Self-adapted Service Offering for Residential Environments

Juan M. González (1), José A. Lozano (1), Jorge E. López de Vergara (2), Víctor A. Villagrá (3)

(1) Telefónica Investigación y Desarrollo. 'Autonomic Communications Division'
(2) Universidad Autónoma de Madrid
(3) Universidad Politécnica de Madrid

Abstract-- One of the main challenges that the telecom industry is currently facing, is how to manage complex personal communication environments. These environments present an increasing number of varied devices such as computers, PDAs, webcams, Home gateway, sensors, etc. This implies a radical change from traditional telecommunication settings, composed of just a few types of standardized terminals. Indeed, Service Providers have no control over the devices that users install and deploy at home for service consumption. Furthermore customers have problems finding the services that fit well with their equipment from the myriad of services and Service Providers on the Internet.

This paper demonstrates an application and a proof of concept for Autonomic Communications architecture to solve this problem. Using a Home Gateway (HG), with the OSGi framework deployed, an Autonomic Element (AE) has been developed with the goal to find and personalize the service offer for a specific user. The AE is context aware, as it senses devices connected to the home network together with user preferences. ISPs only have to define the services and service profiles and the AE would be able to extract useful services and make a personalized advertisement for each customer. For the AE construction, an information model based on ontologies has been developed, as well as behavior rules to be applied to the ontology instances, so that an inference engine can reason with them.

Keywords: Context, Aware, Ontologies, OSGi, Services, Autonomic Communications.

I. INTRODUCTION

One of the main challenges that the telecom industry is currently facing, is how to deal with complex personal communication environments as HANs (Home Area Networks). These environments will provide a quantum leap in the communication and information services improving people's quality of life.

These environments present an increasing number of varied devices such as computers, PDAs, webcams, home gateways, sensors, etc. The complexity of managing such environments is a handicap for the development of the Information Society.

In the Internet services model, users are usually responsible for installation, configuration, and maintenance of the applications and devices that they own. These operations are seldom an easy task for non-expert users. Furthermore, normal users just want to 'enjoy' services, while 'others' manage their technical environment.

In addition to the complexity of managing such environments, customers also have problems finding services that fit well with their equipment amongst the myriad of services and Service Providers on the Internet.

Current initiatives such as Autonomic Communications (AC) can help users and Service Providers to cope with this complexity. AC design enables systems to know what is happening in their surroundings, as well as to adapt their behavior in an automatic and autonomous way. AC is a research area currently of interest, including a number of topics as is stated in [1]

In HAN networks, the Home Gateway (HG) becomes the link between service providers or the public domain and the user's personal domain. A number of applications offered by providers could be installed on them. These applications or bundles, as they will be later called, run in the gateway and are connected to other applications and devices. For this reason this device is a good candidate to become autonomic.

The gateway provides the necessary processing logic to achieve seamless application and services fulfillment. Additionally, the service provider should offer suitable services for each particular customer. This will, in turn, result in increased benefits and higher service consumption.

This paper demonstrates proof of concept on how to use an Autonomic Communication-based architecture, in order to achieve a highly scalable solution that will allow service providers to offer services to thousands of customers in the same way as if there were just a dozen.

On a Home Gateway (HG), using the OSGi (Open Service Gateway initiative [2]) as the framework installed on it, an Autonomous Element (AE) has been built. The AE's goal is to find and offer personalized and available services for specific users. The AE is able to automatically find and show the services that fit best for each user according to the devices installed in his HAN and the characteristics of his subscription. The Service providers' only responsibility is to define services and service profiles enabling the AE to extract useful services and to make a personalized advertisement for each customer.

For the AE construction, an information model based on ontologies has been developed, as well as behavioral rules applied to the ontology instances, so that an inference engine can reason with them.

This paper is structured as follows: The next section briefly

presents the related technologies contained herein; the OSGi framework and autonomic systems. A scenario is then presented illustrating the usage of the developed system. Continuing, the architecture of the AE is analyzed, then, the defined ontology is given, as well as the rules used to calculate the suitable services for a user. Finally, some conclusions are given.

II. State of the Art

As stated above, the proof of concept has been developed around the OSGi framework as the most representative technology over Home Gateways. The goal is to make home gateways behave as an Autonomic System. This section gives a short overview of both.

A. The OSGi framework

Following [2], the OSGi specifications are defined as a standardized component-oriented computing environment for networked services, which is the foundation of enhanced service-oriented architecture. This framework adds to a networked device, the capability to manage the lifecycle of the software components in the device, from anywhere in the network. Software components can be installed, updated, or removed on the fly without ever having to disrupt the operation of the device.

The reason why OSGi is so widely accepted in the industry is that it forms a small layer that allows multiple JAVA™ based components to cooperate efficiently in a single Java Virtual Machine (JVM).

The OSGi framework is divided into layers depending on the functionality of the services it provides to its upper layer and applications.

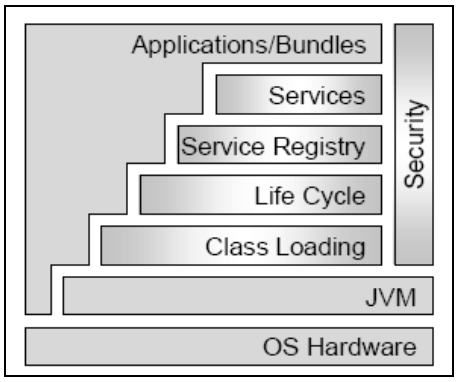

Figure 1: OSGi Architecture

On the OSGi framework, applications are called bundles. A bundle is the specific implementation of an application that runs on the OSGi Framework.

B. Autonomic systems

In this paper the Autonomic System concept is understood as stated in [3]. It is defined as 'a system that operates and serves its purpose by managing itself without external intervention even in the case of environmental changes.' Figure 2 presents a description of an autonomic system where one fundamental block of the AS is its capability to observe the external operational context, represented by S1 to Sn sensing inputs. Another inherent block of an AS is the goal or purpose it serves, but also the know-how to achieve these objectives. Logic is the block responsible for making decisions to serve the system's purpose, but taking into account observations of context.

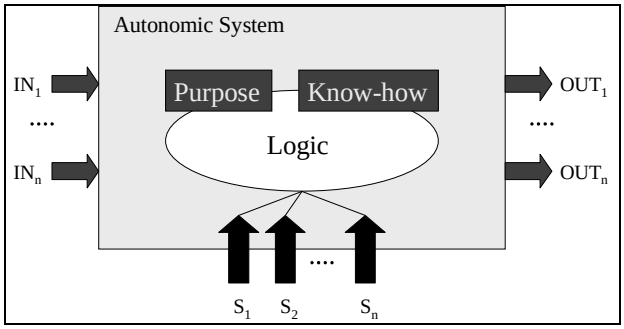

Figure 2. Autonomic System Description

Following this definition of Autonomic System it is important to clarify those characteristics that make a system behave as an autonomic system. These properties are:

- **Automatic:** The system must be able to self-control its internal functions and operations.
- **Adaptive:** An autonomic system must be able to change its operation or behavior (i.e. its configuration, state and functions).
- **Aware:** An autonomic system must be able to monitor (sense) its operational context, as well as its internal state, to be able to assess if its current operation serves its purpose.

Any system that presents these properties would be classified as an Autonomic System.

III. Scenario

The testing environment for the proof of concept described in this paper, consists of a home area network (HAN), with a home gateway (HG) acting as the central core and based on the OSGi standard. Service delivery lies in the installation of bundles in the HG. This HG then distributes services to the correct devices. Figure 3 briefly describes this scenario.

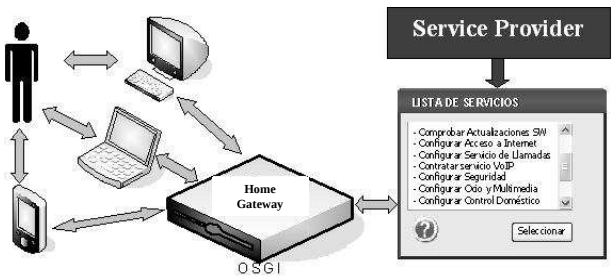

Figure 3: Scenario

Without having a centralized system checking the technical

environment, the proof of concept shows how to transform the HG into an intelligent device capable of knowing which is the perfect offer for a customer in an 'autonomic way' from users' context information.

In order to transform the HG in an autonomic system, a bundle implementing an autonomic element has been developed. This management agent will use the services offered by the OSGi framework to detect any possible changes in the customer's environment and also to act in order to customize the product offers for the customer.

Each of the levels in the OSGi framework (see figure 1) offers services that will be used by the agent to become aware of the customer's environment. For that purpose the agent has to know at least:

- The bundles installed on the platform
- The devices inside the network

Furthermore, in order to reason with the information that has been captured, behavior modeling is another variable that the agent must take into account. To this end a conceptual model has been implemented through the use of ontologies for modeling the information, and ontology-based languages for modeling the behavior. These languages have been used to describe the user's domain, that is, to describe customer's profile and the services and devices needed for service delivery. The agent is able to link this knowledge to the devices inside the network and the commercial offer to infer which services are suitable in each situation.

The main actor of the execution scenario is the management agent, based on the autonomic element architecture. It is able to become aware of the customer's environment, and that will indicate to the service provider how to personalize the offer.

It is placed in the customer's HG acting as the mediator between the user and the service provider. The agent is able to relate intelligently the user's context information, which it obtains from its communication with the OSGi framework, and the information shared with the service provider, contained in the Ontology repository.

The execution sequence is described below in three steps:

1. Firstly the agent obtains information about the customers environment from the OSGi framework that can be classified in:
   - Information related to bundles installed on the platform and implemented services, using OSGi services from the interface 'public interface BundleContext' and analyzing the bundles Manifest.
   - Platform characteristics, configuration parameters and OSGi version.
   - Information related to client's devices, using services provided by the OSGi framework uPnP.

   This information is obtained using asynchronous or synchronous procedures, depending on the agent's information needs.
   - Synchronous environment detection: the agent feeds itself periodically with information related to the user's gateway, the bundles list and the implemented services. Changes in this data do not require a rapid update in the agent, so they can wait for the next state request from it.
   - Asynchronous environment detection: in this case it is not the agent that carries out the request but the gateway that informs the agent of the disconnection or connection of a new device. This will release the appropriate actions. It would be necessary to install a bundle listener that will gather the events of interest related to the customer environment: services installations and uninstallations, the appearance of new devices, etc.

2. Secondly the agent comes in contact with the Ontology Repository and carries out the following two actions:
   - Downloads the ontology. It includes the domain definitions as well as rules that can be explicitly or implicitly defined in the ontology so as to model the agent behaviour In accordance with the policies and rules defined by the service provider, the agent decides which services or types of services can be offered to the customer.
   - Updates the customers' instances in the repository with the information related to the environment, based upon the information obtained from the OSGi framework.

3. Finally, the agent informs the customer directly sending a message with a URL to their personalized services offer. This way, the service provider can offer users the services that the management agent has selected as suitable for them. This is possible because it shares the knowledge described by the ontology.

Figures 4 and 5 present the two types of interactions (asynchronous and synchronous) from the proof of concept:

- **Asynchronous**: The user plugs a device into the HAN and the agent automatically personalizes an offer based upon the customer profile that will definitely be of interest to them.

- **Synchronous**: The agent updates the information knowledge about the customers' environment and the appropriate offer for them. When the user connects with the service provider it will show them a personalized offer.

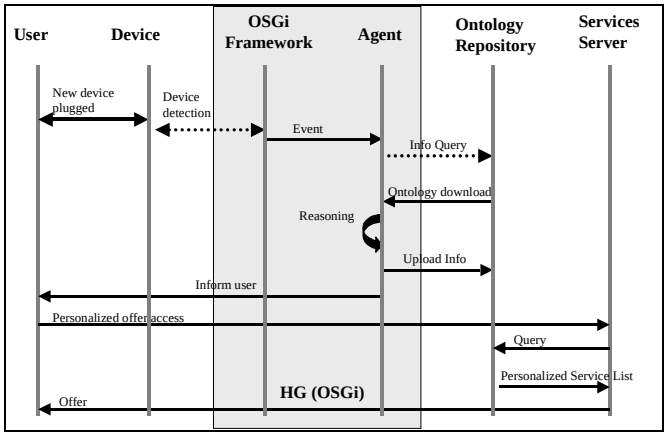

Figure 4: Synchronous actions sequence

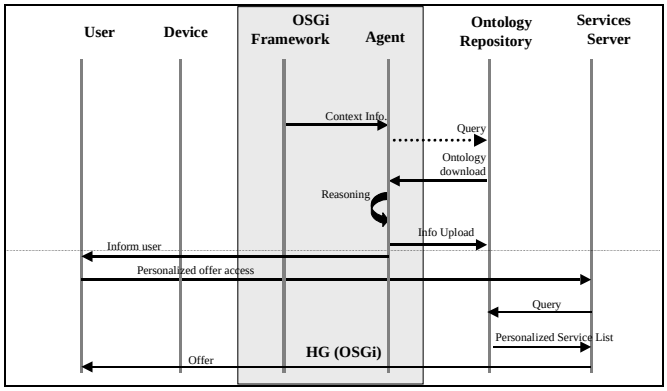

Figure 5: Asynchronous actions sequence

Testing these interactions will validate the use of ontologies to add a new semantic level, in which we can interpret requirement information and customer environment requirements, in order to personalize their services offer.

A specific example of a situation that can be solved with this type of technology will be:

1. The user connects a webcam to his network.
2. The agent detects this event and uses the knowledge described by the ontology to search for services which make use of the webcam to operate (video-vigilance, videoconference, etc.) and that are also appealing to the customer due to their profile.
3. A message will inform the customer about new services that could be enjoyed with a webcam. This message might include a link to detailed information and subscription options.

Another example would be when the service provider modifies its offer with new products or services, meaning that suitable products are offered to the customers, according to their profiles.

IV. AUTONOMIC ELEMENT ARCHITECTURE

As stated before, in order to achieve the business objectives concerning service offer customization, home gateways should evolve towards autonomic systems. Customer devices are usually connected to a home area network (HANs) and all these HANs have one point in common, that is the home gateway (HG). The proposed architecture is designed to distribute intelligence over these HGs in order to make them behave as autonomic systems. These intelligent HGs must be able to behave according to operators' rules and policies.

The global architecture proposed is depicted in figure 6.

In order to get the HG running as an autonomic system, a management agent has been developed. It has been designed following the autonomic element architecture depicted in figure 7 and described by IBM in [10].

Such an autonomic element presents five function modules:

- Knowledge
- Monitor
- Analyze
- Plan
- Execute

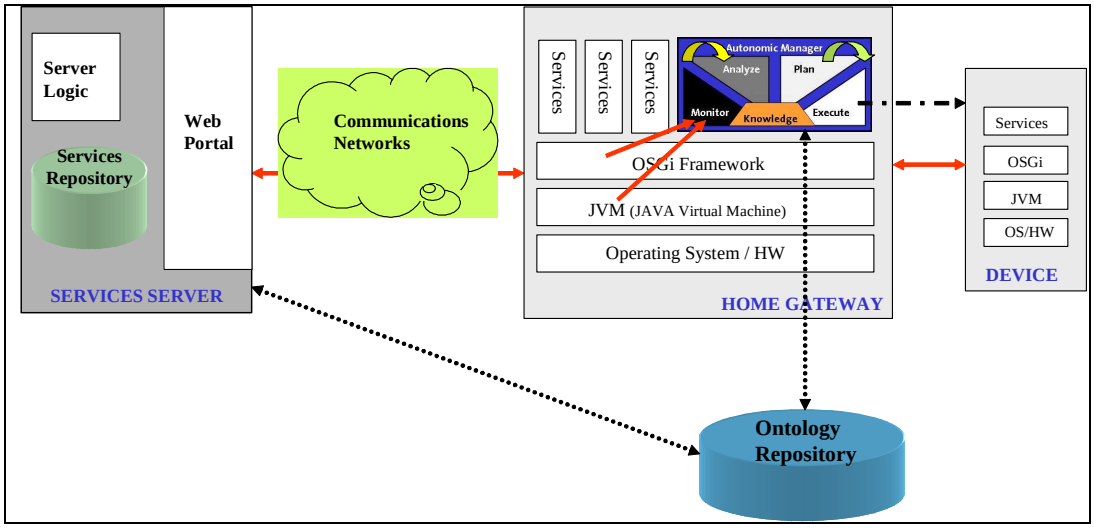

Figure 6: Global Architecture

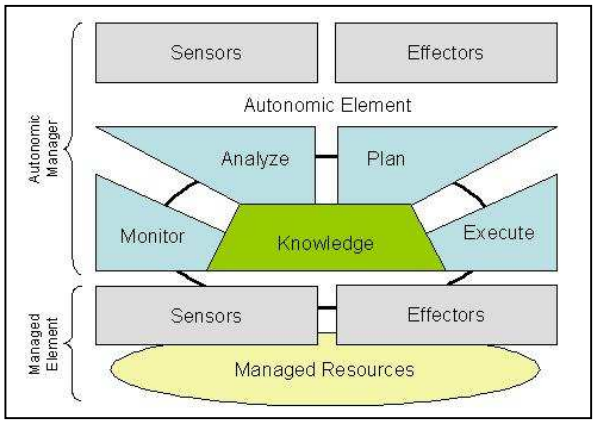

Figure 7: Autonomic Element (source: IBM)

A. Knowledge

Knowledge (see figure 6) is the necessary description of the domain that the agent requires. Technologically it is implemented through a conceptual model; this is an ontology. The element manager downloads this knowledge from an ontology repository and acts, supporting all functional modules that constitute the Autonomic Element.

The ontology is used to make a conceptual description of the whole user context, what a service is, which relation it has with devices and technical platforms, dependencies with bundles, etc. Figure 8 shows some entities defined within the ontology. This ontology is later presented in section V.

B. Monitor

This functional module (see figure 6) will capture information from the HG about the technical characteristics of the Home Gateway, like the version of the operating system and devices running on the home network. Through the knowledge module it will also collect information about non-technical parameters, such as the commercial segment, the kind of subscription, customer profile, etc.

C. Analyze

This functional module (see figure 6) is responsible for the reasoning function. The inputs to the system are then analyzed according to the given rules. These rules are generated by service providers from high-level policies. And they are defined to control the global behavior of the community of Intelligent HGs, in order to have thousands or even millions of agents composing commercial offers within business requirements imposed

This inference engine loads the set of instances about HG context (User Profile, Devices, Platform, and installed Bundles and Services) and available Services. Combining this information with the set of defined rules, it obtains which services fit the user context.

D. Plan

Once there is an inference output, the agent decides actions (see figure 6). In this case it takes the output of the inference engine and prepares a web with the list of services needed. The output of the inference engine will contain a list of Service instances that are to be displayed to the user. For this, the Service Server uses the information of each service, contained in the ontology.

The implementation of the proof of concept does not focus greatly on this planning task; it simply takes information about the recommended services obtained through inference in the analyzing phase. However, it is a very important issue to consider in future work.

E. Execute

This functional module (see figure 6) executes actions over the resources. So it starts all the actions oriented to inform customer about the selected commercial offer

All these modules work together and create a behavior that converts the HG in an automatic system. Following the definition stated before, this architecture provides the following characteristics to the HG:

Automatic: If the user's context changes, because they have plugged a new device, made changes to the subscription with the provider, or even updated software versions; whatever the situation, the HG will generate a new offer. Additionally, if the service provider changes the environment by adding new services to the catalogue, or if it generates new commercial rules for a specific market segment, Home Gateways will adapt the customer offer without any external action.

Adaptive: HGs are highly adaptive. They are able to automatically adapt their behavior (in this case the way they "think") when composing commercial offers. The behavior of the HG depends on the policies that service providers establish to accomplish their business.

Aware: The HG not only senses all needed parameters in the context, it is also aware of it in relation with the business goals of the service provider.

F. Proof of Concept Implementation

The logic of the AE has been implemented in an OSGI bundle that is responsible for obtaining the installed devices and bundles using the OSGi API, loading the knowledge in the inference engine and reading the results and acting accordingly. Results are published in the web server included in the OSGi framework. In order to notify that new services are available, a message with a link to the result web pages is sent to the customer.

The following software has been used for the described implementation:

- **Protégé:** This ontology editor has been used in the definition of the ontology and the behavior rules. The ontology has been defined in OWL, and rules have been defined in SWRL. The use of OWL is important, to allow a distribution of the ontology knowledge.
- **Knopflerfish**: This is the OSGi platform implementation used in the HG. The necessary logic to behave as an AE is contained in a bundle to be installed in this platform.

- **Jena**: This Java library has been used to pass on the ontology information.
- **Bossam**: This Java inference engine has been used in the AE to load onto the ontology, existing instances and rules to infer which services can be advertised to the user.
- **Apache Tomcat**: This is used for the implementation of the service provider server, to present the user with the set of advised services. For this, the server accesses the inferred knowledge, published in the OSGi platform.

V. Conceptual Model. Ontologies and Behaviour Rules

As stated in previous sections, an ontology with a set of classes or concepts has been defined based on the described scenario. This ontology is depicted in figure 8.

Ontology classes, as well as their related properties, are necessary to specify the domain of the system. Properties can be both data type properties, useful to contain information (e.g. the name of a service); or object properties, useful to associate instances of different classes (e.g. the services contracted by a customer). Property values can be inferred from existing knowledge by using a set of rules, as explained later. The set of classes, depicted in Figure 8, includes the following:

- **Service**: this class models the services provided by the service provider. It contains properties such as the price, complexity, customer segment, type of device needed and bundle dependency. These properties are useful to infer if a service should be recommended to a customer.
- **User profile**: this class defines the profile of a customer that uses the service provider services with the HG. It contains properties such as the type of customer; the maximum service price and the complexity that can be assumed by the user; identifiers assigned by the network operator, and a set of properties that link this user with the service provider services. It is possible to infer what services are already contracted, what services are available to the user given its profile and hardware used, and services that can be advertised to a user given a change in the HG context.
- **Bundle**: this class models the OSGi software packages that can be installed in the HG. Most of the properties of this class can be obtained from the Manifest file of the bundle (e.g. imported and exported packages). Based on these it is possible to infer bundle dependencies.
- **OSGi platform**: this class models the HG. Its properties are related to its technical characteristics, as well as the set of installed bundles and devices, and users of the platform.
- **Device**: this class models the devices of the home network. Its properties are related to the information that can be obtained through the OSGi services. Two properties are important to infer knowledge: the type of device, and the associated services that can be provided thanks to that device.

The classes and properties of this ontology have been defined in OWL (Web Ontology Language) [4]. This ontologies definition language will allow us to take advantage of the set of tools developed by the Semantic Web community (e.g. Protégé [5], Jena [6]). The ontology is expected to include an instance of each service provided by the service provider and their relationship with bundles, device types they need, and customer segment to which they are directed. At the same time, the HG adds to the ontology instances information about the set of bundles and devices installed on it, which represent its context.

To perform the reasoning in the HG, the following set of logical rules have also been defined. These rules have been specified in SWRL (Semantic Web Rule Language) [7], due to its tight integration with OWL. Once again, Semantic Web community tools (e.g. SWRLJess [8], Bossam [9]) have provided valuable assistance with executing the rules, acting as inference engines. The rules have been defined in generic terms, to avoid defining a rule for each user or each device.

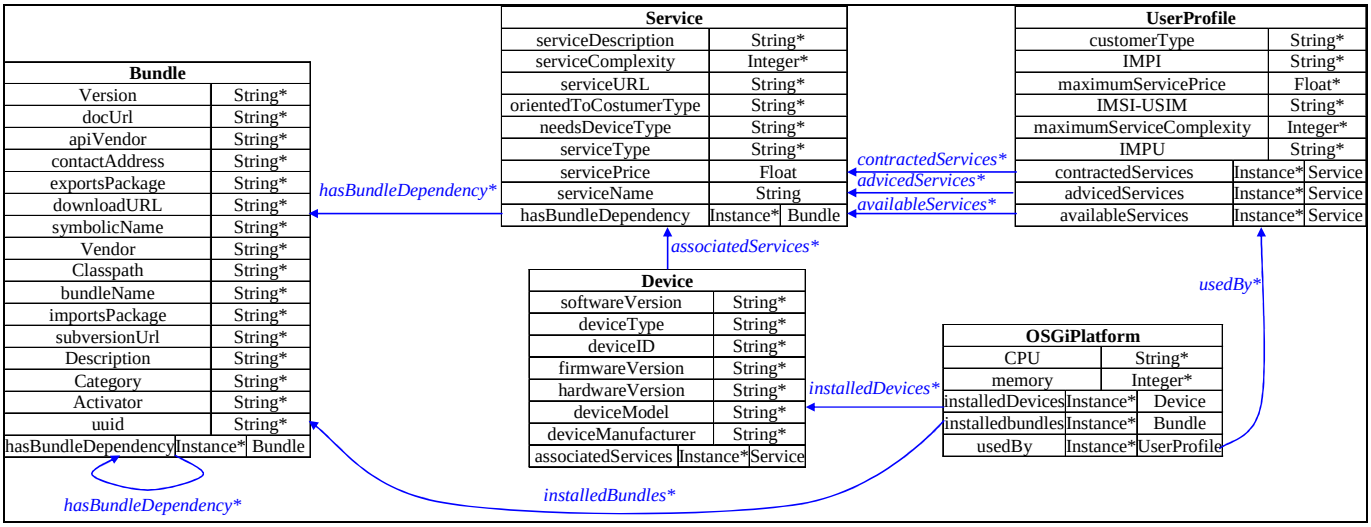

Figure 8: Conceptual Model.-Ontology

This is also possible if the operator's policies need to be fulfilled. These rules have the aim of obtaining the services that are going to be recommended to users. Other rules should be defined if the HG has to perform other tasks (e.g. autoinstalling bundles).

Rule 1: It obtains the set of bundles necessary for another bundle. This rule compares the importsPackage and exportsPackage of bundle instances to infer if there is a dependency between two bundles:

Bundle(?importer) ∧ *importsPackage(?importer,?importedPackage)* ∧ *Bundle(?exporter)* ∧ *exportsPackage(?exporter,?exportedPackage)* ∧ *swrlb:equal(?importedPackage, ?exportedPackage)* → *hasBundleDependency(?importer, ?exporter)*

Rule 2: It obtains the set of services that have been contracted by a customer. It depends on the installed bundles in the HG that are associated with a service of the service provider. Then, if a bundle is found that is associated with a service, and it is installed in the HG of a user, then this user has contracted that service:

Service(?availableService) ∧ *Bundle(?availableBundle)* ∧ *hasBundleDependency(?availableService, ?availableBundle)* ∧ *bundleName(?availableBundle, ?availableBundleName)* ∧ *OSGiPlatform(?gateway)* ∧ *installedBundles(?gateway, ?installedBundle)* ∧ *bundleName(?installedBundle, ?installedBundleName)* ∧ *UserProfile(?user)* ∧ *usedBy(?gateway, ?user)* ∧ *swrlb:equal(?availableBundleName, ?installedBundleName)* → *contractedServices(?user, ?availableService)*

Rule 3: It obtains the services that are associated with a device. The device is detected to have been installed in the gateway. In this case, if a type of device is found that is needed by a service, then that device can be associated to this service.

Service(?availableService) ∧ *needsDeviceType(?availableService, ?neededDeviceType)* ∧ *Device(?installedDevice)* ∧ *deviceType(?installedDevice, ?deviceTypeValue)* ∧ *swrlb:equal(?deviceTypeValue, ?neededDeviceType)* → *associatedServices(?installedDevice, ?availableService)*

Rule 4: It obtains the services that can be offered to customers of a certain type. This rule can be amplified by including properties related to complexity or price. In this case, a service is available to a user if that service is oriented towards the same market segment to which the user belongs.

Service(?availableService) ∧ *UserProfile(?user)* ∧ *orientedToCustomerType(?availableService, ?userType)* ∧ *customerType(?user, ?userTypeValue)* ∧ *swrlb:equal(?userType, ?userTypeValue)* → *availableServices(?user, ?availableService)*

Rule 5: It obtains the services that can be recommended to a customer, given the devices that are attached to the HG. Prior rules are assumed. This rule compares the services that are available to a user to those that can be used with a particular device, installed on the user platform.

UserProfile(?user) ∧ *availableServices(?user, ?service)* ∧ *Device(?device)* ∧ *OSGiPlatform(?gateway)* ∧ *usedBy(?gateway, ?user)* ∧ *installedDevices(?gateway, ?device)* ∧ *associatedServices(?device, ?service)* ∧ ¬ *contractedServices(?device,?service)* → *advicedServices(?user, ?service)*

This last rule cannot be completely expressed in SWRL, because SWRL does not have the NOT (¬) operator. However, it would be possible to obtain all services that are related to a customer type and device type.

VI. CONCLUSIONS

This paper has presented a proof of concept to show that Autonomic Systems are a suitable technology that will allow operators to face the management complexity of future service and networks infrastructures. The application of autonomic technologies in order to manage complexity must result in an increase on the ease of use for users. In the proof of concept presented, when a customer buys and installs a new device in his home network, the Autonomic Element is able to offer services suitable for such a new device. The same occurs when the customer subscription is updated or changed. In this way, customers will not be worried at all about which services are suitable for them, because all services offered will fit their requirements. It is also necessary to remark that not only the technical aspects are considered; commercial information and customers' personal preferences are also taken into account.

On the other hand, from a service provider point of view, autonomic technologies imply a great improvement. Customer offers are personalized by simply updating information about services in the shared knowledge. That means that it does not matter if there are hundreds, thousands or millions of users, the solution is absolutely scalable. That is one of the main challenges when facing customer management.

The implementation described in this paper has included both the definition of an ontology to maintain the knowledge needed by the system, and the development of a management agent that is loaded in an OSGi platform. All scenarios have been tested satisfactorily, obtaining in each case those services that were suitable to the real context of the user. Context was based on the user profile, OSGi platform characteristics and installed devices. It has been verified that services that didn't fit this context were not offered to customers. Regarding service provider's point of view, this decentralized approach scales better than centralized ones, where service providers must monitor available resources in each HAN and offer services after processing all the information.

As a result, we have achieved the goal of making potential functionality of AS powerful and in the interest of operators to

manage complex infrastructures. But other aspects will be study in future work. For example it is necessary to check the performance of AS. The case presented has shown that inference engines require quite a lot of memory and CPU, and it is necessary to build them into very limited resources such as Home Gateways. Performance is a critical aspect that will affect the deployment cost of these technologies. It is necessary to improve hardware of current gateways to support inferences. Additionally, inferences engines' memory use and processing requirements must be optimized.

Another important aspect is that of knowledge management, that is to ask: how should ontologies be supported? Management architecture for knowledge is essential to have AS working properly. Ontologies must be defined and updated, they are distributed and at the same time shared among all entities. This scheme does not seem to be easy to tackle given the current state of technology.

Other future work is related to the definition of policies and rules, and how they should be included implicitly in the ontology or explicitly as rule instances.

Performance, operation, maintenance and standardization aspects of autonomic systems must be considered and resolved before introducing these technologies in live environments. However, we see Autonomic Systems as the key to overcome current management state of technology for the forthcoming communication environments.

These technologies must evolve to a more mature state, not yet achieved currently. This proof of concept has been a first step to get an insight about how these technologies must evolve.

VII. References

[1] Michael Smirnov, "Autonomic Communication, Research Agenda for a New Communication Paradigm" Autonomic Communication, White Paper, November 2004

[2] OSGi Alliance, "About the OSGi Service Platform" Technical Whitepaper Revision 4.1, 11th November 2005

[3] S. Schmid, M. Sifalakis, and D.Hutchison, "Towards autonomic Networks" In proceedings of 3rd Annual Conference on Autonomic Networking, Autonomic Communication Workshop (IFIP AN/WAC), Paris, France, September 25-29, 2006.

[4] D. L. McGuinness, F. van Harmelen, "OWL Web Ontology Language Overview," W3C Recommendation, 10 February 2004.

[5] H. Knublauch, R. W. Fergerson, N. F. Noy, & M. A. Musen, "The Protégé OWL Plugin: An Open Development Environment for Semantic Web Applications," Third International Semantic Web Conference, Hiroshima, Japan, 2004.

[6] B. McBride, "Jena: A Semantic Web Toolkit," IEEE Internet Computing , vol. 06, no. 6, pp. 55-59, November/December, 2002.

[7] I. Horrocks, P. F. Patel-Schneider, H. Boley, S, Tabet, B. Grosof, M. Dean, "SWRL: A Semantic Web Rule Language Combining OWL and RuleML," W3C Member Submission, 21 May 2004

[8] M. J. O'Connor, H. Knublauch, S. W. Tu, B. Grossof, M. Dean, W. E. Grosso, M. A. Musen, "Supporting Rule System Interoperability on the Semantic Web with SWRL," Fourth International Semantic Web Conference (ISWC2005), Galway, Ireland, 2005.

[9] Minsu Jang, Joo-Chan Sohn, "Bossam: An Extended Rule Engine for OWL Inferencing," Lecture Notes in Computer Science, Volume 3323/2004, Springer Verlag, pp. 128-138

[10] Jeffrey O. Kephart, David M. Chess, "The Vision of Autonomic Computing". IEEE Computer Society, January 2003. IBM Thomas J. Watson Research Center

[11] H. Knublauch, R. W. Fergerson, N. F. Noy, & M. A. Musen, "The Protégé OWL Plugin: An Open Development Environment for Semantic Web Applications," Third International Semantic Web Conference, Hiroshima, Japan, 2004.

[12] Knopflerfish OSGi, http://www.knopflerfish.org/

[13] Apache Felix, http://cwiki.apache.org/FELIX/

[14] Apache Tomcat, http://tomcat.apache.org/