for service management it does not deal with the devices as part of a distributed environment.

Other approaches like [6] and [7] use CORBA (probably the best-known large-scale object-oriented distributed middleware) for service development. In [6] an IEEE 1394-based home environment is presented, using CORBA to build an IEEE 1394 driver that keeps real time properties, while [7] shows a quite generic description of a client-server structure. None of these approaches contribute with tools or establish guidelines for service development and they do not allow service management either.

Other former approaches focus on the integration of different middlewares for home networking, enabling a transparent interaction between them. In [4] a universal middleware bridge is designed creating XML templates for services and instantiating virtual services in each domain to be

[1] This work has been partly funded by the Spanish Ministry of Industry under project CENIT Hesperia.

All authors are with the Department of Information Technologies and Systems, University of Castilla-La Mancha, Spain. School of Computer Science, Paseo de la Universidad, nº 4. 13071, Ciudad Real, (Spain). e-mail: {felix.villanueva, david.villa, mariajose.santofimia, francisco.moya, juancarlos.lopez}@uclm.es.

integrated. This solution is also a centralized solution (based on a home server) with problems of scalability and reliability. A distributed agent-based approach is described in [5]. In this work, and by means of a scripting language, the developer can use services from different middlewares. The communication between different agents is TCP/IP based.

Our approach, as we will see in next sections, takes a more natural approach to network communications, and, using a distributed object-oriented middleware, allows the developers to create distributed applications and services in a transparent way.

III. Problem Statement

Even though devices are increasing their performance and interoperability capabilities (with several communications interfaces), the development of new home services is taking place much slower than we could expect.

The enormous efforts of the research community to avoid the interoperability problem have not sufficiently favoured the transfer from research centres to massive market.

Our framework design guidelines are derived from the requirements identified by different actors involved in the home services market. We can see these actors, their role in the home services market and the middleware requirements in Fig. 1.

Service developers require supporting tools in order to minimize the development time regarding aspects related with the middleware; that is, they need to focus their attention on service functionality instead of on how services interact with each other. They also need flexibility regarding how (i.e. which development language) and for which platform (i.e. operating system) they can develop services. They finally claim a set of common services with basic functionalities (e.g. log service, configuration method, etc), as well as a well defined set of interfaces to access to the rest of services.

Devices manufacturers tend to reduce the device resources so as to decrease the device final cost. This means that fewer resources are available to run the middleware and the associated services. So, the more lightweight the middleware is, the more competitive their devices can be.

Besides, *service providers* usually require remote management capabilities, such as service deployment, upgrades and remote control of services. This is why they are mainly interested in a remotely accessible centralized service container, the residential gateway.

Additionally, *service integrators* need to aggregate third party services with minimal development and configuration activity. They should select services according to their functional and non-functional properties, without having into consideration the technologies behind the service.

Finally, the *users* only care about service functionality, and not about technologies, protocols, software, etc.

Together with the middleware requirements imposed by the different actors we should also consider the future infrastructure of home environments. Devices (and their associated services) are going to experiment relevant changes soon. In this sense, wireless communications are taking a main role, as wireless sensor networks (WSN) seem to be a good interface between the physical world and services, and the number of mobile multimedia devices is increasing. Besides, the home environment and the rest of the world are now connected using more than one interface (not just through a residential gateway).

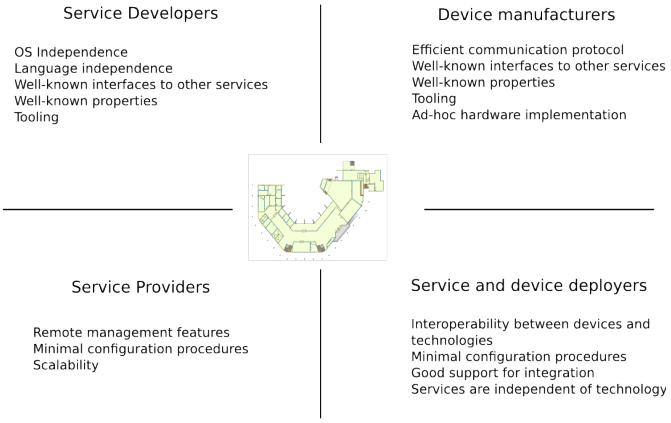

Fig. 1. Middleware requirements from different actor's point of view.

Taking into consideration all these aspects, we decided to use a DOO middleware as a basic technology to create a development framework for home service design and management (DOBS). In general terms, a DOO middleware defines APIs, a communication protocol, and an object/service information model to enable heterogeneous distributed applications (as in the case of home environments). Examples of this type of platforms are CORBA, ZeroC Ice, Jini, etc. The advantages of object-oriented programming are well-known and DOO middlewares have a long history of successful applications in many business domains.

IV. DOBS Core Components

DOBS core components have been designed having in mind the requirements mentioned above. The main components are built on top of a DOO middleware, however some of them may need (to reach a better performance) to interact directly with low level system layers, as the communication protocols or the device drivers. DOBS architecture is shown in Fig. 2. In next sections, a brief description of every component will be given.

A. DOBS interfaces

To model basic monitoring (temperature, humidity, presence, etc.) and control (lighting, door locks, etc.) services, a set of interfaces enabling basic "read" and "write" operations has been defined. They also implement a composition mechanism which allows the modeling of arbitrarily complex services.

For audio and video services the *AVStreams* interfaces from the Object Management Group (OMG) have been adapted. These interfaces, designed with the industry consensus, provide a standard mechanism to configure and control multimedia flows between sources and sinks.

B. Common services

Common services are aimed at:

- Providing common functionality used by the vast

majority of services (e.g. service discovery -see ASDF later-, security mechanisms, etc.)

- Establishing a universal way to perform specific operations such as integrating a new device (bootstrap service) or managing services (stop, start, actualize, etc.).

All common services are available to the remaining home network services. DOBS common services will be considered and explained in later sections.

C. Integration subsystems

These are specific subsystems that allow seamless integration of services from other platforms. As an example, subsystems to integrate UPnP, X10 or Bluetooth services have been developed. Each subsystem is specific for each platform but they use common services in order to integrate them.

We may see a simple example in the X10 integration subsystem in which each X10 device is represented by a distributed object implemented by means of DOBS interfaces.

Additionally, the integration provides the X10 domain with service discovery capabilities (using ASDF as we will see later). Therefore we may use any X10 device as a distributed object and enrich the X10 domain with characteristics that it does not originally enjoy.

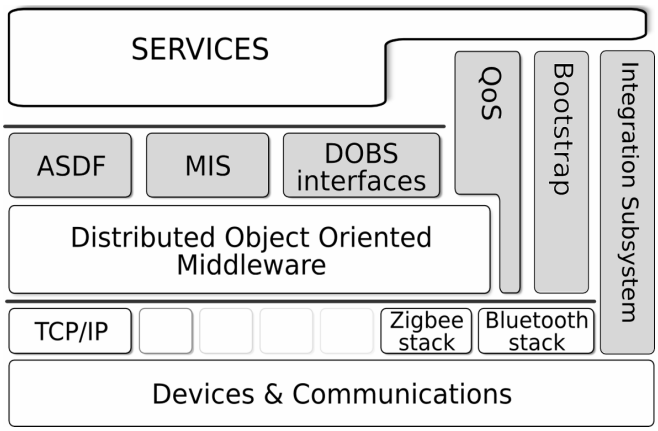

Fig. 2. DOBS framework overview.

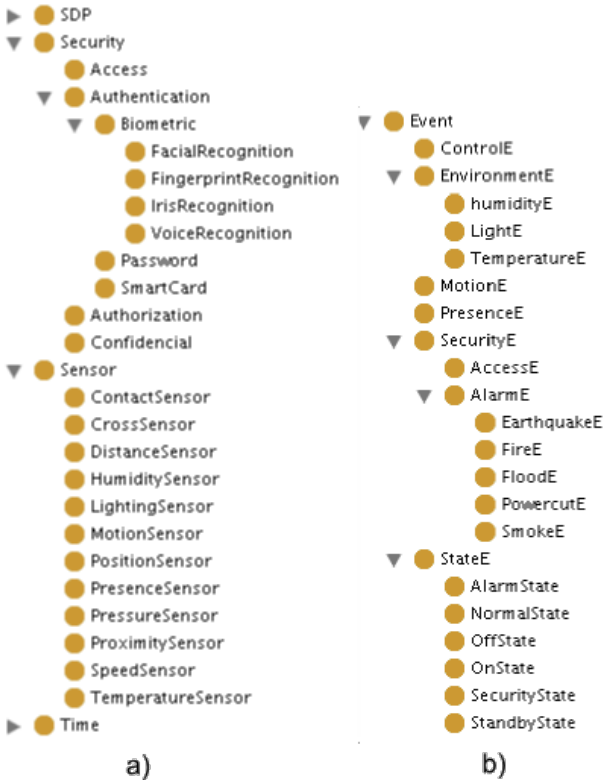

Fig. 3. Partial view of the service (a) and the event (b) taxonomies.

D. Information model

A complete taxonomy of services (with their attributes) and events has been developed so as manufacturers can have access, using a common nomenclature, to the available services, events and their corresponding features (Fig. 3). This taxonomy was first derived from UPnP templates and Bluetooth profiles and it has been completed with other services and properties from most relevant standards (Mobile Location Protocol from OMA, AVStreams from OMG, etc.).

Together with the service type, the taxonomy collects the available attributes for each service. The idea behind this is to create a basic set of home services that works as POSIX interfaces do for operating systems. In fact, we already have developed a compiler which takes this taxonomy and generates a candidate interface for every service intended to be integrated in DOBS. In this way, developers may know in advance the interface offered by any DOBS service.

The *event* taxonomy enumerates a set of common events that may occur in the home environment. Service developers should consider the service behavior for each type of event (at least for those which, due to their importance, require special attention).

This taxonomy has been implemented by means of an ontology in such a way that each service corresponds to a class (with attributes). In order to integrate other domains, each class has instances (similar to the class-object correspondence in software engineering) which represent concrete services in specific domains. This information is used by the service discovery framework (ASDF, see later) to establish correspondences between our framework and the remaining domains (e.g. UPnP, Bluetooth, etc.).

The designed ontology is the core of the *Model Information Service* (MIS) which is used by the rest of the infrastructure in order to get information about services, attributes, etc. The MIS interface is:

```
module MIS {
  dictionary<string, string> AttrDict;
  sequence<string> list;

  interface DomainTranslator{
    string translateSv(string serviceID,
                       string orgDomain,
                       string dstDomain);
    AttrDict attributesOf(string serviceID,
                          string domain);

    AttrDict translateAttr(string serviceID,
                           AttrDict orgAttr,
                           string orgDomain,
                           string dstDomain);
  };

  interface Metamodel{
    PropertyService::PropertySetDef*
      getProperties(string service);
  };
};
```

The interface *DomainTranslator* is used by the integration subsystem in order to get the correspondence between the DOBS model and any other domain introduced in the ontology. With the *translateSv* method we may get the identity of any service in any domain. For example, in UPnP there is a template for a service with the name "DigitalSecurityCamera: 1" which corresponds with an entity of class "Camera" in DOBS terminology. When the UPnP integration subsystem receives an announcement in the UPnP domain, it will ask the MIS (using the *translateSv* method) about the corresponding service in DOBS. Similar procedure is done for attributes with the *attributesOf* method, which provides a list of attributes of a given service, and the *translateAttr* method, which translates attributes between domains.

Finally, the *Metamodel* interface is used to get the properties of a given service, so as to provide with introspection capabilities.

V. DOBS Common Services

DOBS common services reduce the required configuration procedures and provide frequently used facilities to service developers.

A. Abstract Service Discovery Framework (ASDF)

The ASDF allows easy integration in a DOBS environment of almost any existing model of service discovery. It has been specially designed for easy interoperability with many other well-known service discovery protocols (SDP) such as UPnP SSDP, Bluetooth SDP, SLP, etc. Table I shows the ASDF primitives and their corresponding primitives in other SDP platforms.

As shown Table I, we provide primitives to implement almost any model of SDP (directory based, multicast, hybrid, etc.). Nonetheless, the required interfaces are quite simple, which makes it easier to embed ASDF in extremely small devices:

```
module ASD  {
  interface Listener {
    idempotent void adv(Object* prx);
    idempotent void bye(Ice::Identity oid);
  };
  interface Search {
    idempotent
    void lookup(Object* cb, string tid,
                PropertyService::Properties query);
    idempotent void discover();
  };
  interface PropHldr {
    idempotent
    PropertyService::PropertySetDef* getp();
  };
};
```

The *Listener* interface is used to announce/disconnect a service to/from the environment. With the *adv* operation the service announces itself publishing its reference (*prx*). The *bye* operation requires only the identity of the service (an URI like description) in order to notify its disconnection from the environment.

The *Search* interface is used to look up services or to discover the whole environment. The *lookup* operation

TABLE I
ASDF CORRESPONDENCE WITH OTHER SERVICE DISCOVERY PROTOCOLS

ASDF	UPnP SSDP	JINI	SDP (Bluetooth)
Search Directory	No directory based.	Search Lookup Service: Multicast Request	ServiceSearchRequest ServiceSearchResponse
Find Service in a Directory	No directory based. ≈ Multicast SSDP messages	Lookup in a lookup service	~ ServiceSearchRequest/Response ~ ServiceAttributeRequest/Response
Find Service without Directory	Multicast SSDP messages	Not supported	ServiceSearchRequest/Response ServiceAttributeRequest/Response
Advertisement	Multicast SSDP messages(URL with device Description)	Announce protocol (lookup Service)	~Register service in the local sdp server
Registration	~ Advertisements	Registration in a lookup Service	Register service in the sdp server
Subscription	~Subscription to GENA events	Not supported	Not supported
Renew Lease	Application policies not Supported.	Lease Renewal Manager	Not supported
Disconnect	Bye bye SSDP message	Remove/Cancel Leasing in lookup service	Not supported

searches for any service of a given type (*tid*), which meets a set of constraints given as key-value pairs (*query*). Any service that meets the requirements should send an announcement (*adv*) to the callback object (*cb*). The *discover* operation forces every service in the environment to send an announcement.

Finally the *PropHldr* interface (and invoking the *getp* operation) is used to get all the properties of a given service.

The ASDF implementation follows the event oriented paradigm. In any environment, there are, at least, four event channels named ASDA, ASDL, ASDB and ASDD for announcements, lookups, byes and discoveries, respectively.

Depending on the intended semantics, all services must connect to one or more of these channels in order to receive or send events. For example, in the case of a directory based environment, the directory service must implement a yellow pages service in the lookup (ASDL) and discovery (ASDD) channels.

Additionally, each integration subsystem may either easily translate different service discovery protocols to ASDF semantics, or even add service discovery capabilities to subsystems which do not usually support them (e.g. X10 subsystem).

B. Bootstrap

While ASDF reduces configuration procedures for services, the bootstrap service enables a Place & Play philosophy for devices.

When a device is attached to the network, it requires several parameters in order to be integrated in the environment. Besides, its services need some initial information such as, for example, the location of the ASDF event channels, the QoS environment profiles, etc.

The bootstrap service is a multicast service which selects, among the available devices, a coordinator that will be the responsible for starting basic services for the remaining environment (e.g. the event channel manager). The interface of this service is also extremely simple to allow easy embedding in small micro-controllers:

```
interface bootstrap{
  void coordinator(int i, object *prx);
  void lookup();
}
```

When a device is switched on, its bootstrap service sends a multicast lookup invocation to find the coordinator (or coordinators). If there is already a device or computing element which became a coordinator, it invokes the coordinator operation in the newly attached device with its identifier (there could be more than one) and a reference to the *environment manager* (*prx*) implemented by the coordinator. If there is no answer after a timeout, the device itself may request to become a coordinator (it is also a multicast invocation) and start the *environment manager*.

C. Security

Our security infrastructure combines transport-level mechanisms such as secure socket layer (SSL) communications with the ability to inspect and manipulate directly digital certificates. It is also possible to attach a per-invocation user context that may be used for sending credentials or one-time passwords to a remote service. This enables DOBS to be used in more complex environments that require a high degree of security such as public spaces (airports, railway stations…) or critical facilities (industrial plants, power stations…), being adequate in general, for homeland security applications.

D. QoS

By means of profiles, the network resources are assigned according to the state of the environment (day/night, normal, intrusion alarm, etc.). Most of the former approaches to quality of service (QoS) in home environments were focused on user preferences or service requirements. In DOBS we add a new point of view: the environment resources (mainly the network bandwidth) are subordinated to the environment state. For example, if a fire is detected and a *FireE* (see Fig. 3) event is generated by a service, a QoS component in every device will modify the network stack configuration in order to give more resources to the traffic associated to security services (doors and windows control, security cameras to track the evolution of fire, etc.).

In order to achieve this, we follow a *differentiated service* approach (*DiffServ*) as defined by the IETF [8] where the traffic generated by each service is classified in a set of traffic classes. Each type of traffic has a set of resources assigned that can be modified in a dynamic way. The following entities have been defined:

Environment profile: It is a pre-established bandwidth assignment. It is composed of three main parts: a list of events which would trigger the profile, the percentage of bandwidth associated to each type of traffic and a service specification which characterizes the traffic generated by each service and its classification.

QoS Service: It takes environment profiles and configures the network interface according to such specification. It must also mark the outgoing traffic for each service (using the IP TOS field) according with the traffic types that have been defined in the environment.

E. Service management

DOBS includes a mechanism for service deployment, configuration, upgrade, etc. enabling both centralized and distributed management.

Using a grid-computing point of view, each device has a *service manager* which enables remote service management. In the master device (the coordinator selected by the bootstrap service), there is an *environment manager* running. There is at least an *environment manager* for each environment which can be used to start, stop or upgrade all the services that are registered in the *service manager* of each device. The bootstrap service sends to each *service manager* the location of the *environment manager* so that each service manager is able to notify its services and register them in the *environment manager*. Although there is a single logical instance of the *environment manager* there may be a replicated service to improve overall fault tolerance. The bootstrap service may use multiple coordinators to instantiate a replicated *environment*

manager.

VI. DOBS Development Process

The information model depicted in Section IV constitutes the starting point to help developers in the design of DOBS compliant services. It includes the different relationships between services as well as other important semantic information (e.g. correspondences with other domains, such as UPnP or HAVi, for interoperability purposes). It goes beyond the possibilities of other approaches such as, for example, those using templates VIII.

Fig. 4 shows the DOBS development toolchain. It is mainly focused on hiding most of the complexity of the networking infrastructure so that developers may spend most of their time and resources implementing service functionality and not communication procedures.

From the described ontology, and using the DOBS OWL compiler, a candidate interface for the selected service can be obtained. It is expressed by composition of basic DOBS interfaces in an interface definition language (IDL). This specification constitutes the contract between the service and the client that wants to use it.

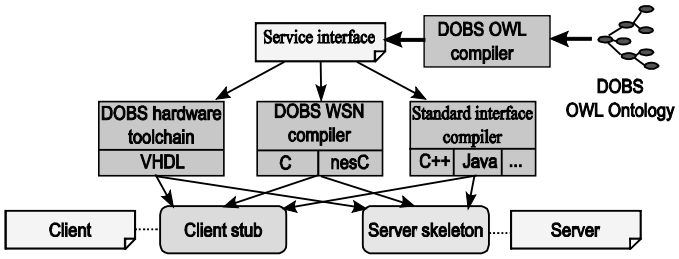

Fig. 4. DOBS toolchain diagram

From this specification and using the IDL compiler, developers get the stubs and skeletons for both, client and server, in the desired programming language. We have also developed tools that allow the generation of stubs and skeletons for services and clients that are intended to run on small microcontrollers (e.g. WSN devices). It is also possible to automatically obtain a hardware implementation (VHDL). More information on these two possibilities can be found in VIII and VIII.

The DOBS tools (DOBS OWL compiler, DOBS VHDL compiler and DOBS WSN compiler), together with the native compiler of the object-oriented middleware selected for the implementation, compose a complete toolchain (Fig. 4) able to generate in an automatic way service implementations (servers) for a variety of scenarios (medium size computers such as set-top-boxes or residential gateways, small processors such as WSN nodes, or even ad-hoc hardware versions). These tools allow the developers to get rid of annoying service communication details (a very error-prone development task) while better focusing on the service functionality itself.

On the other hand, from the client-side developer's point of view, the access to a specific service is transparent and can be dealt with in the same way, no matter whether it is implemented in a residential gateway, in a WSN device or in hardware.

VII. Prototype

We have selected the Internet Communications Engine (Ice) VIII from ZeroC for the DOBS implementation. Ice is a CORBA-like middleware that uses a specific protocol (ICEP) and a specific interface description language (Slice).

Regarding the use of the DOBS common services, templates and examples of use are available, so the developer can focus on the service functionality. Using these common services, the final service implementation can be improved in some general (but important) aspects, such as security, management, and so on.

In order to show the features of the proposed middleware, a set of user services, integrating different types of devices, has been developed. Some working scenarios developed are:

- A generic server for RTSP cameras (VCC4 and AXIS).
- A presence detection platform with mica2 WSN devices.
- Integration of X10 devices adding SDP capabilities.
- Integration of the UPnP SSDP protocol (used by AXIS cameras).

For monitoring, controlling and debugging purposes, an *Inspector* software has been developed. With this software we may see any service in a DOBS environment. Debugging and monitoring task can be done in a generic way as we can see in Fig. 5.

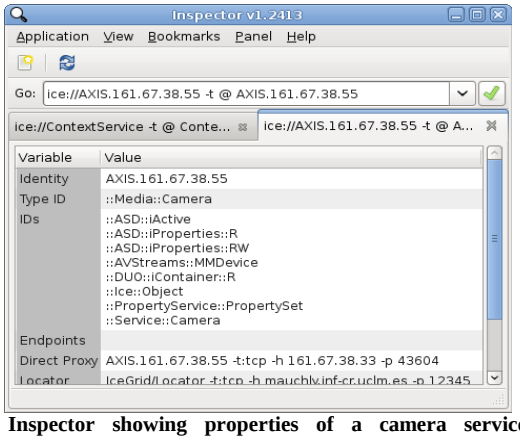

Fig. 5. Inspector showing properties of a camera service using introspection capabilities.

On the left-hand side of Fig. 6, a list of services announced by ASDF is shown. Most of the services are AXIS cameras using UPnP to announce their activation. Therefore, the UPnP integration subsystem has to instantiate a server of type "camera" (according with the information model) and send *adv* operations to the ASDA event channel (as described in section V). The *inspector* only has to listen to the *adv* operations in this event channel and access the services in order to check out whether they are active. The two services labeled with PIR1 and PIR2 are WSN devices with a movement sensor attached. The red button indicates that PIR2 service is not active at this moment.

From the *inspector* point of view (also from any client point of view) there is no difference between a service embedded in a WSN device or running in a conventional PC.

On the right-hand side of Fig. 6 a plugin for a camera service is showed. From the plugin point of view all cameras have the same interface and the same properties, and they can

be accessed with location, language, operating system, etc. transparency.

VIII. CONCLUSION

While OSGi provides a Java-based platform for service management without any consideration about user services, and UPnP offers templates for services without management service procedures, DOBS incorporates both aspects and also provides a common set of services to help in the development of advanced services.

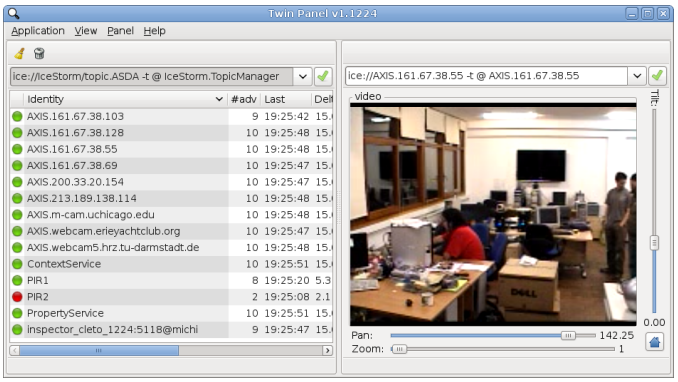

Fig. 6. Screenshot of inspector with the camera plugin activated.

Besides, DOBS avoids the use of virtual machines, XML parsers and web servers, what has a clear impact on the final resource requirements. This is a key feature regarding the final implementation cost, especially considering the type of devices that have to support home services (white goods, cook machines, sensors, actuators, etc.). Finally DOBS provides a toolchain that eases the implementation of both, the service itself (in a variety of alternatives) and the client that is supposed to use it.

DOBS also uses a service discovery mechanism (ASDF) and a bootstrap service that enable Place & Play environments in which minimum configuration procedures are required.

As a future work we will extend the set of DOBS compliant available services and work in automatic service composition in order to infer and compose services according to the user needs or even the environment requirements.

The DOBS architecture is the middleware selected for the HESPERIA project. HESPERIA gathers seven companies and eleven universities and research centers working together on the development of integrated services to provide security and operation control in public spaces (homeland security). In this application field, the DOBS architecture is proving its efficiency and flexibility integrating software from third parties.

DOBS architecture represents a solid ground for advanced home services development and management including all requirements identified by the different actors involved in this market.

REFERENCES

[1] OSGi Alliance, "OSGi Service Platform Core Specification", Release 4, version 4.1, April 2007.

[2] X. Li, and W. Zhang, "The design and implementation of home networks systems using OSGi compliant middleware" *IEEE Trans. on Consumer Electron.*, vol. 50, no. 2, pp. 528-534, May 2004..

[3] R. D. Redondo, A. F. Villas, M. Ramos, J. J. Pazos Arias, and M. Rey López, "Enhancing residential gateways: OSGi service composition" *IEEE Trans. on Consumer Electron.*, vol. 53, no. 1, pp. 87-95, Feb. 2007.

[4] K. Moon, Y. Lee, C. Lee, and Y. Son, "Design of a universal middleware bridge for device interoperability in heterogeneous home network middleware" *IEEE Trans. on Consumer Electron.*, vol. 51, no. 1, pp. 314-318, Feb. 2005.

[5] H. Lee, and S. J. Kim, "A standard method-based user-oriented integrated architecture for supporting interoperability among heterogeneous home network middleware", *International journal of smart home*, vol. 1, no. 1, Jan. 2007.

[6] J. Oh, J. Park, G. Jung, and S. Kang. "CORBA based core middleware architecture supporting seamless interoperability between standard home network middleware", *IEEE Trans. on Consumer Electron.*, vol. 49, no. 3, pp. 581-586, Aug. 2003.

[7] E. H. Binugroho, J. W. Choi, and Y. B. Seo. "Home network development based on CORBA middleware" *International conference on Instrumentation, Control and Information Technology*, pp. 186-191, Sept. 2007.

[8] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. "An architecture for differentiated services", The Internet Engineering Task Force documents, RFC 2475, Dec. 1998.

[9] UPnP Forum, "UPnP Device Architecture 1.0", Microsoft, June 2000.

[10] F. Moya, J. C. López. "SENDA: An alternative to OSGi for Large Scale Domotics", *In proc. of the joint international conference on wireless LANs and home networks and networking*, pp. 165-176, Aug. 2002.

[11] F. Moya, D. Villa, F. J. Villanueva, F. Rincón, J. Barba, and J. C. López. "Embedding Standard Distributed Object-Oriented Middlewares in Wireless Sensor Networks", *Journal on Wireless Communications and Mobile Computing*, vol. 9, Issue 3, pp. 335-345, John Wiley , 2009.

[12] J. Barba, F. Rincón, F. Moya, F. J. Villanueva, D. Villa, J. Dondo, and J. C. López. "OOCE: Object-Oriented Communication Engine for SoC Design." *In proc. of X EUROMICRO Conf. on Digital System Design* (DSD). 2007.

[13] M Henning and M. Spruiell. "Distributed Programming with ICE", ZeroC company, May 2008.

F. J. Villanueva received the Computer Eng. Diploma from the University of Castilla-La Mancha (UCLM) in 2001. In 2009 he obtained the PhD degree from the UCLM, where he is now working as Teaching Assistant. His research interests include wireless sensor networks, ambient intelligence and embedded systems.

D. Villa received the Computer Eng. Diploma from the University of Castilla-La Mancha (UCLM) in 2002. Since then he works as a Teaching Assistant at the UCLM. He is currently pursuing the PhD degree in Computer Science from UCLM. His current research interests include heterogeneous distributed systems, and distributed embedded system design.

F. Moya received his MS and PhD degrees in Telecommunication Engineering from the Technical University of Madrid (UPM), in 1996 and 2003 respectively. From 1999 he works as an Assistant Professor at UCLM. His current research interests include heterogeneous distributed systems, electronic design automation, and its applications to large-scale domotics and system-on-chip.

M. J. Santofimia received the degree of Technical Engineer in Computer Science in 2001 from the University of Córdoba (Spain); the Master's degree on Computer Security from the University of Glamorgan (Wales, UK) in 2003; and the degree of Engineer in Computer Science in 2006 from the University of Castilla-La Mancha (Spain). She is currently working towards her PhD as a member of the Computer Architecture and Networks Research Group (ARCO) at the University of Castilla-La Mancha. She is an assistant professor in the School of Computer Science in Ciudad Real (Spain)..

J. C. López (M' 94) received the MS and PhD degrees in Telecommunication (Electrical) Engineering from the

Technical University of Madrid (UPM) in 1985 and 1989, respectively. From Sep 1990 to Aug 1992, he was a Visiting Scientist in the Department of Electrical and Computer Engineering at Carnegie Mellon University, Pittsburgh, PA (USA). His research activities center on computer-aided design of integrated circuits and systems. His work is focused on algorithms for automatic synthesis, co-design and embedded computing. From 1989 to 1999, he has been an Associate Professor of the Department of Electrical Engineering at UPM. Currently, Dr. López is a Professor of Computer Architecture in the School of Computer Science at the University of Castilla-La Mancha.