Design and Implementation of a Framework for Self-Configuring Devices Using TR-069

Houda Rachidi

Thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
In partial fulfillment of the requirements
For the degree of Master of Science in Computer Science

School of Information Technology and Engineering
Faculty of Engineering
University of Ottawa
Ottawa, Ontario

Table of Contents

List of Figures

List of Tables

Abstract

Communication network technologies have been evolving exponentially in the late decades. These innovations increase the network capabilities and open new horizons to creating novel and original services. The heterogeneity in equipment qualifications increases the level of complexity in the technological advancement. In such environment, service management has become an everyday challenge to service providers. Important efforts have been deployed to innovate in the exploitation of intelligent devices in the home and other private locations.

In this Thesis, we propose a framework for self-configuration of devices within Hone Area Networks. We propose a device self-configuration architecture based on IBM Monitor-Analyze-Plan-Execute using Knowledge autonomic control loop. To prove the validity of our system architecture and support its applicability, we developed a prototype system that gives a general control loop implementation for device self-configuration using the CPE WAN Management Protocol. A video streaming scenario is implemented and used to evaluate validity our framework.

Acknowledgment

It is my pleasure to convey all my gratitude and humble acknowledgment to all the wonderful people that made part of my life during these two years, both at the academic level and at the personal level.

In the first place, I would like to thank Professor Ahmed Karmouch for his supervision throughout this research and thesis work. His guidance, advices, and continuous support have enlightened the way to achieving all my goals. His constant encouragement and patience helped me give the best I could during the hard times of research. Professor Karmouch's commitment to hard work and passion for continuous seeking of knowledge set a great example for all students eager to develop their scientific and research skills.

I would like also to thank Stenio Fernendes, Imad Abdeljaouad, Yousif Al Ridhawi, Ismaeel Al Ridhawi, and all my colleagues at the Intelligence for Mobile Autonomic and Cognitive Networks Laboratory for their cooperation and support during my research. I really enjoyed the fruitful discussions during and outside the lab meetings.

My deepest thanks and love go to my parents, the two most wonderful persons in my life. Their endless love and patience constitute my real source of energy. Without them believing in me, I would have never been able to overcome all the obstacles encountered during my research journey. To you, dear parents, I dedicate this thesis.

I am very grateful to all my great friends and my two amazing brothers who have been always there for me, cheering me up, and encouraging me. The list of friends to thank is too long. I would like to express special thanks to Fatima Azzahra El Azzouzi, for her daily presence, despite of all the constraints. Thank you dear friends, you have been a great source of inspiration to me.

Acronyms

AC	Autonomic Manager
ACI	Autonomic Computing Initiative
ACS	Auto-Configuration Server
ADsM	Advanced Device self Managing
CC/PP	Composite Capabilities / Preferences Profiles
CHOP	Configuration, Healing, Optimization, and Protection
COMANTO	COntext MAnagement oNTOlogy
CONON	OWL Encoded Context Ontology
CPE	Customer Premises Equipment
CWMP	CPE WAN Management Protocol
DHCP	Dynamic Host Configuration Protocol
DM	Device Management
DMS	Device Management Server
DSL	Digital Subscriber Line
EHNRMP	Extended Home Network Remote Management Platform
ER	Entity Relationship
FOCALE	Foundation Observation Comparison Action Learn rEason
GENA	General Event Notification Architecture
HAN	Home Area Network
HTTP	HyperText Transfer Protocol
HTTPMU	HyperText Transfer Protocol Multicast
HTTPU	HyperText Transfer Protocol Unicast
IGMP	Internet Group Management Protocol
IP	Internet Protocol
IPTV	Internet Protocol Television
LAN	Local Area Network
MAPE-K	Monitor, Analyze, Plan, and Execute using Knowledge
OMA	Open Mobile alliance
OUI	Organization Unique Identifier

OWL	Web Ontology Language
QoS	Quality of Service
RGW	Residential Gateways
RPC	Remote Procedural Call
RTSP	Real Time Streaming Protocol
SLA	Service-Level-Agreement
SOAP	Object Access Protocol
SSDP	Simple Service Discovery Protocol
SSL/TLS	Secure Socket Layer/Transport Layer Security
STB	Set-Top-Boxes
SyncML	Synchronization Markup Language
TR	Technical Report
UPnP	Universal Plug and Play
WAN	Wide Area Network

Chapter 1

1. Introduction

1.1. Motivation

In residential premises, customers nowadays use a new generation of devices and services, which incorporate an ever-growing set of technologies. This rapid growth makes the management of devices and services an arduous endeavor. Such technological challenges should be transparent to companies. Rather, the advancements should be a tool for them to enlarge the scope of their area of specialty and introduce new services that will help keep their customers satisfied.

Service providers are more and more concerned about enhancing the user's entertainment experience, and meet their expectations for quality. Furthermore, customer support service might be problematic, as it takes considerable time when no proper management tools are available to ease the task for support representatives. Besides, in front of complex systems, technical support bodies might face difficulties remedying to problems within limited time constraints. They are limited in terms of competences and cognitive thoughts, and end up facing all the management overhead. The customer service processes serve mainly as a "reactive remedy" and are usually too late. This inconvenience of manual or human intervention in configuring, troubleshooting, and policy making results in errors and lack of efficiency, and above all, it is time consuming.

From an end-user perspective, the management complexity should definitely be kept away, as much as possible. System complexity might be a factor that determines whether a service grows or vanishes. Devices or services that are complex to use or manage cause user's dissatisfaction and hence, they are easily abandoned. The end-user does not have to worry, at any time, about neither the operation of devices, nor the availability and well-functioning of the service. Moreover, users are becoming more demanding in terms of perceived quality. They are opting for systems that will make their user experience enjoyable. A customization

of the services according to user needs is also highly desirable.

In particular, communication services have never been in such increasing demand as they are currently, and they are associated with high complexity. This complexity comes from various factors as pointed out in [1]:

- Heterogeneity in devices
- Abundant services and applications
- Unpredictable traffic pattern and dynamics

As communication networks evolve both technologically and in size, complexity increases. Isolations between different communication systems are diminishing due to the worldwide spread mobile technologies. Other factors are related to the integration, composition, and inter-networking of all types of networks (wired and wireless). In addition to all these networking issues, high Quality of Service (QoS), security, and scalability has become vital.

Conscious of all these concerns, the network and tech community encourages research towards communication systems and devices to go autonomic, and manage their functions and performance with "efficiency", "resilience", "immunity", and "evolvability" with a restricted human interference. "What is really sought-after is mature self-management that features proactivity rather than reaction, formulates knowledge instead of simple enforcement, and performs situated reasoning as opposed to throwing question marks at the humans." [1]

These issues can be addressed with self-management, where system operation and maintenance are hidden. Self-management is defined as the process by which computer systems shall manage their operations and computations, ideally with no human intervention. This way, the machine does the management incessantly and the technical pressure is eliminated for both system administrators and end-users.

The core idea of autonomic computing systems is based on self-management. Automating management platforms for devices and services will significantly reduce the amount of management tasks to be done by administrators on the one hand, and will make it seamless and not cumbersome for users on the other hand. Hence, special efforts are directed towards

improving the service provisioning. Self-management introduces four functional areas: self-configuration, self-healing, self-optimization, and self-protection. Self-configuration can be considered as an important pillar in achieving the desired monitoring, diagnosis, and healing.

1.2. Definition of Concepts

The concept of "autonomic" was drawn from nature. Specifically, the analogy is based on the human/animal autonomic central nervous system which regulates different body parts and adjusts them following different situations automatically without any external intervention. The term "*autonomic computing*" [2] [3] was first used by IBM to describe a vision of computing systems that self-manage their behaviors in accordance with high-level administrative objectives, directed by humans, in dealing with the crisis of ever-increasing complexity in the management of large-scale distributed and heterogeneous information technology infrastructures.

Self-management refers to the four 'self-* properties': self-configuration, self-optimization, self-protection, and self-healing. These properties are also referred to as the CHOP (Configuration, Healing, Optimization, and Protection) attributes (See Figure 1-1). The objective behind adopting these properties is having a total autonomous system able to adapt its functionalities to any change in the surrounding environment, taking into account its functional behavior and purpose. This results in an adaptive, optimal, secure and resilient system.

*Self-*Properties:*

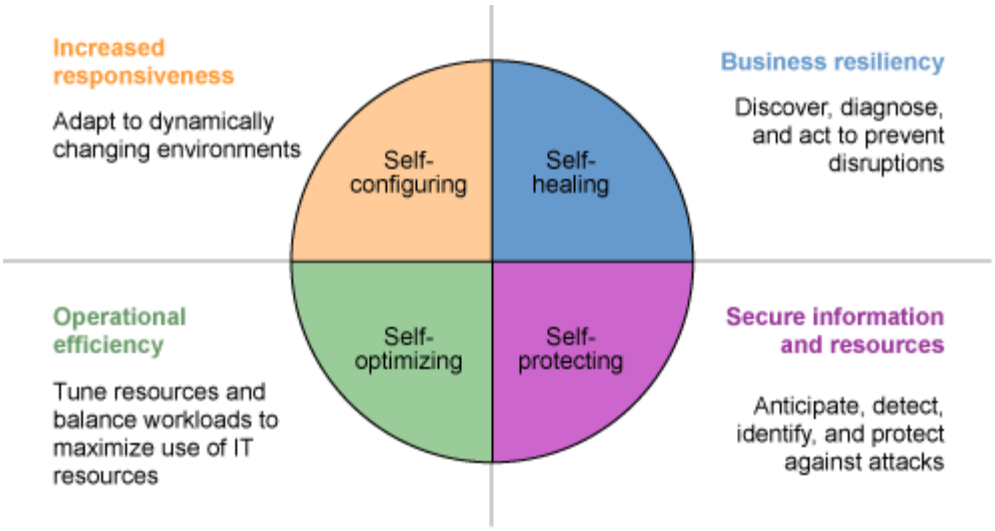

Figure 1-1: Autonomic Computing Self-* properties [5]

Self-configuration

Self-configuration refers to the system's ability to dynamically configure itself. The system can adapt instantaneously, with minimal human intervention, according to changes happening in the environment. Such changes can concern the addition or removal of system components, or noticeable changes in the system's characteristics. "Dynamic adaptation helps ensure continuous strength and productivity of the IT infrastructure, resulting in business growth and flexibility." [5]

Self-healing

The self-healing property is concerned with detecting problematic operations (that might be proactively using predictions or other approaches), diagnose the problem, and then start corrective action without disrupting system applications. These corrective actions represent a remedial response that alters the state of the system itself, or influences changes in other elements of the environment.

Self-optimization

Self-optimization property reflects the system's ability to tune resources to efficiently maximize their allocation and utilization to meet both end-users' needs and business objectives, with minimal intervention [2]. The resource tuning might regard reallocation of

resources (as in the case of a workload change) to improve overall utilization, or ensuring that given business transactions can be completed within a favorable time.

Self-protection

Self-protection property of a system helps anticipating, detecting, identifying, and protecting against possible threats. The threats can be in the form of hostile behavior of the system. Such behavior can be an unauthorized system access and use, virus attack and propagation, denial-of-service attacks, and other system failures. Corrective actions need to be made as such behavior occurs, to reduce vulnerability.

Control Loop:

An interesting approach within the IBM Autonomic Computing Initiative (ACI) is the MAPE-K (Monitor, Analyze, Plan, and Execute using Knowledge) control loop [4]. Figure 1-2 illustrates the control loop in a structural arrangement of the components, not a control flow, because one component might call any other component as needed.

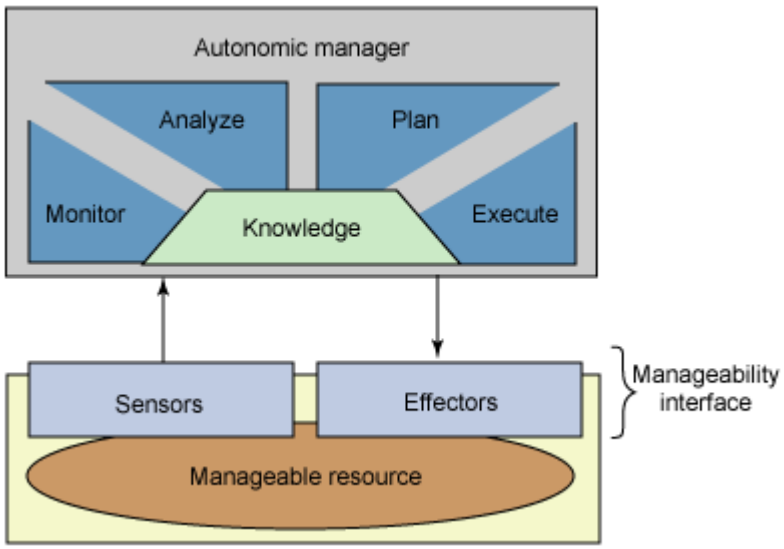

Figure 1-2: IBM MAPE-K Control Loop [6]

In this approach the state of a system is monitored through sensor operations at the level of the managed resources, then it is analyzed, and appropriate actions are planned and sent to execution, at the level of the autonomic manager, through effectors' operations. The overall autonomic process is based on pre-defined, and/or learned knowledge about the managed

resources. In the following we give a brief description of the MAPE component as it has been defined in [2]:

Monitoring: offers the means that collect, aggregate, filter, manage, and account for system information details (metrics and parameters) collected from a managed resource.

Analysis: provides the mechanisms which associate and model complex system situations. This way, the autonomic manager learns about the system environment and helps predict future situations.

Planning: presents the methods that structure the corrective action necessary to attain goals and objectives. This component might make use of policy information to direct its process.

Execution: is the component responsible of controlling the execution of an action generated by the planning, with considerations for on-the-fly updates.

1.3. Thesis Objectives

The goal of this thesis is to build a system that enables devices to self-manage their components and operations. Support for device self-configuration in the system must meet the following objectives:

Control loop: The architecture for device self-configuration must be open to all types of devices. We consider a device to be a physical unit that has computing capabilities, and offers a given function or service. Hence, we consider not only the configuration of the device itself, but also service components. To achieve the self-management goals, the architecture should be based on a MAPE-K control loop to add autonomics to the devices. For an optimal autonomous system, the control loop logic should be integrated in the device itself. The drawback of this centralized approach is that it is too complex to develop and it requires that the devices have enough resources and computational capabilities to perform complex analysis. Therefore, using the combination of both centralized and decentralized approaches, where the autonomic manager functions are divided between the two parties: managed resource and an external management entity allows for better scalability (in terms of the types of devices that can be managed). This approach also has the advantage that the

management entity has a global view of the environment knowledge, which helps in a better diagnosis of problems, and gives the possibility of widening the scope learning and enhancing the knowledge, this will help analyze the situation faster.

Ontological modeling: The model must be independent of application domain and deployed technology. The model should provide an expressive, flexible, and efficient representation for all sorts of physical and logical measurable characteristics such as object, time, organization, activity and most importantly, device and service details. The ontology language expressive power, at both theoretical and practical expressivity levels, will help simplify the exchange, and sharing of knowledge. A number of markup languages have been adopted by ontology applications for the purpose of context modeling. We will be using Web Ontology Language (OWL) [15] to model information within our device self-configuration system. OWL describes modeled data by using sets of classes and properties, and provides the flexibility and extensibility necessary within pervasive environments.

Device Management: our objective is to provide device management for a wide range of devices from different vendors along with management of some services they offer. The device management should provide support according to the following requirements:

- Ability to seamlessly access the management entity.
- Authentication of users and personalized preferences configuration.
- Obtaining seamless device and service management.
- Control and management of dynamic changes in the user preferences that may occur during service and device operation.

Therefore, the device management approach should be able to perform the following management operations:

- *Configuration management*: This includes the configuration of the devices component e.g. remote management parameters such as management server URL, monitoring periodic interval parameters, system parameters such as data processing, and application specific parameters such as parameters of user preferences, etc.

- *Performance Management*: This is related to the evaluation of device and/or service performance metrics, operation diagnosis such as capturing communication delay, computing component control such as memory and storage usage, etc.

- *Software Management*: This includes installation of missing pieces of software or plug-ins, firmware upgrade, and software modules update, etc.

Given the variety of data to be collected from devices with different technologies and coming from different vendors, the mechanisms used in capturing such information should use a standardized format and be capable of categorizing it. A matrix of device management protocols has been introduced in industry to express device component-specific parameters in a standard format. The most popular protocols for device management are: UPnP [10], OMA-DM [12], and TR-069 [14].

1.4. Thesis Contribution

We need to achieve the goal of a self-configuring framework for devices which provides a reconfiguration based on dynamic changes that occur either to the device components or services offered by these devices. Our self-configuration process is based on three steps: initial configuration, monitoring, and re-configuration of devices. Our system is directed towards achieving the above discussed goals.

The following is an overview of the main contributions of this thesis. First, the adoption of MAPE-K control loop to the device self-configuration context. Moreover, use of an active/passive monitoring approach, to capture device information. We then present how knowledge is represented within our system. This is done through the use of ontologies and policies to model the context and express rules, respectively. Finally, we present a prototype system implementation to prove the validity of our conceptual framework.

Adoption of MAPE-K loop: The device self-configuration framework we present within this thesis reinforces the autonomic concepts necessary to building self-managing systems. We base our framework architecture on the IBM MAPE-K control loop that we adapted to the context of device self-configuration. The framework requires the deployment of an

autonomic manager that represents the management entity for devices. The autonomic manager comprises the system's intelligence within the control loop functional components.

Active and Passive Monitoring: We use an active and passive monitoring approach. In the latter, monitoring is either organized to receive passively periodical messages summarizing the current state of the device or upon a change of value of some parameters. These parameters are first configured with a notification as being active or passive. As for active monitoring, we actively trigger monitoring to reinforce data analysis. The monitoring module gets feedback from the analysis module to request a parameter value from the device, and to complete the problem identification and target the re-configuration planning.

Ontologies and Policies: To fulfill the requirements of the architecture, a repository is needed to store context related to devices, services, users, and other environment-related information. To achieve easy communication between the management entity and the managed resources, the system requires a common understanding of how key concepts and knowledge are represented. Therefore, we propose to model environment context using ontologies and we describe how policies can be used to express the rules governing devices and services.

Prototype system implementation: We developed a prototype system implementation representing the two main entities of our architecture: control and management and the managed device. We used OpenACS as server that implements functionalities of the TR-069 auto-configuration server, and that allows advanced management functionalities by adding our logic in terms of scripts. We implemented TR-069 CPE, which is conforming to the protocol specifications, and which carries out complete communication with the server. Bed-tests have been carried out to prove feasibility, usability, applicability, and validity of our system.

1.5. Thesis Outline

The remainder of this thesis report is structured as follows:

In Chapter 2, we present an overview of some of the background information used in this thesis. This chapter presents core definition and concepts related to autonomic computing. It

also demonstrates the MAPE-based control loops adopted recently to achieve self-management in software and networks. We also present, in a comparative way, the most used industry device management tools and express the reasons that motivated our choice for TR-069. The chapter will also provide a description of some device self-management systems, showing architectures that have been proposed, outlining their main features, and presenting the weakness of these works compared with ours..

In Chapter 3, we describe our device self-configuration framework architecture. We describe in details our core capabilities towards building an autonomic system. We also describe how a policy-based approach will fit in the system to provide an autonomous and seamless management of devices. We have chosen an OWL-based ontology to model information related to device and services as part of our framework architecture. We have extended and integrated existing ontology.

In Chapter 4, we present our detailed implementation of a prototype illustrating the validity of our system. The chapter describes the implementation of the main components of our proposed self-configuration framework architecture and its use in reconfiguring devices. We implemented the TR-069 agent that should be available in the home device; it plays the role of the CWMP client that communicates with the ACS from one part, and with the user device application from another part. We exploit the functionalities of OpenACS, an open source project implementing the CWMP protocol based on the TR-069 auto-configuration server requirements.

In Chapter 5 we conclude this thesis. We summarize our contributions and the lessons learned from our self-configuration system architecture. We also discuss our future research plans and potential enhancements to our current design.

Chapter 2

2. Background and Related Work

2.1. Objectives

This chapter presents an overview of some of the background information used in this thesis. The chapter gives core definition and concepts related to autonomic computing. It also demonstrates the MAPE-based control loops adopted recently to achieve self-management in software and networks. The chapter then proceeds to discuss, in a comparative way, the industry most used device management tools and express the reasons that motivated our choice for TR-069. Finally, the chapter provides a description of some device self-management systems, showing architectures that have been proposed, outlining their main features, and presenting the weakness of these works compared with ours.

2.2. Autonomic Computing

2.2.1. Definition and Roles

The term "*autonomic computing*" was first used in October 2001, by Paul Horn, IBM's senior vice president of research [3]. Autonomic computing describes a vision of computing systems having the capability to self-manage their behaviors with respect to high-level administrative objectives, generally directed by humans. The goal of autonomic computing is to deal with the crisis of "ever-increasing complexity in the management of large-scale distributed and heterogeneous information technology infrastructure" [1].

In the near past years, autonomic computing has gained an important momentum. Numerous and large scale research were devoted for this area, as well as new disciplines related to autonomic computing have emerged in the computer science field. Academia and industry professionals are combining their efforts through workshops and conferences to help going forward to overcome the topic challenges and fulfill new achievements. As a result of these efforts, dozens of products offer now hundreds of autonomic features that make the life of

administrators much easier. Though, researchers' destination is by no means has been reached. "No one has yet built a large-scale, fully autonomic computing system - embracing multiple components that work together to satisfy high-level business goals - that reveals the ability to configure, heal, optimize, and protect itself." [3]

To deal with all the complexities facing the communication networks, autonomic computing researchers are going back over the existing fundamentals and paradigms. Numerous researchers have been fascinated by the nature systems that demonstrated a perfect harmony and a reliable efficiency. For this reason, autonomic computing researchers are looking to self-management and self-organization in nature (in human body or colonies of insects, for example).

Self-organization of autonomic communication networks is as an example of autonomic computing systems that has gained a great importance in the community. Self-organization is defined in [7], as the act by which a "system is able to evolve toward a coherently organized configuration without external intervention," self-organization mechanisms conceives a system as if it were able to self-manage by its nature, and without any external intervention of "non-self entities" (humans).

Though, draining modern multimedia communication systems laws from natural phenomena is not evident. From one side, such a "reverse-engineering" approach or other approaches need a deep understanding of the natural phenomenon in question, whereas, the mechanism behind the complex behaviors have not been studied fully [1]. From another side, decentralization in self-management at local entities remains a challenging task.

2.2.2. Concepts and Principles

Autonomic computing aims for efficiency, immunity, resilience, and evolvability taking into consideration some self-* properties: self-optimization, self-protection, self-recovery, self-adaptation, and self-configuration. Condensing complexities encountered in the networks, taking advantage of their intelligence, and making use of modern technologies, human intervention in an autonomic system can be greatly reduced while management efficiency can be pushed to its extremes.

Autonomic systems can be characterized by the following principles: context awareness, distributed policy-based management, self-*, autonomy, and evolvability as described in [1].

- *Context awareness*: be aware of the surrounding context by having the network elements sensing, perceiving, managing functional interdependence between them, as well as responding, reactively or proactively, to any contest change.

- *Distributed policy-based management*: this designates a need of a proactive approach to self-organization of the policy-based systems. This implies that the system should be able to anticipate and eventual actions and make the right decision, in a dynamic way, according to the predefined policy.

- *Self-**: This represents the self-management properties; self-configuration, self-optimization, self-protection, and self-healing. This means a total autonomous and complete process in adjusting the functionalities according to any change in the environment, taking into consideration the functional behavior and purpose, towards a adaptive, optimal, secure, and resilient system.

- *Autonomy*: a self-governance of the network element with a limited and tiny human intervention in management, after having a well defined high-level objectives. This includes the abilities of information retrieval and processing, learning, adapting, evaluating, evolving, etc.

- *Evolvability*: this implies shaving a flexible design open for yet unknown new extensible features and vision and system scalability. This requires the development of behavior patterns and the corresponding rules, and the coupling of this with self-awareness, so that the emergence of the new behavior can be modeled and hence be managed explicitly.

2.2.3. MAPE-K based Control loop

2.2.3.1. Self-Managing software

Researchers believe that self-managing software can guarantee safer, more reliable, and cost-effective computer systems. Developing software systems that are self-directed, self-governing, and self-adapting has been the focal point in maturity of autonomic computing

and autonomic communications. To realize the self-managing objectives a system must have the following attributes as described in [8]:

- Self-aware (aware of the system internal state)
- Self-situated (aware of current external operating conditions)
- Self-monitoring (able to detect changing circumstances)
- Self-adjusting (able to adapt accordingly).

The control loop presented in Figure 2-1 shows the different actions for an autonomic element. As shown, the control loop is divided into two sub-control loops, the first being related to the self-awareness of the autonomic element itself, the second being related to the environment-awareness (or what has been referred to as self-situated) and this involves the components that are interacting with a given autonomic element.

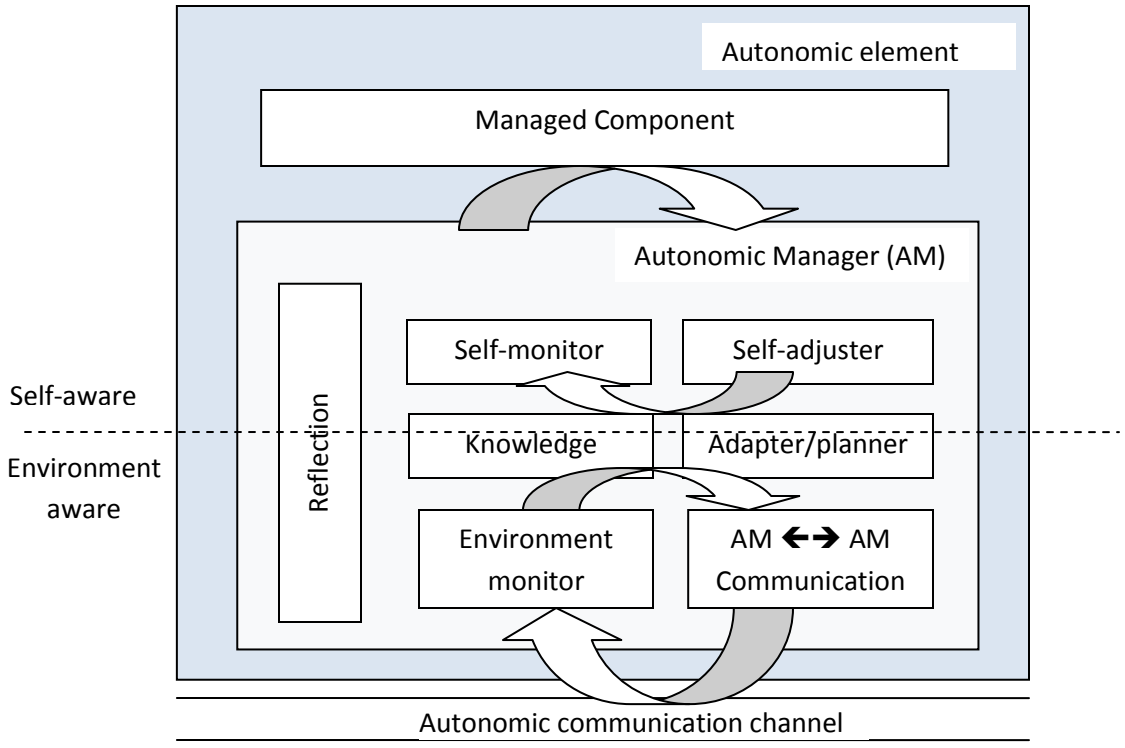

Figure 2-1: The control loops in an autonomic element [8]

Each of these two loops is based on two central elements: sensors and effectors. The sensors monitor the behavior of autonomic managers comparing to available historical and current data, rules, beliefs, and policies. Then the appropriate action plan, if necessary, is undertaken and executed through sensors. IBM separates the functions of sensors and effectors (self-monitoring and self-adjusting) in what is called MAPE: monitor, analyze, plan, and execute. According to [8], the first two former functions are the duties of the sensors, whereas the two others are reserved for the effectors. Another important component in the figure above is reflex signals. For an efficient and robust interoperability between the autonomic managers, a continuous communication is required. This communication includes reflex signals that are represented by "I am alive" messages to notify the other components of the correct operation. This is inspired from the double biological heart beats. Authors also highlight an interesting safety mechanism that is self-destruction, also drawn from nature, where the body tends to destruct some infected cells. Following the same logic, to ensure security, the system should go autonomously for self-destruction.

2.2.3.2. FOCALE Control loop

Following the same objective of achieving autonomy in communication networks, researchers have fixed the aim to simplify network management processes by automating and distributing the decision making processes. Authors of [9] propose the FOCALE architecture for network communication. They assume that the core idea of autonomic management is the ability of a system to self-govern its behavior, but only within the constraints of the (human-specified) goals that the system tries to achieve. The authors emphasize on the use of information and ontological modeling to capture environment knowledge related to network characteristics, environment and context constraints, and business goals and policies, along with reasoning and learning mechanisms, to evolve the captured knowledge. The policy-based network management system makes use of the knowledge incorporated within system models, featuring translation/code generation and policy enforcement techniques that automatically configure network elements in response to changes in the environmental context. The flow of information is described in Figure 2-2.

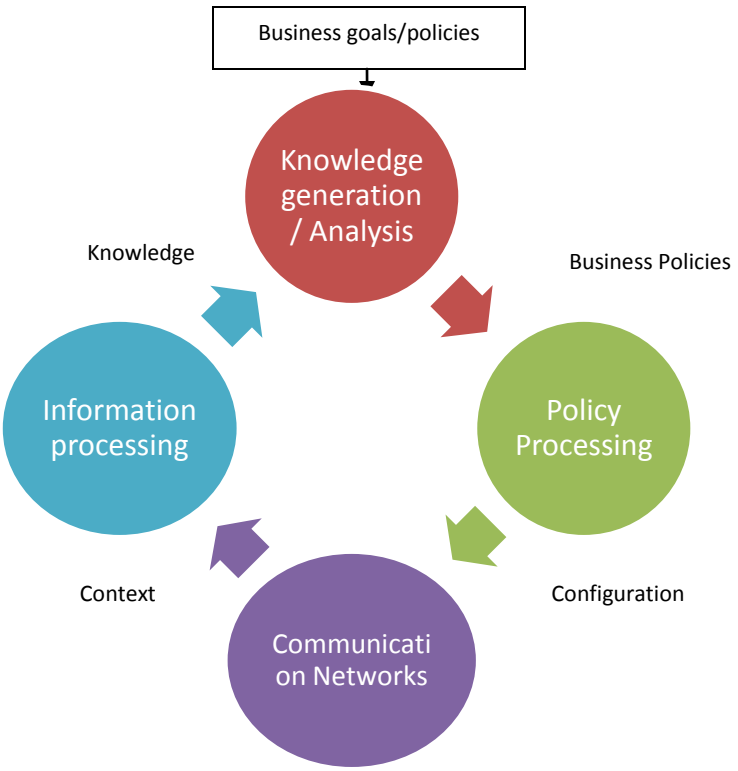

Figure 2-2: Conceptual autonomic control loop for network management [9]

The FOCALE (Foundation Observation Comparison Action Learn rEason) architecture embodies all these concepts. FOCALE is takes into consideration all the changes that affect the system, this includes dynamic changes in: "business objectives, user requirements, and environmental context. For this, two control loops have been defined in FOCALE: a *maintenance control loop* that is used when no critical changes are detected in the system and an *adjustment control loop* used when policy configuration actions must be carried out and/or new policies must be generated.

Figure **2-3** shows the FOCALE autonomic management architecture. It contains two main functional components: the Autonomic Manager (AM) and the Model-Based Translation Layer (MBTL). Each AM realizes the autonomic management functionality described in the previous section via an event manager, a state manager, an action manager, a reasoner, a learner, and a policy analyzer/policy decision point (PDP). All these subcomponents can communicate with each other and have access to the

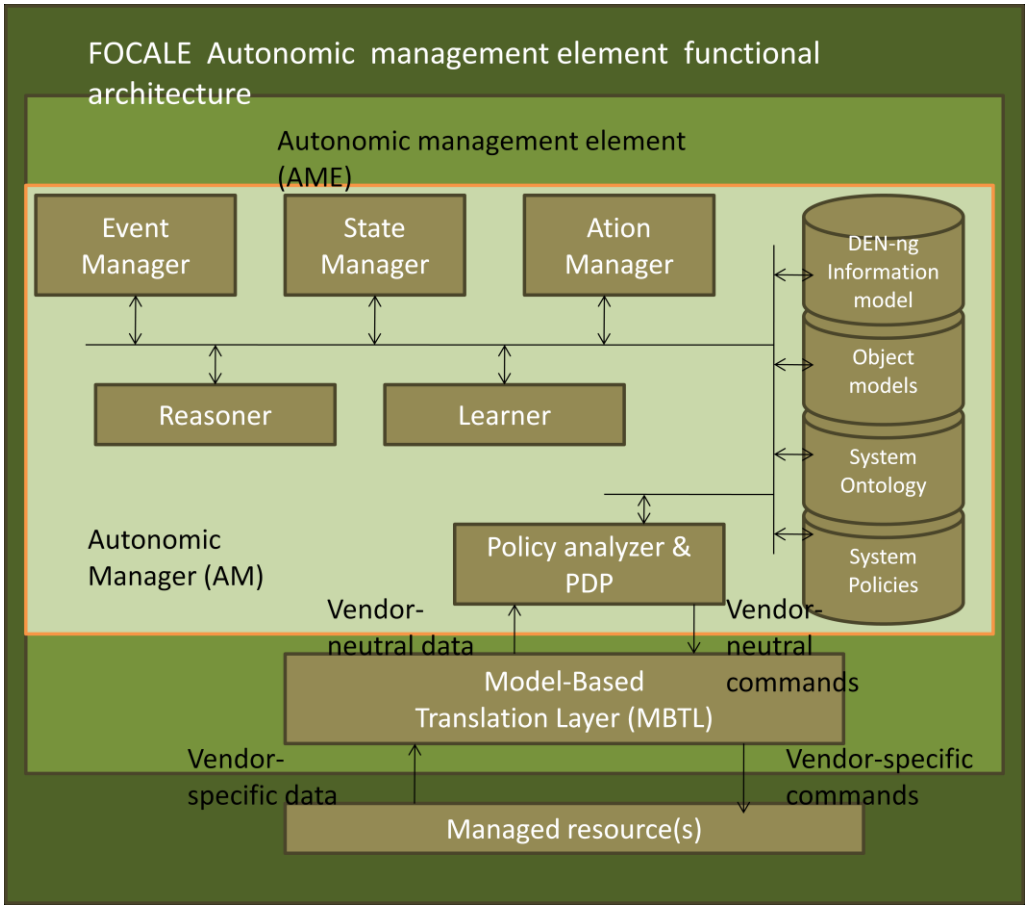

Figure 2-3: FOCALE autonomic management element functional architecture [9]

DEN-ng information model, an object model reflecting the state of the autonomic manager element's managed resource(s), the system ontology, and the set of policies ruling the managed resource(s). More details on the mechanism of this architecture can be found in [9].

2.3. Device Management Tools

2.3.1. UPnP Device Management

Universal Plug and Play (UPnP), promoted by the UPnP forum [10], sets a distributed computing framework that defines a group of networking protocols designed for residential and corporate environments holding networked devices. The goal of UPnP is to allow

devices such as personal computers, printers, residential gateways, Wi-Fi access points, and mobile devices to connect and discover each other seamlessly, in addition to control and data sharing and transfer among them. As its name indicates, UPnP devices are plug-and-play in the network and automatically deliver their capabilities to other devices present in the network.

The Device Management (DM) Working Committee is part of the UPnP Forum Working Committees that develop standards for UPnP services for managing device firmware, software, configuration, and maintenance, diagnosing network and device problems, and monitoring device and network performance. Devices can be controlled from any part of the network. Other committees are specialized in developing standards dedicated to services such as audio / video services where the AV standards feature capabilities such as storage management, playback control, and scheduled recording. Quality of Service committee provides standards to enhance the user experience of networked services that need a management of bandwidth, and network latency of multimedia streaming services. UPnP device control protocols definitions can be found in [10].

UPnP protocol Stack:

Figure 2-4 presents the UPnP protocol stack. The first three layers are UPnP related protocols. UPnP device architecture sets the required template device or service description for a particular device. UPnP forum-specific corresponds to the forum working committees' domain- and device-specific meaning and format of data definition built on the top of the

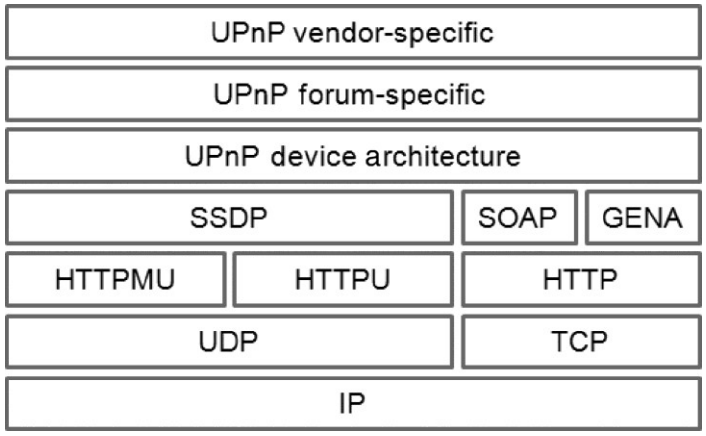

Figure 2-4: UPnP protocol Stack [10]

UPnP device architecture. UPnP vendor-specific is the vendor's extension of the protocol with regards to the device specificities. For example, if the working committee sets the general information of UPnP enabled printer, then the vendors complete the information by providing information specific to their device such as device model name, serial number, and URL. General Event Notification Architecture (GENA) is used for sending and receiving notification messages. Simple Object Access Protocol (SOAP) is used to execute Remote Procedural Calls (RPC), both protocols use HTTP over TCP for description, control, and eventing. Simple Service Discovery Protocol (SSDP) is used to advertise devices in the network it using HTTPU (Unicast) or HTTPMU (Multicast) over UDP for discovery. All the above protocols are built on the Internet Protocol (IP).

UPnP Technology:

Figure 2-5 depicts a canvas of the UPnP devices and services overall technology. UPnP devices advertise services to the UPnP control point. Every UPnP device implements a device type standard developed by the UPnP working committee. The device type

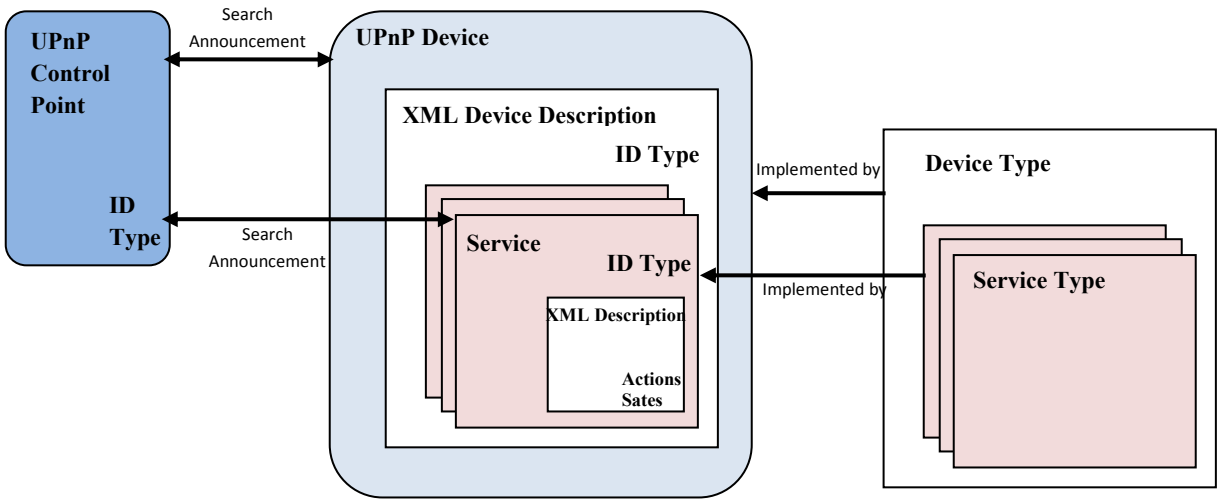

Figure 2-5: Overview of UPnP Devices and services [11]

customized information is enclosed in the XML Device Description along with the device ID and a service list, representing the set of services offered by the device. Every service has a service ID and is linked to an XML Service Description that specifies the UPnP service type that is implemented together with the service actions and states.

Table 2-1 summarizes how general device and service technologies map within the UPnP technology.

Table 2-1: UPnP device and service technology [11]

Function	UPnP Technology	
	Device	**Service**
Representation	XML-based Device Description implements Device Type	XML-based Service Description implement Service Types
Classification	*Device ID* : URI specified by vendor *Device Type*: URI specified by UPnP Forum or vendor	*Service ID*: URI specified by UPnP Forum or vendor *Service Type*: URI specified by UPnP Forum or vendor (e.g. PrintBasic:1)
Discovery	Using SSDP: • Control point searches for devices via multicast • Device announces itself via multicast	Using SSDP • Control point searches for services by service types • Device announces services via service types
Service invocation	-	SOAP over HTTP (over TCP or UDP)
Event notification	GENA over HTTP	

2.3.2. OMA Device Management

Open Mobile alliance (OMA), is a standard-developing organization responsible of developing open standards for mobile phone industry [12]. The goal of OMA is to provide interoperable enablers across different devices, countries, service provider, operators, and networks. OMA defines different standard specifications. OMA Device Management (OMA DM) working group develops protocol standards that intend to small mobile devices such as mobile phones, PDA, and palm top computers [13].

The Device Management functionalities includes: configuration of devices (including initial configuration) by enabling and disabling feature and/or allowing changes to parameter settings of the device, software upgrades (including application and system software), and fault management comprising device error reports and device status queries. Since OMA DM is aiming mobile devices, therefore designed standards take into consideration the following issues: resource limitation (i.e. memory and storage space), constraints of

bandwidth of communication especially the case of wireless connectivity, and security issues.

OMA Technology:

Figure 2-6 depicts a canvas of the OMA devices and services overall technology. OMA DM devices are managed by Device Management Server (DMS). An OMA device is identified by a model number. Every device offers a set of services. These Services are expressed as

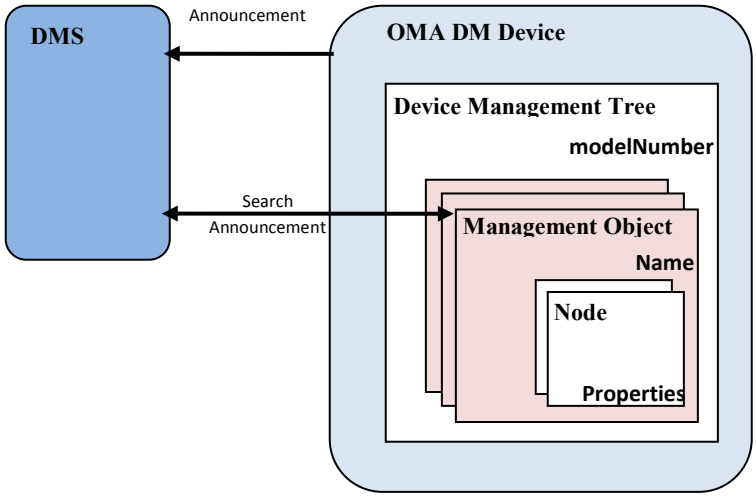

Figure 2-6: Overview of OMA Devices and services [11]

management objects according to the OMA DM Device Management Tree, i.e. a type of data model. The services contain nodes that have properties corresponding to the actual service properties which are managed by the DMS. To configure the management objects, the calls are done over Synchronization Markup Language (SyncML) for data synchronization.

Table 2-2 summarizes how general device and service technologies map within the OMA DM technology.

Table 2-2: OMA DM device and service technology [11]

Function	OMA DM Technology	
	Device	**Service**
Representation	-	Management Objects in the hierarchical data model (device manage-ment tree)
Classification	*Device ID* : Device manufacturer + model number – specified by vendor	*Service ID*: Management Object Name specified by OMA or vendor (e.g. DevInfo)
Discovery	Using SSDP: • Device announces itself on the network to all DMS it detects.	Using SSDP: • Device announces services on the network Over SyncML Methods: • DMS searches for management objects and their associated Properties
Service invocation	-	SyncML over HTTP / WSP / OBEX
Event notification	User / Device initiated (over SyncML) DMS initiated (over SyncML or WAP Push over SMS)	

2.3.3. CPE WAN Management Protocol (CWMP)

CWMP [14] (or commonly known as TR-069) is a remote management protocol for communication between Customer Premises Equipment (CPE) and a management entity in the form of an Auto-Configuration Server (ACS) whose functionalities are summarized in: auto-configuration of user CPEs (including initial configuration), dynamic service

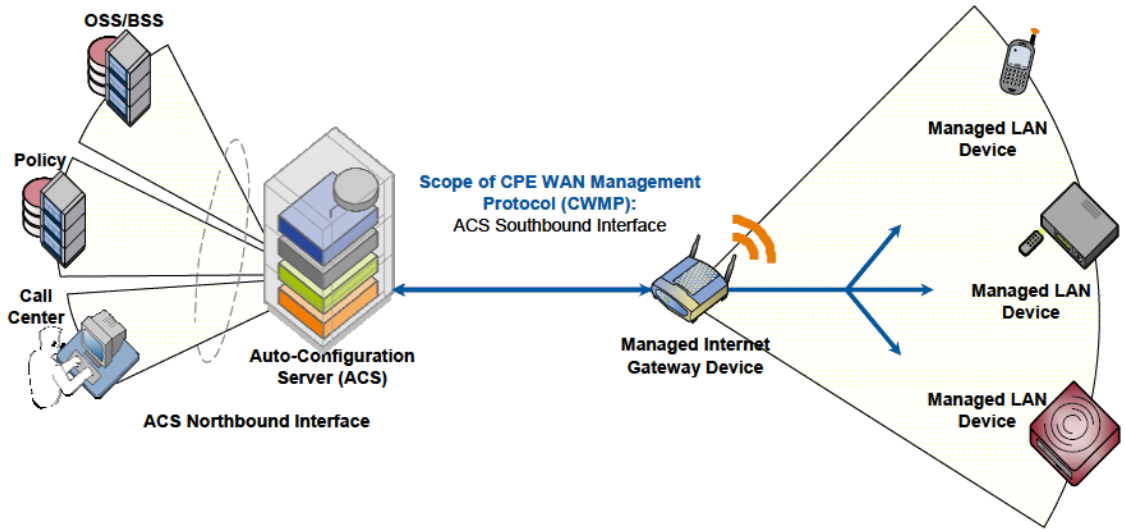

Figure 2-7: TR-069 End-to-End Architecture [14]

provisioning, software/firmware image management, status/performance monitoring, and diagnostics. The ACS has two communication interfaces: southbound interface; for communication with user CPEs, and northbound for communication with third-parties such as service providers (Figure 2-7).

In this work we are more concerned with the former. BBF also introduced a family of CWMP-based data models to suit different devices that inside the home such as Residential Gateways (RGW), VoIP phones, Set-Top-Boxes (STB), games consoles, etc. The data models structure the set of components, capabilities, and services of a device in such way related information can be easily integrated in the communication messages the ACS exchanges with the CPE.

TR-069 Protocol Stack:

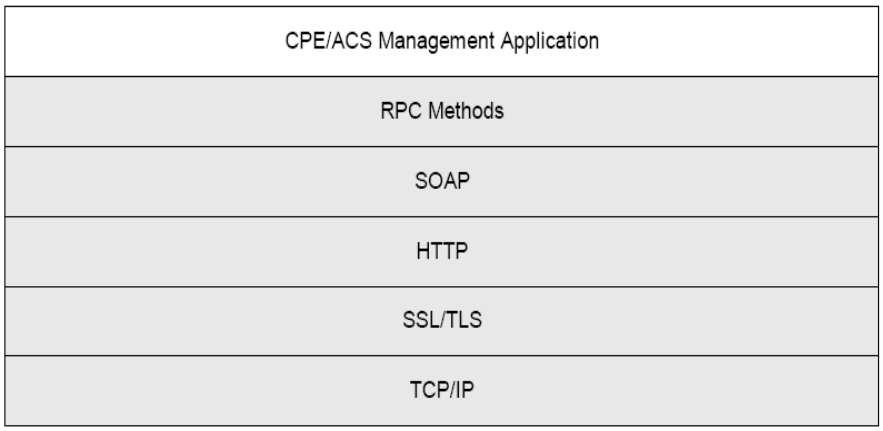

Figure 2-8: TR-069 protocol Stack [14]

Figure 2-8 presents the RT-069 protocol stack. The first layer represent the CPE/ACS Management application, this means the application uses TR-069 on the CPE and the ACS. The RPC Methods layer corresponds to the specific RPC methods that are defined in the CWMP. These methods are presented in Table 2-3. The RPCs are encoded using SOAP XML-based syntax. Secure Socket Layer/Transport Layer Security (SSL/TLS) adds security to communication between ACS and CPE as it might be needed in some transactions. The exchanged messages are carried over HTTP and TCP/IP protocols. TR-069 uses SOAP remote procedure calls to achieve description, control, and eventing, unlike UPnP that uses

HTTP for description, SOAP RPCs for control and GENA for eventing. Table 2-3 describes the baseline RPC methods used in the CPE WAN Management Protocol.

Table 2-3: The Baseline Methods of TR-069 Protocol

Function	RPC Method	Description
Description	*GetRPCMethods*	Can be called either by ACS or CPE to discover the set of methods supported.
	GetParameterNames	Called by ACS to discover parameters accessible on a CPE
Control	*SetParameterValues*	Used by ACS to modify value of one or more CPE parameters
	GetParameterValues	Used by ACS to obtain value of one or more CPE parameters
	Reboot	ACS forces CPE to reboot whenever configuration changes requires CPE to reboot before they are applied
	Download	Used by ACS to command the CPE to download a certain file from a specific location. Used mainly for firmware updates.
Eventing	*Inform*	The basis of eventing. It is sent whenever the CPE initiates a session with the ACS. It indicates one or more events that caused the transaction session (e.g. *BOOT*, *PERIODIC*, etc.). It also includes the changed CPE parameters that ACS should be aware of.
	SetParameterAttributes	Used by ACS to declare the parameters that should generate events whenever changed.
	GetParameterAttributes	ACS calls this method to retrieve the notification and write access attributes for each CPE parameter
Presentation	-	Not available since CPE operates as an HTTP client

TR-069 Technology:

Figure 2-9 depicts a canvas of the TR-069 devices and services technology TR-069 devices hold multiple services that can be remotely managed by the ACS. In other words, the ACS is not responsible for invoking the service but rather managing it (configuring it). TR-069 devices are identified by a serial number and a type (other parameters such as manufacturer OUI makes part of the device identification parameters). The services are represented in terms of service objects that can be found in the TR-069 data model corresponding to the device (e.g. TR-098 is the data model used for a Gateway device, where service objects can be added as *ABCService*). These service objects contain parameters, with names and values, which characterize the actual service. Those parameters are monitored and configured by the

ACS. To limit the scope of services deployed in the devices, services are classified by profiles that are specified in the data model technical reports found in BBF.

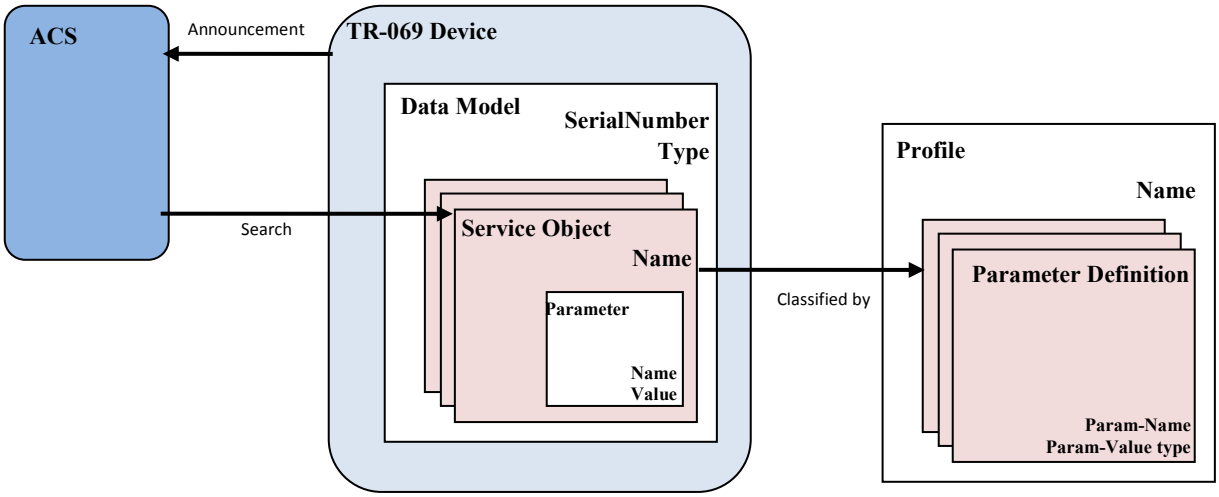

Figure 2-9: Overview of TR-069 Devices and services [11]

Table 2-4 summarizes the device and service function within the TR-069 protocol.

Table 2-4: TR-069 device and service technology [11]

Function	TR-069 Technology	
	Device	**Service**
Representation	Component objects that represent the device components in the TR-* data model	Service objects in the hierarchical data model which can be classified by profiles
Classification	*Device ID*: ManufacturerOUI + ProductClass + SerialNumber – specified by vendor *Device Type*: CPE Device Type – "Device" or "InternetGatewayDevice" speci-fied by BBF	*Service ID*: Service Object Name specified by DSLForum or vendor (e.g. VoIPService:1.1) *Service Type*: Profiles specified by DSLForum (e.g. SIPProfile:2)
Discovery	Initial connection from Device to ACS: • ACS URL via LAN side CPE auto-configuration (TR-064) • ACS URL from DHCP server via DHCP option • default preconfigured ACS URL	Over proprietary RPC Methods • ACS searches for services by profiles which are represented by as-signed profile names • ACS directly searches for objects and their associated Parameters
Service invocation	-	SOAP over HTTP (over TCP)
Event notification	CPE initiated or asynchronous ACS initiated (over proprietary RPC Methods)	

After this comparative presentation of the most leading device management protocols in the community, our choice in implementing the device management part of our framework was for CPE WAN Management Protocol. Apart from the fact that is the most used protocol for remote management of devices, in the following we give reasons that motivated our choice for TR-096:

- UPnP is leading protocol that is limited to internal management of devices within a given network environment to provide a management. In this thesis work, we would like to provide control and management for a wider range of network environment simultaneously. Hence the need for an external management entity of the environment networked devices. CPE WAN Management Protocol can perform such management by having the ACS present in the WAN and contacts the home or corporate network devices that might be in a LAN either directly or via a control point, such as a residential gateway.

- OMA from one part concerns limited device profiles (mobile phone like devices), Those devices can be included into the scope of devices that can be managed by ACS, where ACS can replace the DMS in the OMA architecture.

- TR-069 end-to-end architecture allows an extension of management functionalities that will permit us to implement our device self-configuration framework. This will be realized by adding necessary analysis and management components behind the ACS.

2.4. Ontologies

In autonomic systems where knowledge plays a great role in identifying all environment components, it is important to have a unified method of modeling and representation of context information. The system must also be flexible and extensible enough to handle the wide range of context types, as well as the relationships between them. Most context-aware systems are distributed; as it is the case of autonomic systems, it is therefore necessary for context models to be easily shared especially in our architecture, given its use by devices and autonomic manager. Models should also have a high level of formality and be able to

represent existing context relationships. For these reasons, various context modeling methods have been developed by researchers.

Ontology-based management provides a mechanism to represent the knowledge following a predefined and structured specification of concepts. Ontologies have been widely used to model context. In our work, by context, we particularly mean all information (i.e. facts and circumstances) that describe the environment a device is in. This information should include also details about the device itself, its component, the services deployed in it, the network connectivity, communication bandwidth, etc.

Ontologies as our preferred choice due to the many advantages; mainly because ontologies can represent concepts and relationships by employing a computer-usable data structure while also sharing a common understanding of the domains in which the modeled context can be easily used for identifying the detailed information about devices and service parameters. Ontologies are structured to be represented through a set of entities, functions, instances, and axioms. Ontologies are also known for their normalization abilities and formality, making them a favorable candidate for modeling context knowledge.

Most proposed ontology-based context models have adopted the OWL language [15] as a means of ontology representation. This is due to OWL's superior expressive abilities over other languages, such as XML, RDF, RDFS, or DAML+OIL. OWL also allows for the exchange and comprehension of context information between different entities of a system. The availability of a number of OWL-based reasoning engines that can interpret context information present within the ontology or that can infer new context from existing context and relationships, makes OWL an especially effective language for context modeling.

For a better understanding of ontologies and how they can be relevant in our system, we present some proposed ontologies in the literature that we based on our ontology.

As there is no unified OWL-based ontology for knowledge representation, several research groups have proposed a generic context model.

Stimpakou et al. proposed an ontology called COMANTO [18]. The proposal aims to enable various end-users, context providers, service providers, resource providers, and

manufacturers to share a common vocabulary and to collaborate and synchronize their knowledge. COMANTO (COntext MAnagement oNTOlogy) is a generic upper-level context ontology that is not specific to any domain, application, or condition. On the top level of the ontology is a class called Semantic Context Entity (SCE). This class represents the root from which the most widely used concepts extend. Several subclasses that extend this major super class have been identified. These concepts have been categorized into 12 core or sub-classes as shown in Figure 2-10. The following is a brief description of these core classes.

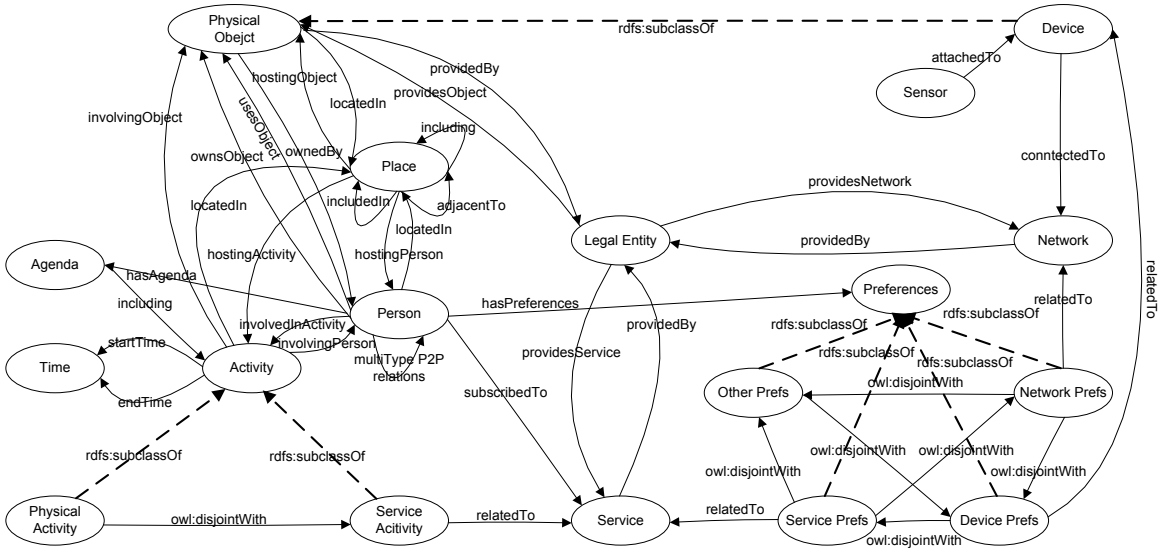

Figure 2-10: COMANTO upper-level ontology [18]

- Person class: This class corresponds to all human entities and offers various properties for incorporating the user related context, relationships, and behaviors.
- Place class: This class represents the abstraction of a physical spatial place. It is used to describe the physical location in terms of symbolic or geographic representation.
- Preferences class: This class represents user preferences in a generic manner. It is used to describe the user's device, service, and network preferences.
- Agenda class: This class describes the calendar information model of a user. This class is necessary to efficiently model the user's activity.
- Activity class: This class describes information relevant to the user's task during the specified task's duration. The class is divided into two types: the physical and the service activities.

- Time class: This class contains all necessary information related to the current time. It serves as a timestamp for all context information that may change over time.
- Physical Object class: This class is a super class of another class called "Device." The physical object class describes any object physically, except for devices.
- Sensor class: This class represents sensors that are used to collect context information. It contains datatype properties of sensors as well as configuration features.
- Service class: This class represents information to all services and applications to which the user subscribes.
- Network class: This class models context information related to the underlying network.
- Legal Entity: This class represents the corporate actors involved in the system.

Strimpakou et al. used the COMANTO ontology to describe the properties, relationships, and structure of context objects, while their context management infrastructure employed an object oriented context model for context information communication. Similar upper context ontology has been introduced in OWL Encoded Context Ontology (CONON) [19] that captures general concepts about basic context and also provides extensibility for adding domain-specific ontology in a hierarchical manner. CONON's domain-specific low-level ontologies can be dynamically plugged and unplugged from the upper ontology based on changes in the environment. The upper ontology is broken down into four subcategories: person, location, computational entity, and activity.

Another attempt in gathering the context related information for autonomic pervasive environments was provided by Ganna and Horlait in [20]. Their ontology captures the environment knowledge to deal with autonomic system's adaptation. They used a built-in interface featuring OWL capabilities to deduce new environment conditions and concept's associations. Figure 2-11 shows their environment's ontology drawn as UML diagram.

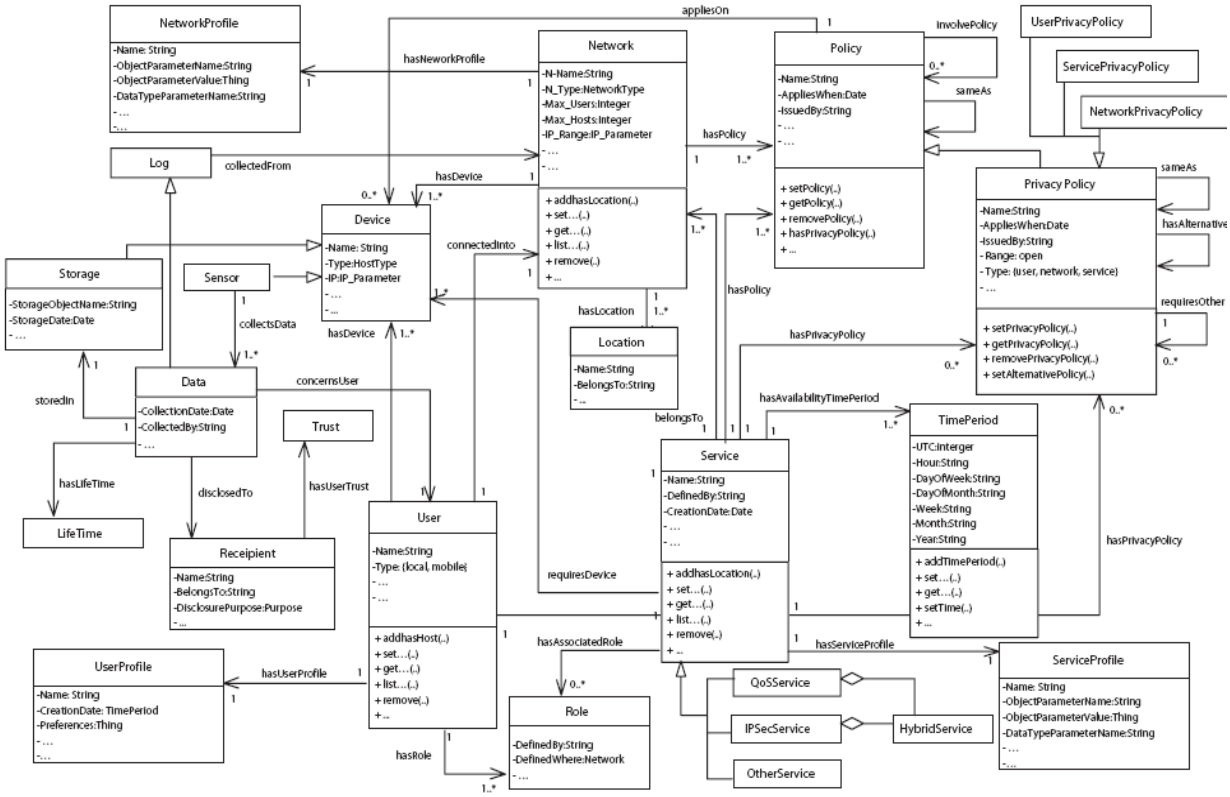

Figure 2-11: Environment's Ontology as UML Diagram [20]

The four main elements in the above ontology as outlined by the authors are:

- Policy: this element corresponds to the authorization policies that control access to the environment and services. Also new policies can be generated according the inference property *sameAs*.
- Network: represents the network general and specific characteristics, the location of the network, the devices present in it, the users connected, etc.
- Service: reflects all services that are offered in the environment, and can be used by users.
- User: similar to class person described above, it represents an actual user within the environment.

In our work we choose to adopt and extend the OWL-based ontology conducted by Y. Al Ridhawi [16] and I. Al Ridhawi [17], within our research laboratory, to model our context information within our system since it covers most of the required information.

2.5. Device self-management Systems

Self-management is defined as the process by which computer systems shall manage their operations and computations, ideally, with no human intervention. The purpose being to put less technical pressure on both system administrators from one end, and system users, from the other end. This is done by hiding the details of system operation and maintenance. Hence, the key idea is to let the machine do the management incessantly

Device self-management and self-configuration, in particular, has been the focus of research for both academia and industry that are working together towards achieving fully autonomous systems. In the remaining of this section we describe some selected research work done towards developing device self-management systems.

2.5.1. ADsM

Bernd S. And Klaus S. [21], from the Nokia research center, aim to achieving the goal of self-management. They introduce an Advanced Device self Managing (ADsM) framework as an evolved solution for mobile phone devices (not necessarily present inside a home network) featuring the use Open Mobile Alliance Device Management (OMA DM) [12]. The proposed framework provides an added-value to the market by applying autonomics and therefore enabling Self-Management of mobile terminals. Authors doubt that the centralized Device Management Server approach can be effective with the current and future fast changing environment. In a way that the configuration that the management server provides the device with, during a given communication session, might not be helpful for an unforeseen device state that occurs when the device is no longer connected to the server. Such situations can happen repeatedly as the environment is changing quickly. Hence, adding an autonomic behavior to the device will improve the current DM approach by evolving towards an advanced device self-management.

To achieve this goal, a combination of the OMA standard features and smart policy based scheduling is proposed. Following this approach, the device will take care of configurations decisions according to schedules and smart policies defined locally. The device will only seek help from higher system layers (up to reaching human support) when unfrozen and

critical situations happen. Figure 2-12 depicts an overview of the device management and control logical architecture. It is based on the end-to-end re-configurability (E^2R)

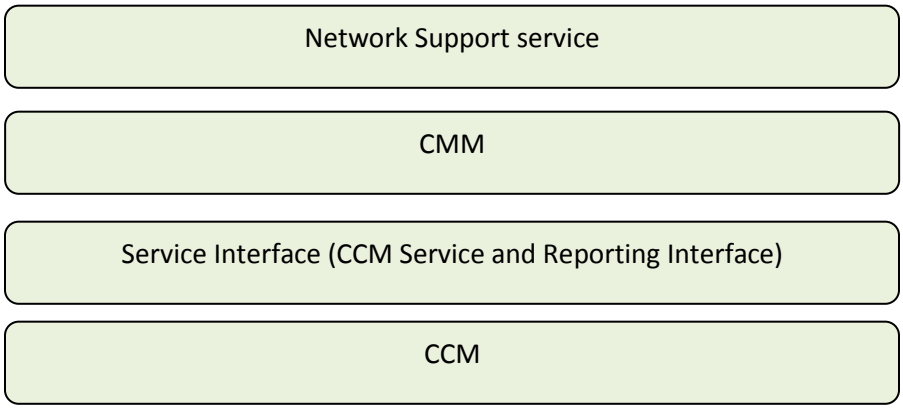

Figure 2-12: Overview of Device Management and Control [21]

architecture [22] where three types of modules are identified: *Configuration Management Module*, *Configuration Control Module*, and *Configuration Execution Module*. Detailed description of the two first modules component can be found in [21]. We give, hereafter, a brief description of each module:

- *Configuration Management Module (CMM)*. The CMM is the module responsible of the management of reconfiguration processes according to predefined "semantics, protocols and configuration data model." It is also accountable for making decision on the accurate functional configuration for the equipment.

- *Configuration Control Module (CCM)*. The CCM is an intermediate module between management module and execution module. It interprets the configuration requests coming from the CMM into real implementations on the hardware resource.

- *Configurable Execution Module (CEM)*. A CEM is the module that comes underneath the CCM module (in terms of hierarchy as shown in Figure 2-12). It represents the targeted configurable hardware resource. CCM can configure one or multiple CEM.

2.5.2. EnComPAs2

Alonso et al. [23] describe Telefonica's experience in residential customer networks management attained in EnComPAs2 CELTIC project. Their proposed Extended Home Network Remote Management Platform (EHNRMP) architecture presents a global architecture for home network management taking into consideration the main concepts of autonomic computing to achieve the self-* properties in management.

EHNRMP architecture design takes into consideration interoperability issues between the management components and devices. This is done by consolidating the use of industry available standards (e.g. OSGi HGi, TR-069, UPnP, or OMA) expressed as enablers to achieve remote device management. EnComPAs2 proposes a technology independent framework by having an integration layer, which also addresses the management technologies convergence with respect to both fixed and mobile devices. The platform addresses two types of management: one that is locally within the customer network leaving some management decision taking to the residential gateway (RGW). The second is remote in which a centralized system takes place, the objective being to ease the task for centralized systems, especially with regards to scalability matters, where the centralized management entity has to deal with millions of devices.

Figure 2-13 illustrates the EHNRMP architecture components. The architecture is divided into several layers which contains subsystems. The upper layer represents Customer Support Subsystem (CSS) which offers a web interface that is accessible to both support staff and end-users; it gives a transparent summary of the available services. The Service Management layer which copes with the management of service operations. The Device Management Layer, on the other hand deals with all implementation technologies of devices and it composed of two subsystems: Remote Configuration Subsystem (RCS) that deals with management of devices and software present in the home network via different technologies

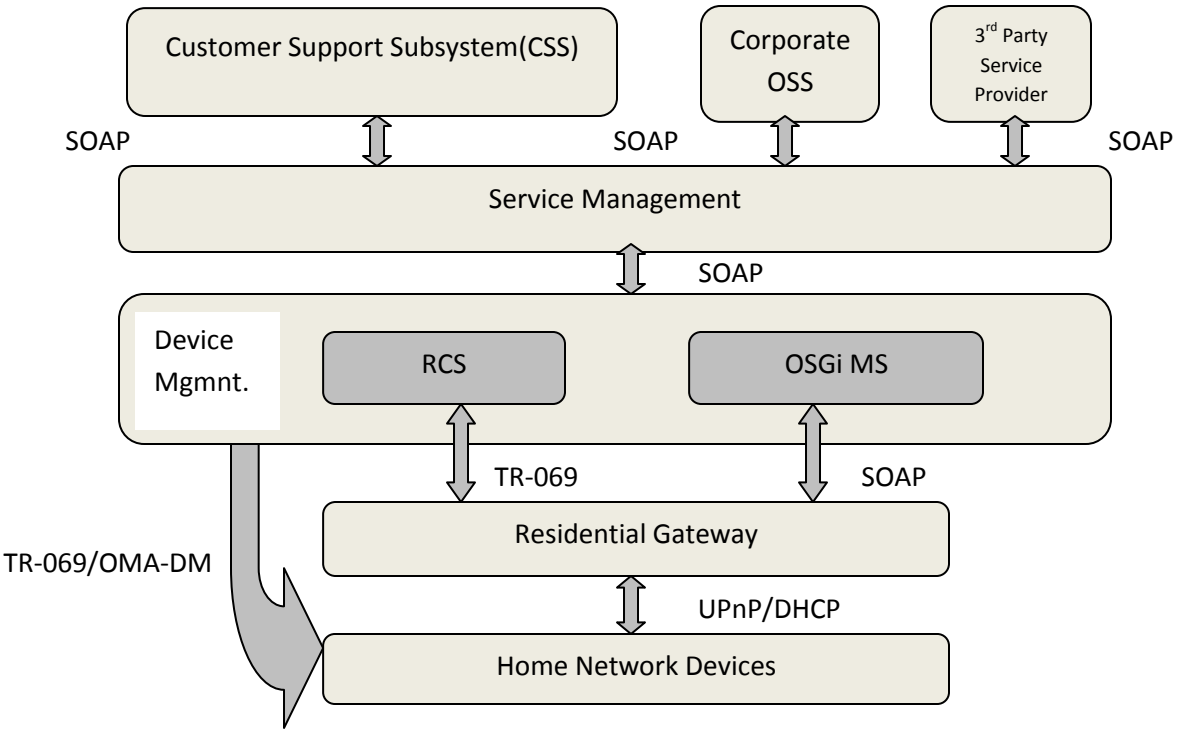

Figure 2-13: EHNRMP Architecture [23]

(e.g. TR-069, OMADM, OMA-CP). The OSGi Management Subsystem (OSGiMS) takes care of the "lifecycle, dependency and configuration management of software bundles

located in the OSGi-Enabled Residential Gateway or in any other OSGi-enabled device in the customer network."

Most communication among upper management is done through SOAP. The residential gateway as stated above takes part of the device management, especially the tasks particular to the customer network. The most common device management protocol inside a customer network is UPnP (more details about UPnP, TR-069, and OMA protocols are provided in section 2.3.

Both previous projects present device self-management frameworks with interesting approaches, and tackle most of the issues and challenges facing autonomic computing. However, neither works presented convincing implementation and/or testing details to assess system validity and usability. In this thesis, we present a general architecture for device self-

management, and we give a general control loop implementation for device self-configuration using CPE WAN Management Protocol.

2.5.3. Other Related Work

Nikolaidis et al. [24] investigate the capabilities of CWMP protocol and propose a CWMP-based methodologies and suite tools that make use of the advantages of the standard XML representation to create and manage diverse device Management Information Base (MIB) objects, by distinct parties collaborating in remote configuration, management, life-cycle support and testing of devices. In [25] Nikolaidis et al. addressed the issue of increased management packets size. In particular they investigated the repetitive nature of text patterns in typical configuration and management XML messages that are produced during a TR-069 session between a CPE and an ACS. They applied the Lempel-Ziv compression algorithm in the solution proposed. Results support the applicability and advantages of their solution.

The nature of CWMP standard has been contrasted against UPnP in [26] where Nikolaidis et al. produce bridging functionalities of the two protocols for a local (i.e. UPnP) and remote (i.e. CWMP) transparent and scalable management of home devices.

Figure 2-14 shows the main functional blocks for the UPnP-CWMP coexistence. ACS uses the residential gateway as a bridging point between CWMP and UPnP to view the configuration details of the UPnP devices. This is done through adoption of the CWMP data model specification to represent the service and device descriptions. We also point out the two protocol similarities in section 2.3.

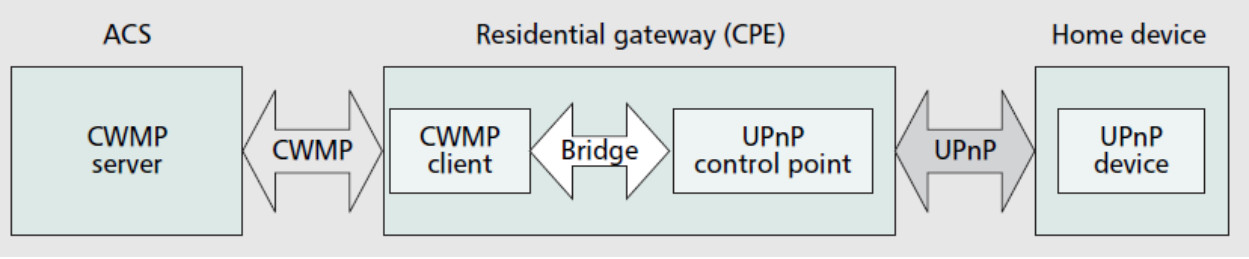

Figure 2-14: The main functional blocks for the UPnP-CWMP coexistence [26].

2.6. Summary

In this chapter we have presented, in a comparative way, the industry most used device management tools and expressed the reasons that makes TR-069 protocol most suitable for our thesis to reach our objectives. We then presented ontologies, as a selected method to model context information, and related work our system ontologies were based on. We also presented some of the available device self-management systems suggested by researchers in industry and academia. Many of the proposed models are extremely general and abstract, thus complicating the process of modeling context information. They also lack convincing implementation and/or testing details to assess system validity and usability. That we overcome in this thesis by presenting a prototype implementation of our proposed system.

Chapter 3

3. Device Self-configuration Framework

3.1. Objective

The core idea of autonomic computing systems is self-management. The purpose is to put less technical pressure on both administrators and end-users of the system by hiding the details of system operation and maintenance, and let the machine do the management incessantly [27]. Hence, self-management is defined as the process by which computer systems shall manage their operations and computations, ideally, with no human intervention. Self-management introduces four functional areas: self-configuration, self-healing, self-optimization, and self-protection. Self-configuration represents an effective support for the other three self-properties - as none of them can happen without reconfiguration of system's operational elements. A self-configurable system has the ability to automatically configure itself, with respect to high-level policies referring to business level objectives, for instance, user service preferences. When a new system component gets introduced in the system, it will automatically gain knowledge of the composition and configuration of the embracing system. It will be registering itself and its capabilities so that other system parties can make use of it.

A device is considered self-configurable if once deployed, it is configured by automatic installation procedures that will allow it to get the basic configuration necessary for its operation. So that during its manoeuvre, the self-configuring device can detect any changes in component parameters that can potentially violate management objectives, through either disruption of the device's normal operation or degradation of its services. Following this, appropriate reconfiguration of parameters should be triggered and new values should be set. Figure 3-1 represents a compilation of self-configuration functions. The functionalities depend on the phase the device is in: pre-operational mode or operational mode. The pre-operational mode is when the device is first powered on - it prepares itself for proper functioning. Therefore, it acquires basic settings relative to the environment it is deployed in

(e.g. residential gateway or related devices); as well as its initial configuration of device or service parameters (e.g. network parameters, initial service related parameters, etc.). After this preparation phase, the device enters the operational mode. During this phase, the device keeps track of key device and service parameters (e.g. performance parameters) for management and control. In case of improper functioning, with regards to what the user is expecting, not to how the device should normally behave, the re-configuration/adaptation is triggered. This process is followed iteratively until the device state problem is solved, and the desired performance is achieved.

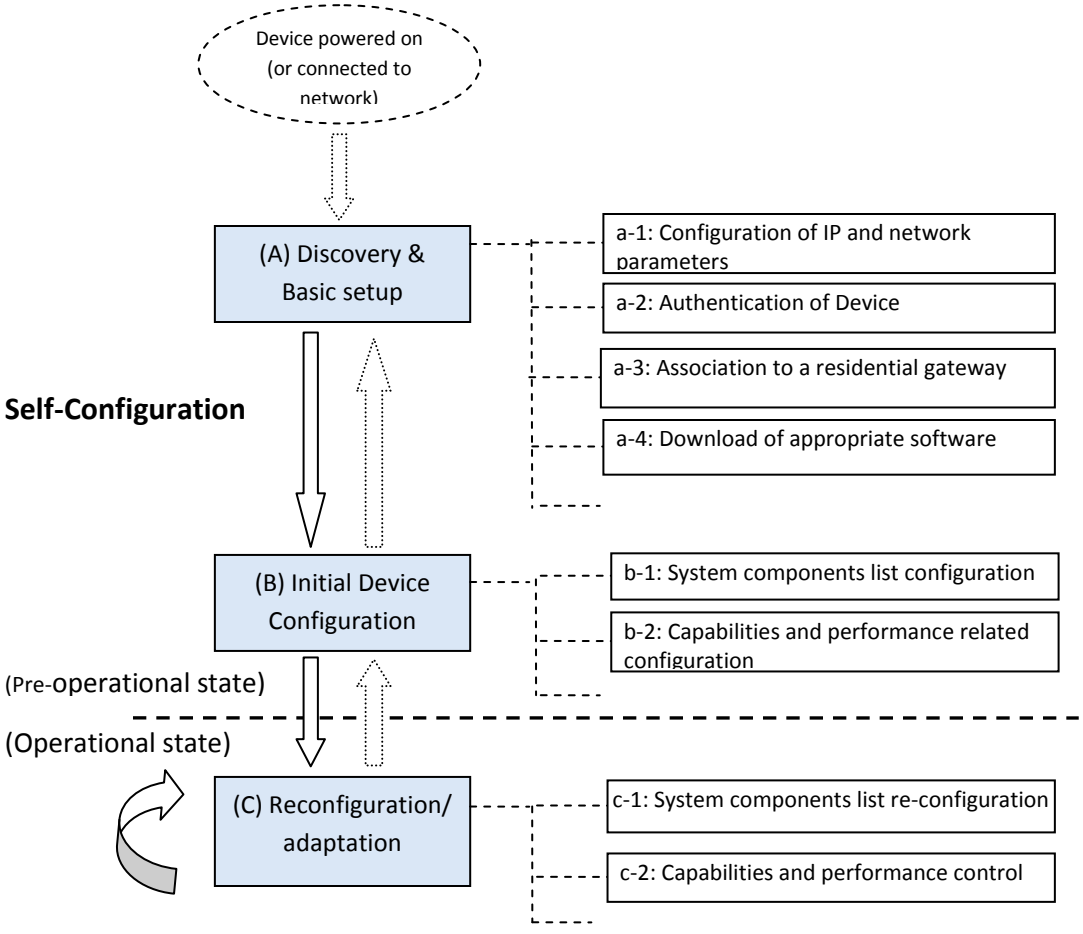

Figure 3-1: Self-configuration process at a glance

3.2. Device Self-configuration challenges

Self-configuration as a concept is broad and its mechanism can differ depending on the context. In this thesis work, we are focused on Home Area Network (HAN) devices self-configuration. Before we start elaborating on our approach, we present hereafter selected challenges that we took into consideration during our work. We explain each challenge and we report how we coped with it in the proposed framework.

- *HAN Device Resource Limitation*: Various digital devices are now available to be deployed and widely used in residential areas. Besides the smart equipments that come with high computational capacities such as computers and laptops, other equipments with limited CPU power, RAM, and flash memory (such as printers, cameras, etc.) also coexist. These types of devices are characterized by their ability to perform one specific task at a time; therefore, their resources are limited. It is necessary for such devices to delegate their management to other parties. Typical solutions have been introduced in the literature to address this matter. Our focus in this work will be regarding digital home devices that have some processing power and available memory storage, for instance, Set-Top-Boxes and PDAs.
- *Software Mobility:* we mean by software mobility the need of an external entity to execute pieces of code on another device that could be placed into the same machine or into another remote machine without pre-installation of the software. There exist two approaches: the first is, forcing software installation through RPC-style functions and executing the necessary procedures. The second is using Mobile Agents with their concept of moving, executing, and returning. In the framework we propose, software mobility can be achieved by RPC invocations. However, our architecture can be further modified to be adapted to the mobile agents.
- *Distributed Management*: Giving the high heterogeneity of Home Area Networks where many devices from different vendors are deployed, with diverse hardware, OS, technologies, etc. a distributed management framework for coordination is of great need. The concept of Web Services is very relevant in this case, as protocol interoperability between different machines is part of our concern. The two most ubiquitous technologies that web services are based upon are the Internet (HTTP) and

XML as a semantic language. Therefore, web services are virtually independent from the vendors' specifications, OS, programming language, etc. Hence, web services can be used to cope with several challenges.

- *Communication protocol:* The operators and HAN device manufacturers do not use a common communication protocol for the management and configuration of the devices. The proposed framework can be adopted by any management communication protocol that has been introduced in the community, but we opted for TR-069 in our implementation thanks to its high relevance.
- *Self-adaptation:* The network management is usually done by operators without the involvement of HAN devices; hence, devices face some lack of self-adaptation in HAN. Through our framework implementation, we deal with this matter by considering also the network parameters as part of the information that should be managed by the management entity.

The remainder of this chapter is dedicated to describing our device self-configuration framework architecture. We describe in detail the core capabilities towards building an autonomic system. We describe how a policy-based approach will fit in the system to provide an autonomous and seamless management of devices as it has been described by H. Harroud [28]. We have chosen an OWL-based ontology conducted by Y. Al Ridhawi [16] to model all the context information within our framework architecture, since it covers most of the required context. We have extended and integrated this ontology within our architecture.

3.3. Global Architecture

We base the framework architecture on the IBM MAPE-K control loop presented earlier in chapter 2. The global architecture shown in Figure 3-2 presents an illustration of the modified control loop architecture as we adapt it for the device self-configuration mechanism in this work. Since the aim is to manage multiple devices in different locations at the same time, the core autonomic components are logically gathered in the *Autonomic Manager* but physically they may not be centralized in one management entity local to the device.

The ontology approach is adopted to capture the knowledge needed for device management. We use policies to define the rules. Devices such as Set-Top-Box, PDAs, and PCs where

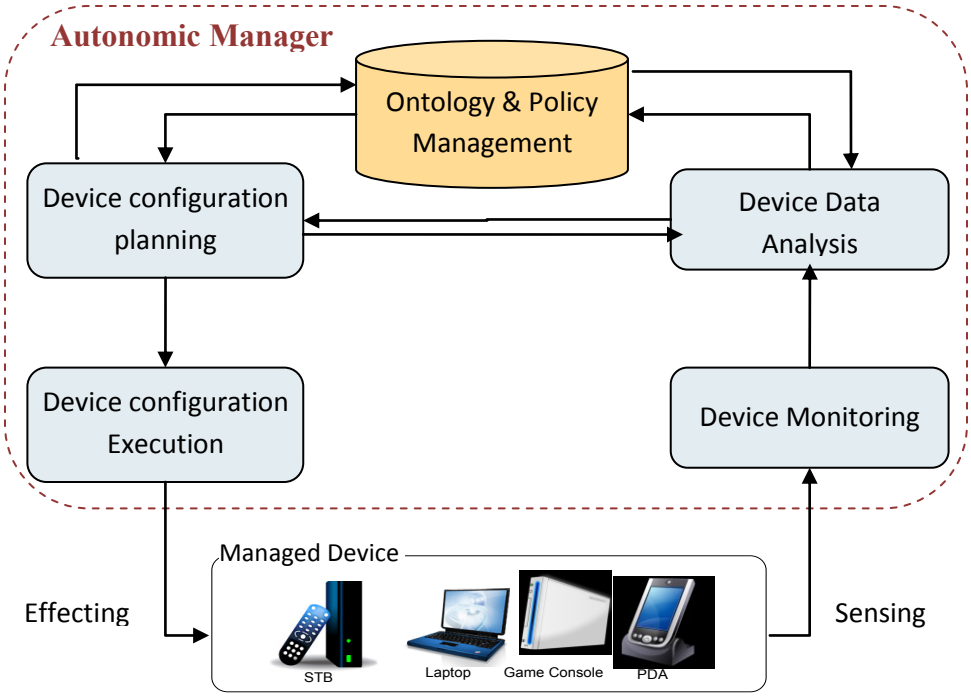

Figure 3-2:Self-configuration control loop

services such as IPTV, audio/video streaming and media abstraction services are deployed can be managed within this system. Each device has an associated *sensor* and *effector*. The former operator is responsible for sensing the status and performance parameters of the device, whereas the latter is accountable for altering the device with the parameters for re-configuration. Sensors and effectors are device/service specific as their actions require a deep

knowledge of data model of the managed resource. In the following subsections, we briefly describe the global architecture components.

3.3.1 Autonomic Manager

The Autonomic Manager (AC) represents the entity that manages the resources. For efficiency and scalability purposes, we are considering this managing entity to be present outside the managed devices. Hence, the management entity will not be restricted to only one user HAN, but it can cover a wider number of HANs concurrently, where thousands of devices are being managed. Likewise, this approach offers the possibility to manage high-end network equipments and devices, as well as fairly limited resources user equipments by keeping the management computation overhead separate from the device.

The AC gathers the core components to offer autonomic capabilities to the system. All the components interoperate to assess the status of and analyze the data received from the device to produce a re-configuration plan if necessary. AC is composed of *Device Monitoring*, *Device data analysis*, *Device Configuration Planning*, *Device Configuration Execution*, and *Ontology and Policies Management*. Hereafter, is a description of each of the components:

- ***Device Monitoring:***

  Device monitoring concerns periodic or on-demand collection of device parameter values to learn about its component and the capabilities it supports (e.g. device profile parameters, hardware components, services deployed). Sensing configuration and performance parameters (e.g. network performance parameters, service quality parameters) allow determining the current state of the managed device to check whether the service and device are achieving the desired performance.

- ***Device Data Analysis:***

  Device data analysis is related to analysis and interpretation of monitored information. As a complementary task to monitoring, the analysis describes the state of the device or services according to the parameter values received. This is achieved using the existing and pre-defined records and rules expressed in terms of ontologies and policies. It makes use of the ontologies to check the conformity and correctness of the parameters and their values with respect to the device and/or service

ontologies, whereas it checks the service quality achieved with the pre-set rules in the policies.

- ***Device Configuration Planning:***

  According to the data analysis, if it has been assessed that the current state of the device is not respecting the desired objectives, the source of the problem is identified and plans for re-configuration are generated accordingly. The configuration plans are generated with regards to the knowledge about the device and its components residing in the ontology base.

- ***Device Configuration Execution:***

  The new parameter values derived from the configuration plans are gathered and put in proper format (i.e. depending on the communication protocol used with the device) and then sent to the managed device to get the re-configuration applied.

- ***Ontology and policies Management:***

  This component contains the description of the current model of the managed device(s) and the corresponding service(s). Ontologies are used to capture knowledge not only for the managed element, but it can be extended to include any relevant piece of information related to the functioning of the device (e.g. existence of other resources, location of the device, user profiles, etc.). Policies correspond to human expertise translated into Event Cause Action (ECA) policies, which enumerates the rules that the user and the operator agreed upon, and that generally specify the user desired and expected service quality.

3.3.2 Managed Device

Managed device is the controlled system component. Managed device represents various user equipments deployed in the home. Devices such as Set-Top-Box, PDAs, and PCs where services such as IPTV, audio/video streaming and media abstraction services are deployed can be managed within this system. Other services can be managed too, but our focus in this work has been concerned with the triple-play services. To make the communication possible between the autonomic manager and the managed device, it is important to set touchpoint

interfaces through which the devices can be controlled. *Sensors* and *Effectors* correspond to the operators that mange this interaction. They are both attached to the managed device.

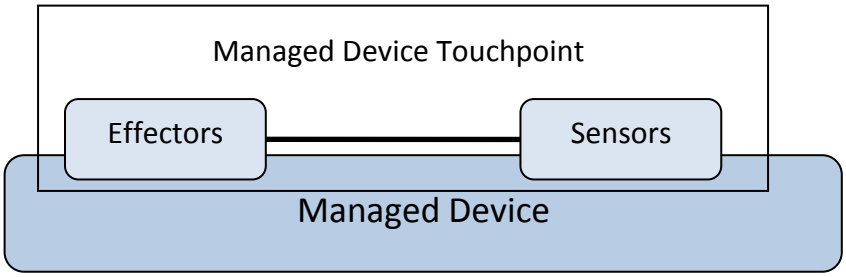

Figure 3-3: Managed Device Effectors and Sensors

- ***Sensors*:**

  Sensors provide an interface for the autonomic manager. These operators are responsible for providing a mechanism of collecting information about the device state and status. The implementation of sensors consists of either a collection of ***Get*** methods to retrieve the device's current state parameters; or a set of management functionalities related to the running of device or service application in terms of value change of state or any significant notification, or both.

- ***Effectors*:**

  Effectors represent the other interface that applies changes sent from the autonomic manager to the managed device (e.g. re-configuration of parameters). The implementation of effectors can consist of a collection of ***Set*** methods to alter the device's configuration parameters.

The combination of both sensors and effectors corresponds to the manageability interface that is made available to the Autonomic Manager; this is what is expressed as *managed device touchpoint* in Figure 3-3 above. So in terms of manageability, sensors represent those operators that retrieve and transmit device state to the autonomic manager. On the other hand, effectors execute the actions on the devices as directed by the autonomic manager.

The link between sensors and effectors in the architecture is relevant because the re-configuration made by effectors on the device can trigger notification of value change to be carried out by the sensor. Communication between autonomic manager and device occurs according to certain interaction approaches depending on how this communication is initiated. Table 3-1 bellow summarizes the sensors and effectors interaction approaches between the autonomic manager and the managed device.

Table 3-1: Sensor and Effector Interaction Approaches

Operator	Interaction Approach
Sensor	Initiated by autonomic manager Retrieves device status
Sensor	Initiated by managed device Receives notice of change
Effector	Initiated by autonomic manager Executes re-configuration
Effector	Initiated by managed device Calls out request

In the following section we present our proposed framework for device self-configuration. We will show how the above control loop maps into our architecture. In-depth investigation of the analysis, planning control loop components is out of the scope of this work; our concern is to show how we can achieve self-configuration of devices within the framework of autonomic computing, regardless of the technical approaches that can be used for these components.

3.4. System Architecture

We designed the framework architecture based on the control loop described in the previous section. The framework functional architecture is shown in Figure 3-4. We separate the management entity, represented by the control and management module, from the managed device.

We present the arrangement of components according to the usual control flow scenario; though it should be thought of "as a common messaging bus rather than a strict control flow", as advised by IBM [4] because of the autonomy property. Hence, communication between these four components should be asynchronous. In other words, in some situations, the device configuration planning component might contact the device monitoring component to collect some specific information that can be useful in making a decision about a new plan.

The management interface and device interface represent the entry point to the control and management component and to the device, respectively. They are responsible for coordinating the communication between the management entity and the managed device, so they need to understand the communication protocol used. The communication protocol can be any of the communication protocols presented in chapter 2. Both interfaces should implement a user-friendly interface that an authority, such as an administrator or a user, is able to view it to get feedback about the device performance or manually change some management parameters.

Services such as the audio/video streaming, conferencing, and media abstraction are usually deployed within devices that offer multimedia services to the user. Each service has an execution environment and a user interface. Also, we keep track of selected service and device parameters in the service/device metadata that can be in the format of properties or configuration files. When the management entity requests parameter information from the devices, it is extracted from these files, and in case of re-configuration, they are affected back into these files. Some parameters can be periodically updated in these files, especially those related to performance.

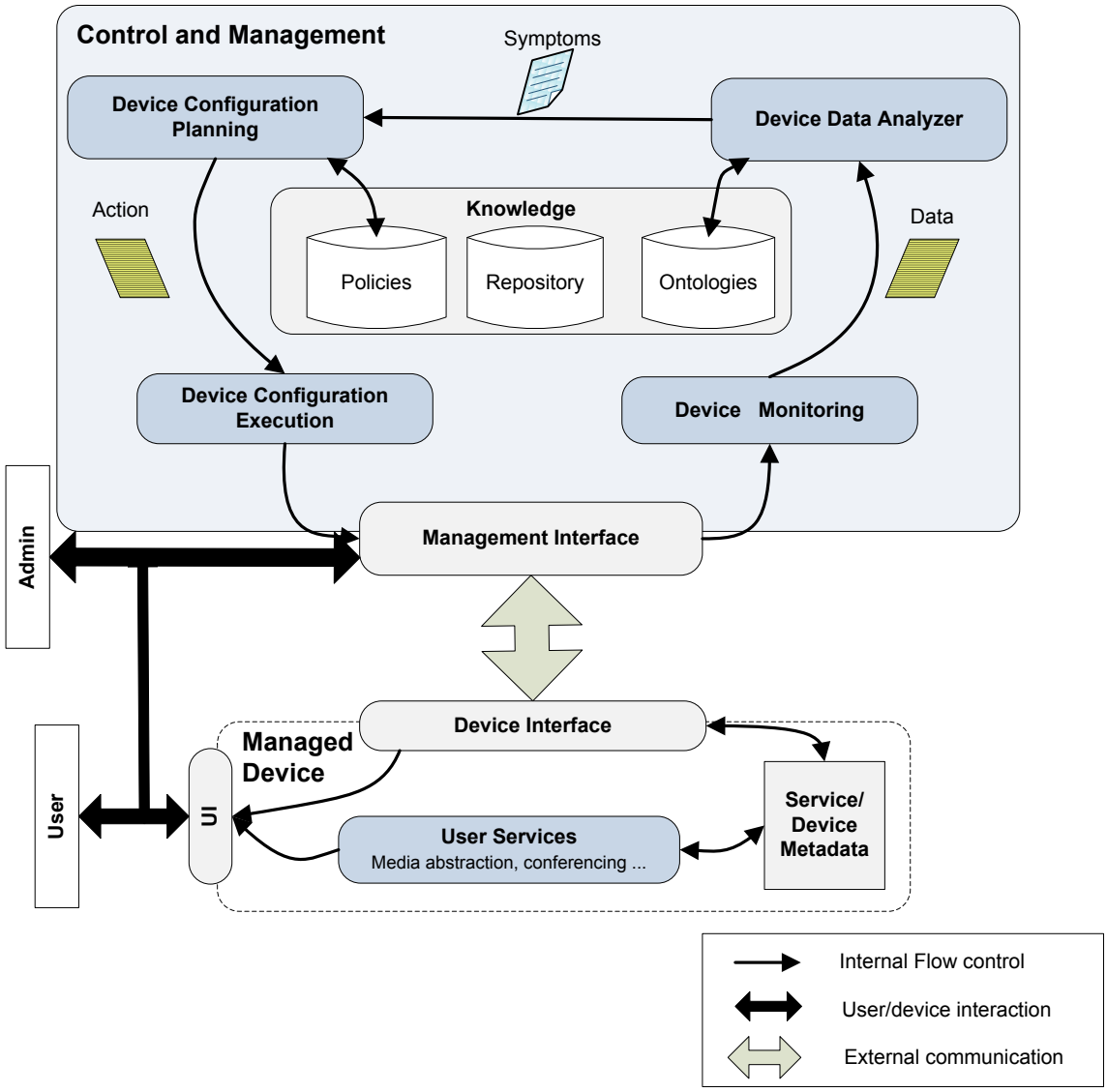

Figure 3-4: Framework Functional Architecture

In the following subsections, we describe in more detail the control and management components; we gather the analysis, planning, and execution in the data processing subsection. We then describe the knowledge component, mainly how the ontology and policy management are applied within this framework.

3.4.1 Device Monitoring

The device monitoring component is the entrance point to the management entity. It has the capability to gather and filter data received through sensors. This component is used by the

management entity to filter the data received, represent it, aggregate it, and forward it to be analyzed. Figure 3-5 demonstrates the monitoring architecture and the functionalities

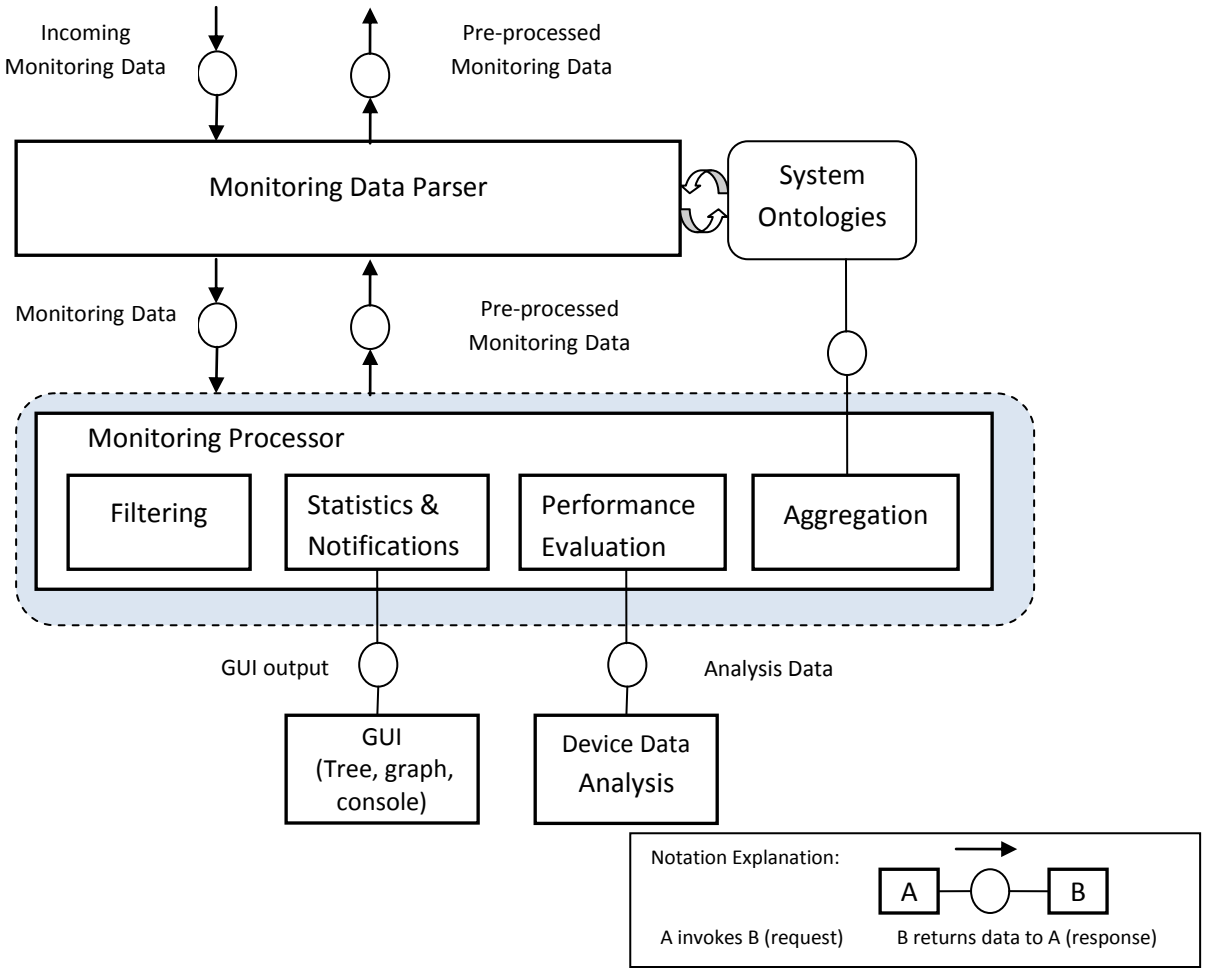

Figure 3-5: Monitoring Architecture

enclosed in it.

Once the monitoring data is retrieved from the incoming messages, it is first parsed and checked for conformity to the ontology. Verified, the data is then passed to be filtered and processed. If the data received represents new information about the device (e.g. new configuration user profile has been added) it gets aggregated and stored in the repository or added to the ontology. In case the data represents some statistic parameters that should be transparent and shown to an authority (system administrator or customer) in a user-friendly way, it is then passed as GUI output where appropriate graphs and trees are plotted. When

the data represent performance parameters that the system has to evaluate, they are forwarded to the data analysis component.

Different monitoring approaches have been introduced in the literature [29]. We adopt both an active and passive monitoring approach.

- *Passive Monitoring*: the management entity is organized to receive, passively, messages summarizing the current state of the device. These messages are either based on a time factor (i.e. both entities agree on a periodic time interval to send messages), or upon a change of value of some parameters that we first configure their notification as being active.
- *Active monitoring*: monitoring is actively triggered to reinforce data analysis. The monitoring module gets a feedback from the analysis module to request a parameter value from the device to complete the problem identification and target the re-configuration planning.

Active and passive monitoring can collaborate together to achieve an accurate device re-configurability. Let's consider the following scenario for example: An end user is streaming some media when degradation in the video quality resulting in QoE issue can trigger a value change event upon a significant change in the visibility indicator; a message is then sent to the management entity (passive monitoring). The monitoring module filters the data and forwards the concerned parameters to the analysis modules. While analyzing the source of the problem, the analysis module might request additional parameter values related to QoS, for instance, or any other potential candidate parameters (active monitoring).

It is important to note that the monitoring module is not responsible for coordinating the communication between the autonomic manager and the managed device. Instead, it is the management interface component representing the point of entrance to the management entity that takes care of this task.

3.4.2 Device Data Analysis, Configuration Planning, and Execution

In this section we explain the steps that follow the monitoring phase of pertinent device performance data. The level of intelligence in an autonomic control loop depends on this

processing. The autonomic manager should perform complex data analysis and reasoning, leading to an accurate decision planning for reconfiguration. In data processing we gather the three modules: *Device Data Analysis*, *Device Configuration Planner*, and *Device Configuration Execution*.

Figure 3-6 shows the general algorithm that describes the control flow we present in our architecture. The autonomic manager continuously checks for any changes within the device. In case there was a modification in the management rules, policies are updated (steps 2~4). The Device Monitoring component monitors device parameters received through the sensor, and passes the performance parameters representing the current state of the device to the analysis module (step 5). This information is analyzed against the desired state provided in the system rules available by the Device Data Analysis component (step 6). If the symptoms generated (e.g. metric records) following the analysis depicts any state deviation or service violation, Device Configuration Planning plans new actions to bring the device back to its

```
1: While (true)
2: If (any modifications in the device or service
     management rules)
3:    read and update policies
4: End if
5: state ← DeviceMonitoring (DeviceParameters)
6: symptoms ← DeviceDataAnalysis(state, DesiredStateRules)
7: If (sympotoms correspond to state deviation or service
violation)
8:   action = DeviceConfigurationPlanning(sysmptoms, policies)
9:   DeviceConfigurationExecution(action)
10: End if
11: End While
```

Figure 3-6: Autonomic Manager General Algorithm

expected performance (step 7~8). An action is a method invocation which is related to a specific entity. The action is then passed to the Device Configuration Execution module to carry out the execution.

Below, we describe each module by referring to techniques introduced in the literature; however, deep investigation of these modules is beyond the scope of this work, as they are more relevant for other functions of autonomic computing such as self-healing and self-protection.

Device Data Analysis:

As the monitoring architecture shows, not all monitored data are transmitted to the analysis module. Only performance-related information is transferred to analysis for evaluation. There exist different approaches for analysis, depending on the intelligence degree of the management system.

Basic management systems refer to analysis as "diagnosis". Diagnosis is the operation by which the current state of the device is being interpreted against pre-defined rules or heuristics [29].

Contrary to traditional diagnosis that is based on existing rules to describe the state of the device, autonomic diagnosis, deployed in autonomic management system, rely on self-learning abilities to enrich the knowledge base with new states (detected in instances of devices anomalies) to improve the analysis mechanism. Several techniques have been adopted in the literature to achieve self-learning such as complex probability models and Bayesian methods introduced in [30], where Zhang et al. use these techniques to analyze and identify the problems causing violations of service-level objectives, along with the measurement of low-level system properties contributing to such noncompliance.

The main objective behind self-learning is state anticipation; this notion has been introduced in the context of autonomic network management. This said, more generic autonomic anticipation approaches have seen light in recent times. For example, Songyun Duan and Shivnath Babu [31] propose an automated tool for accurate prediction of Service-Level-Agreement (SLA) violations caused by performance issues. They make use of probabilistic

models, such as a Bayesian network, to analyze the collected performance data (i.e. at the application level, database server level, and operating system level) to predict possible future violations of the service.

Such self-learning and anticipation techniques are of potential relevance to the framework we are proposing in this work. In-depth study of this component has not been carried out within the scope of this thesis. It can be regarded for further research directions.

Device Configuration Planning and Execution:

The two remaining functions of the control and management in our architecture are device configuration planning and execution. The planning function provides mechanisms of configuration for the device operations and services. It makes use of policies to draw the required plan actions to achieve the configuration objectives. The execution function controls the implementation of the configuration taking into consideration on-the-fly updates.

The planning paradigm has been explored in various autonomic management systems using different approaches. In [32], the authors' AI planning mechanism is based on generating the best final states according to the desired goals. A path from the current state to the final state is then found, and related actions are performed for execution.

Autonomic adaptation reflects best the objective of the planning and execution functionalities as we view it in our system. By "adaptation" we mean the process of applying those changes to the system that brings it back to a state that corresponds to the present system goals and objectives. Autonomic adaptation makes use of the system knowledge to provide the ability of system adaptation without external intervention. A classification of adaptation dimensions has been surveyed in [29]. Most common systems that go towards achieving autonomic adaptation are rooted in approaches that are policy-based and/or goal-based. These systems use a multitude of complex heuristic algorithms and probabilistic methods to enforce policies and generate new ones [29].

Both policy-based and goal-based approaches significantly fit within our device self-configuration framework, where the configuration rules (related to both device operation and

service requirements) can be expressed in terms of policies. This will allow continuous fine tuning of the user devices, and provide a better experience for users.

In the following section, we describe how policies are represented within our system. However, technical details on policy enforcement and management that determine how policies are evaluated, triggered, and generated are beyond the scope of this research. It is being carried out in a separate project within our research laboratory.

3.4.3 Knowledge: Policies and Ontologies Management

Device configuration is based on numerous parameters constituting the device organization, reflecting the state of the equipment with regard to other variables and factors, and revealing the performance conditions it is going through. All these millions of parameters are to be accurately managed by the control and management entity. Hence, there is an obvious need to have all this information well represented and organized to make it simple for the components to exploit. Raw data is not the only information we would like to keep track of in such systems. To make the devices self-configurable there should be some defined rules to govern the configuration and re-configuration logic. These rules are expressed in terms of policies.

The data used by the four components of the management entity are stored in a shared 'knowledge' base. In this architecture we divide this knowledge into two main components: ontologies and policies. Ontologies, on the one hand, can store information such as device and service data-models, system log, performance metrics, etc. Policies, on the other hand, define the rules that govern the self-configuration planning and triggers device re-configuration.

In the following sections, we describe how policies can be used in the system to enhance the knowledge processing about user devices and running services. In-depth technical details of how policies are to be implemented are beyond the scope of this work as it is another challenging topic that should be considered for future work.

We also explain how ontology management is applied in this system. An ontology-based management system has been already developed within the Mobile Agent and Multimedia

Information research laboratory; we adapt and expand the model to respond to our project requirements.

3.4.3.1 Policies Representation for device self-configuration

Policy-based management provides a mechanism to allocate different types of resources according to defined policies. Network resources, such as network bandwidth, quality of service, and security are well-known examples of resources to be managed by policies, For instance, as the requirement for QoS increases with the use of an IPTV service or other real-time applications, the requirement for policy-based bandwidth allocation increases.

A policy-based management system allows an authority (the system administrator) to define rules and manage them in the policy system. Rules usually take the form of "*if* <*condition*> *then* <*action*>." A condition might be related to a particular user or group of users, a specific time of day, a type of service, a specific network access point, or a combination of specific events. Events are triggered either by a time-period condition (periodic monitoring), a change in the state of a resource or as a result of an action. An action is a method invocation which is related to a specific entity. An example of policy structure is giving in Figure 3-7.

```
Policy ServiceAViolation (paramterX)
  If paramterX < ValueX
  Then performTest1  test parameterY < ValueY
       PerformTest2  test parameterZ < ValueZ
       …
```

Figure 3-7: Sample structure of policy expectation

The introduction of policies inside the autonomic manager provides it with more pro-activeness in facing new situations at managed resources, with service and devices status and performance management. Another advantage deriving from the policy-based approach is the increased adaptability of the management entity from one user profile to another, as every user or specific device can have different policies.

It is required to have a clear *policy model* that specifies how policies are expressed and represented. Technical details on *enforcement model* that determine how policies are evaluated and triggered are provided in a separate project within the Mobile Agent and

Multimedia Information laboratory. In the following, we describe how policies can be represented and illustrate with examples, we adopt the presentation model developed by H. Harroud [28].

A policy is a set of actions to be performed by an effector on one or more target devices providing some conditions are satisfied or some events are triggered. A subject is the entity responsible for executing one or more actions of a policy. Targets are entities on which an action is performed. The conditions typically represent the state of one or more entities in the system (Figure 3-8). Events are triggered either by a time-period condition, a change in the state of a device or as a result of an action (i.e. before/after events).

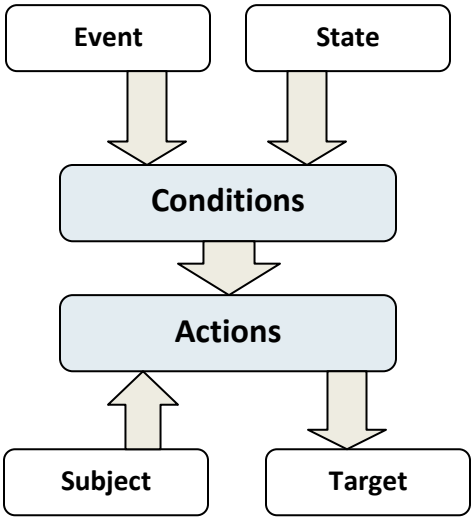

Figure 3-8: Definition of a caption [28]

A policy can be either an **obligation policy** that acts as the trigger for actions to be performed when conditions are applicable or an **authorization policy** that permits or prohibits actions on target entities.

Policies are applied to several levels of the system including admission control, user profile, and resource management. Examples of these policy levels are shown in Table 3-2 and Table 3-3.

Table 3-2: Sample of authorization policies for devices

Policy ID	Subject	Action	Target	Conditions	Priority
P-1	Managed Device	Service Access	3rd party service	qos<v.s	High
P-2	Managed Device	Session Initiation	Autonomic Manager	All Devices	High
P-3	Managed Device	Encryption	Monitoring	Key-len >64	High

P-1 authorization policy (permission). It indicates that a particular user is authorized to use a particular service (specified in the service access action) if the quality of service will not include video streaming.

P-2 is an example of admission control policies that specify the devices authorized to initiate a session with the autonomic manager.

P-3 allows the managed device to request encryption action when communicating with the autonomic manager (i.e. monitoring) using an encryption key length of more than 64 bits.

At run-tine, a conflict between two or more policies may occur. The resolution of this conflict is based on the use of *Priority*. The priority element defines the importance of the policy to be applied, so that the management entity can establish a sorting of policies to know what policy has higher priority to be applied. The policy that has higher priority will apply first. Different ways can be used to represent the priority value. Every value corresponds to a weight that is part of an evaluation function to determine what policy to be picked. For example the priority values can be defined as follows:

Low policy execution is optional.

High policy execution is necessary, for example, to avoid a service disconnection.

Table 3-3: Sample of obligation policies

Policy ID	Type	Trigger	Subject	Action	Target	Conditions
P-5	OP	Active Monitoring	Analysis	Notify ("critical value change")	Planning	Bandwidth <8
P-6	OP	Passive Monitoring	Analysis	Notify ("normal device status")	Planning	Bandwidth >10

P-5 and P-6 are obligation policies that apply to the autonomic manager, in particular the analysis component. The analysis component has to notify the planning component if there is any change in the managed device status according to the trigger.

We represent policies using the Resource Description Framework (RDF) [33]. RDF is one way of representing information understood by other RDF speaking communities, RDF follows the XML representation so that it is system independent and understandable. RDF represents a resource uniquely identified by a URI (Uniform Resource Identifier). Each of these resources or objects has a set of properties or attributes. A collection of these properties

```
<policy: Name rdf:about = "Policy Name">Access Restriction</policy:Name>
<policy: PolicyRule rdf:about= "Policy Rule">
   <policy:Enabled rdf:about= "Policy enabled">Yes</policy:Enabled>
   <policy:Trigger rdf:about=>Initialize Session</policy:Trigger>
   <policy:Subject rdf:about=>Autonomic Manager</policy:Subject>
   <policy:Target rdf:about=>Managed Resource</policy:Target>
   <policy:Condition rdf:about= "Policy Condition">[media = video; audio; data] AND
                                                  [device=PDA]</policy:Condition>
   <policy:Priority rdf:about= "Policy Priority">High</policy:Priority>
   <policy:Action rdf:about= "Policy Action">Rejected</policy:Action>
   <policy:Audit rdf:about= "Policy Audit">Yes</policy:Audit>
</policy: PolicyRule>
```

Figure 3-9: A sample of access restriction policy

makes up the description of the object.

Figure 3-9 shows an example of a policy in RDF format that specifies that a user device is participating in a session initialization via a PDA that can use audio, video, or data while there might be restrictions to some other type of devices. If the policy audit is enabled (i.e. Yes in the policy Audit entry), the information about policy enforcement is stored, and could be analyzed by authorized authority when needed.

RDF representation is also used to describe the capabilities of a device or a service together with the preferences of the user for content adaptation using CC/PP (Composite Capabilities/Preference Profiles). CC/PP [34] can be thought of as a collection of RDF statements that describes the services and users. It is stored in the form of tables each table is a collection of RDF statements.

The main purpose of the CC/PP is negotiating content between two parties to facilitate the identification of a service. The RDF data in the CC/PP profile may originate from various sources and merged together into a single CC/PP profile. The description of the device may be from the device manufacturer and the preferences from the user assistant. The analysis component compares this document with the stored documents to find the best service for the user. Figure 3-10 shows a sample service profile that represents a printer.

```
<xml version="1">
<Document>
<Type value="Service"/>
<Device>
        <Device_group value="PRINTERS">
        <Device_name value="HP LaserJet 4050 N PS">
</Device>
<Properties>
        <Color value="no">
        <sided value="yes">
        <Memory>
                <minMemory="8"/>
                <maxMemory="11"/>
        </Memory>
        <Resolution value="1200" />
        <PS value="true" />
</Properties>
<EnteringTime value=""/>
<ExitTime value="" />
</Document>
```

Figure 3-10: A sample of Printer Profile

A user profile similarly contains basic information such as Name, Address, Home number, Designation, Location, etc. and some more parameters such as the status of the user, the services belonging to this user and security parameters of this user. It is important to note that the scope of the study of policies management in this work is mainly to prove how policies are relevant in such systems, how they can be defined, and to present a concrete and semantic way of representation that can to be stored in our knowledge base. The process of interpretation and enforcement of policies will be examined in a separate project within the research laboratory.

3.4.3.2 Ontologies Management

Ontology-based management provides a mechanism to represent the knowledge following a predefined and structured specification of concepts. Ontologies have been widely used to

model the context. Particularly, by context we mean all information (i.e. facts and circumstances) that describe the environment a device is in. This information should include also details about the device itself, its component, the services deployed in it, the network connectivity, communication bandwidth, etc.

An ontology-based management system allows an authority (The system administrator) to represent concepts and relationships by employing a computer-usable data structure while sharing a common understanding of the domains in which the autonomic system has interest. This ontology knowledge is usually represented through a set of entities, functions, instances and axioms. Ontologies are also known for their normalization abilities and formality, making them a favorable candidate for modeling knowledge. Another advantage deriving from the use of ontologies is knowledge sharing and reuse. Therefore ontologies suit our needs with of modeling and representing the knowledge related to resources we would like to manage.

Ontology is a "formal, explicit specification of a shared conceptualization" [35]. We use ontologies to capture the structure of a context or a domain, i.e. conceptualization. This conceptualization represents a description of the all knowledge necessary to learn about the domain. This means that this conceptualization is not affected by changes frequently.

To be able to process ontologies automatically, a formal specification of ontologies is required. In other words, it is required to have a clear *ontology model* that specifies how ontologies are expressed and represented (as previously done for policies).

Most proposed ontology-based models have adopted the emerging Ontology Web Language OWL [15] as a means of ontology representation. This is due to OWL's superior expressive abilities over other languages, such as XML, RDF, RDFS, or DAML+OIL. OWL also allows the exchange and comprehension of context information between different entities within context-aware systems. The availability of a number of OWL-based reasoning engines that can interpret information present within the ontology, or that can infer new knowledge from existing information and relationships, makes OWL an especially effective language for context modeling. We have chosen an OWL-based ontology conducted by Y.

Al Ridhawi [16] and extended by I. Al Ridhawi [17] to model our context information within our system since it covers most of the required information.

In COMANTO [18], and CONON [19], ontologies were divided into an abstract high-level ontology and a lower domain-specific ontology. Using this concept of domain separation, the Ontology presented in [16] can be seen as our modified version of the Upper or High-Level Ontology presented in both of these works. In our work, we expand the domain-specific ontology as we relate it to the environment of an autonomic system. Ontologies for environment's concepts modeling were presented in [20]. Their ontologies are of low-level domain specific. There logical representation has four main components: policy, network, service, and user. Other components such us device, location, and time do exist too, but not given as much importance. For our environment conceptualization all these components are of important relevance.

Many of the classes that existed within [16], [17], [18], [19], and [20] were imported, some were expanded; their hierarchical organization was reordered to allow for easier extensions within the low-level domain-specific ontologies, and an extensive list of inter-class relationships and properties was defined to ease the development of domain-specific ontologies.

Figure 3-11, shows a hierarchical class organization within our ontology. We will express the class's details in the following subsections, including object properties (i.e. relationships between classes) and data type properties.

The *Ontology* class represents the root from which all environment context classes and properties stem. It provides an entry point for declaring all domain-specific ontologies. Four major classes extend from the *Ontology* class, each representing a major concept needed within the environment of autonomic systems:

- *Location*
- *Object*

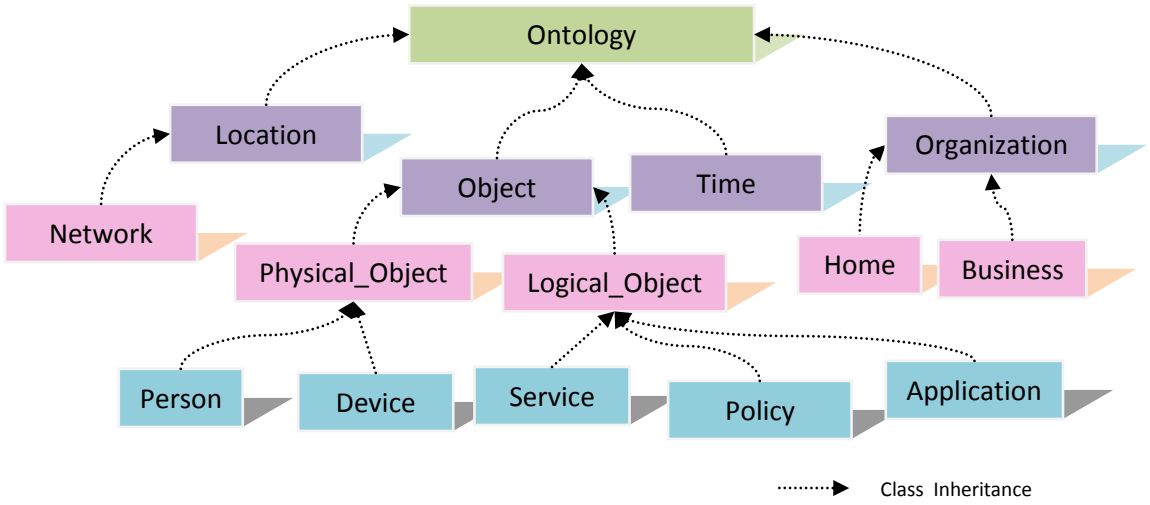

Figure 3-11: Hierarchical class organization of Ontology

- *Time*
- *Organization*

Each class of the upper-level ontology is further divided into subclasses. For example, the Object class is divided into *Physical_Object*, and *Logical_Object*, and *Date*. The following subsections provide details on the domain-specific lasses. We start with the object ontology that contains the core classes related to our self-configuration framework i.e. Device and Service.

3.3.7.1 Object Ontology

Since the self-configuration framework is aimed at providing re-configurability and auto-adaptation to devices and services utilized by an end user (i.e. the computing device carried by user), the presence of an ontology on which to model users, devices, and services is, therefore, highly important. In addition, the use of policies to govern the rules of devices and services' management adds a proactive mechanism in control and management of the end user equipments. To take into consideration the wide variety of context available to be modeled around users, services and devices, we created a top-level domain class called "*Object.*" This class models anything that can be described as an object, such as a PDA, home domain user, policy, residential gateway, and device components. The *Object* class is divided into the following subclasses: *PhysicalObject*, and *LogicalObject*. Details are shown in Figure 3-12.

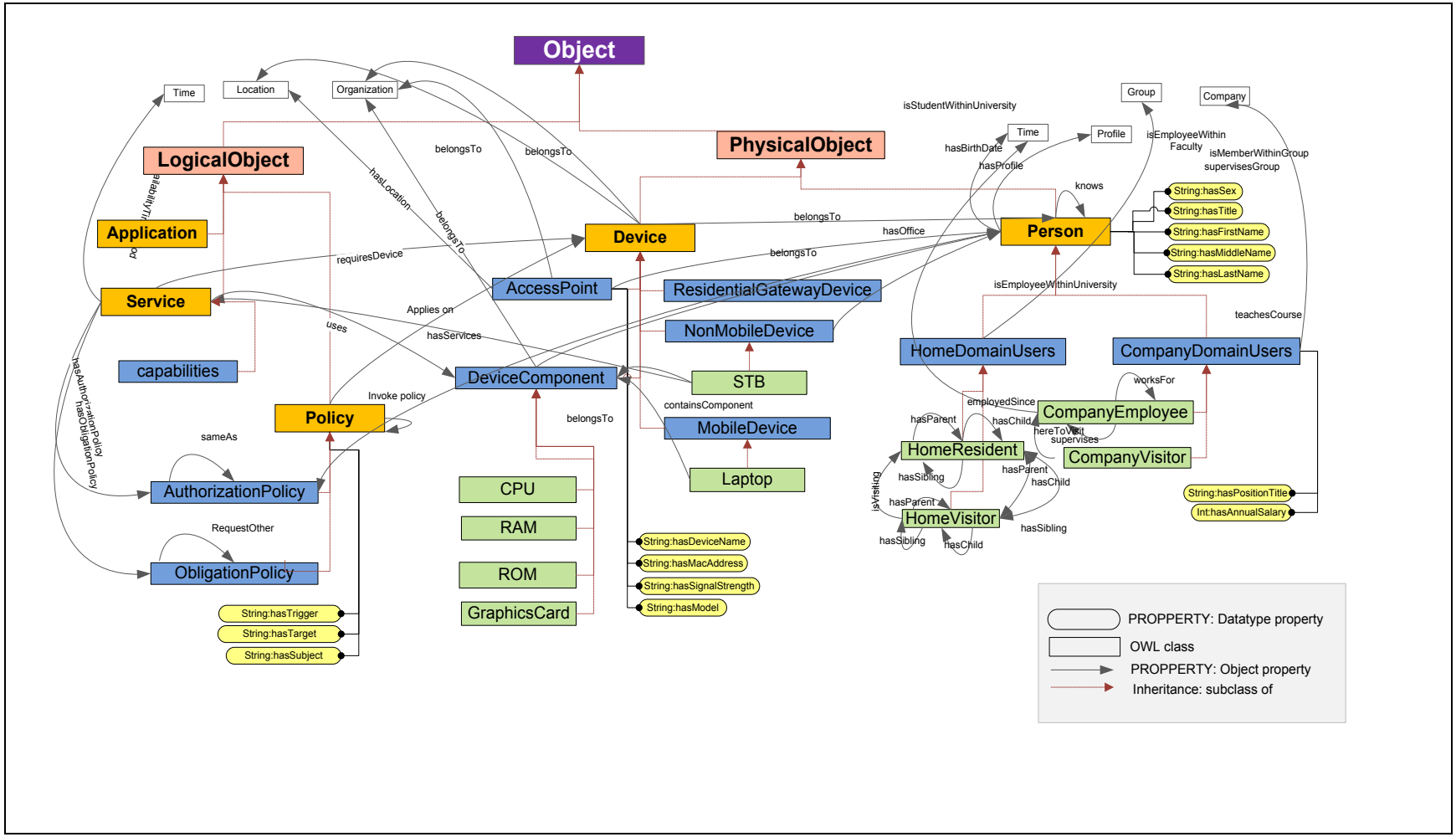

Figure 3-12: Object Ontology

The *PhysicalObject* subclass is a complex class due to the wide variety of objects being classified as physical. Therefore, we divided this class into two more subclasses: *Person* and *Device*.

The process of modeling all possible users in all location-aware systems is a difficult task. For this reason, we have restricted our ontology to a set of limited OWL classes and properties that model the general concept of the *Person* class. The appropriate way of modeling users' information is according to the roles these users play within different domains. The needs and interests of users tend to change according to their role. As an example, for a home domain user the information or services to be used are related to the home domain, and permissions in accessing them depends on the role the user plays in the family i.e. parents are usually permitted to access and manipulate information more than their children or even a guest. The role of the same person within a work domain i.e. in a company, within the family is irrelevant in this context. Rather, the system is more concerned with in identifying this person within the company or institution structure (e.g. colleagues, departments, employers, etc.) Hence, a person in a company has a different role than a person at home. Therefore, *Person* class is split into two main domains: *HomeDomainUsers*, and *CompanyDomainUsers*. Although our conceptualization might require other domains, we are interested in providing a subset of only those which are most widely used and relevant for our work.

There exists a set of context properties that all users share regardless of which domain they belong to. All Individuals could be divided into the categories male and female (*String:hasSex*); all human users have first and last names (*String:hasFirstName, String:hasLastName*), and all have a date of birth (*hasBirthDate*). Users could have many other possible common properties.

Most people have relationships with other individuals; we entitle these relationships in our ontology as *knows*. Many classes have property relationships with different parts of the ontology, which is very helpful to us because it reduces the time spent creating ontology models from scratch. For instance, *hasBirthDate* links the *Person* class to the *Time* ontology.

The *HomeDomainUsers* subclass is used to model human users within the home domain. The needs, interests, and roles of home residents differ. Parents' context interests and accessibility rights are wider than their children, whose access permission may be limited. Also visitors are usually only permitted access to a limited amount of context information. Therefore, two subclasses are needed: *HomeResident* and *HomeVisitor*, and only four object properties are required: *hasParent*, *hasChild*, *hasSibling*, and *isVisiting*. All other context information can be inferred from these classes and properties. The home users can be clustered in groups, represented by the external class *Group*, and every group will have shared properties in terms of permissions to access the information.

The same concept can be interoperated to *CompanyDomainUsers* subclass. For simplicity reasons, we limit the users within the company domain to two major users *CompanyEmployee* and *ComoanyVisitor*. As it has been the case for the home users, company employees have more privilege permission and wider access to context information related to the company. A company visitor should be in a visit to some company employee; this is expressed by the relationship *hereToVisit*. Permissions within the company employees will differ according to their roles and responsibilities. A company employee is usually supervised by another employee ; this is represented by the relationship *supervised*. Every employee has a *hasPositionTitle* and *hasAnnualsalary*. Another link between *CompanyEmployee* subclass and *Time* class is *EmploedSince*, as a reference to the date the employee was employed in. The external class *Company* is the class that encloses information about the company and that it is defined within the *Organization* ontology.

Another class that inherits from *PhysicalObject* is the class *Device*. As we are interested in the self-configuration of devices, it is important to model this class as detailed as possible to promote semantics, knowledge sharing and reuse, as different devices might exist with a comparable system architecture structure. Still, there should be some flexibility in representing the devices data model, so that it can allow adding components that are vendor specific.

The process of modeling all possible devices that we might deal with in a self-configuration framework is a difficult task. For this reason, we have restricted our ontology to a set of limited OWL classes and properties that model the general concept of the *Device* class

according to our needs. We categorize the device subclasses according to their types: *MobileDevice*, or *NonMobileDevice.* Devices such as laptops, PDAs, smart phones, etc. are grouped under the category of *MobileDevice*. Mobile devices share common properties, especially those related to constraints regarding performance issues. It is true that performance issues are the concern of any real-world platform design, but generally mobile devices are subject to limitations when it comes to resources; mostly processing power, and memory. Another issue would be network connectivity that might be considerably slower than the best-case scenario, or that might even get irregular. Other fixed devices such as PCs, STBs, printers, etc. fall in the *NonMobileDevice* category. These two categorises can still get broken down further more according to the device resources. We limit ourselves to this level of categorization as this responds to our needs for this work.

Another criterion in device classification was related to the device role. We have placed in different classes as the other device classes: the *AcessPoint* which might refer to a router, switch, etc. and *ResidentialGatewayDevice* which refers to the home gateway. And finally, *DeviceComponent*, which classifies objects such as CPUs, sound cards, video cards, etc. The devices belong to either organizations or persons, and they are being used from some location; therefore, relationships are required in this case.

The device has a *belongTo* relationship with the class *Location* that describes the location where the device is being used from. Among the address types that have been described for the geographical position of the device is the network address. A network address serves as a unique identifier for a computing device on a network. These addresses are used for identity and communication purposes. One of the best known forms of network addressing is the Internet Protocol (IP) address. IP addresses consist of four bytes (32 bits) that uniquely identify all computing devices. Another popular form of address is the Media Access Control (MAC) address. MAC addresses are six bytes (48 bits) that manufacturers of network adapters burn into their products to uniquely identify them. These concepts have been adopted and grouped in the *NetworkAddress* class. Since each device that has an IP or MAC address is also located within a corresponding physical location, such as a server room, we have provided a *hasGeographicalPosition* relationship that permits a network address to

relate to other types of location whether they are *OutdoorSpace* or *IndoorSpace*. The complete ontology description can be found in [16].

As part of the knowledge and rules governance for self-configuring devices, policies are introduced to the control and management of devices. We represent policies in the *Policy* class that has an *AppliedTo* relationship with the device. *InvokePolicy* denotes the relationship of a policy invocation to another policy. The policies are categorized in the ontology as it has been described in the previous section, when we defined two types of policies: authorization policy and obligation policy represented respectively by the classes *AuthorizationPolicy* and *ObligationPolicy*. As described in the previous section, each policy defines a *Target*, *Triggers*, and *Subject*.

Devices deploy services and hence we have *hasService* relationship with class *Service*. The *Service* class represent services being offered to the user (e.g. IPTV, video streaming, etc.) All or some of the service *Capabilities* (i.e. referring to service functionalities) can be available depending on the user preferences, permission or even the device components. Service availability information is kept up using the *Time* class. Services have preliminary defined policies represented by the relationships *hasAuthorizationPolicy*, for permission or prohibition of actions, and *hasObligationPolicy*, acting as a trigger for actions to be performed. Figure **3-13** and Figure **3-14** represent a downsized view of the ontology for an STB device. We developed this ontology based on the STB data model proposed by the Broadband forum (BBF) [36] published in TR-135 [37]. BBF presents the minimum required data objects needed for devices to integrate their TR-69 protocol family. We use OWL [15] while taking advantage of its features of lucidity and explicitly (i.e. O*bjectProperty*, *dataTypeProperty*) to represent these objects with more semantics for a better expressiveness and machine readability.

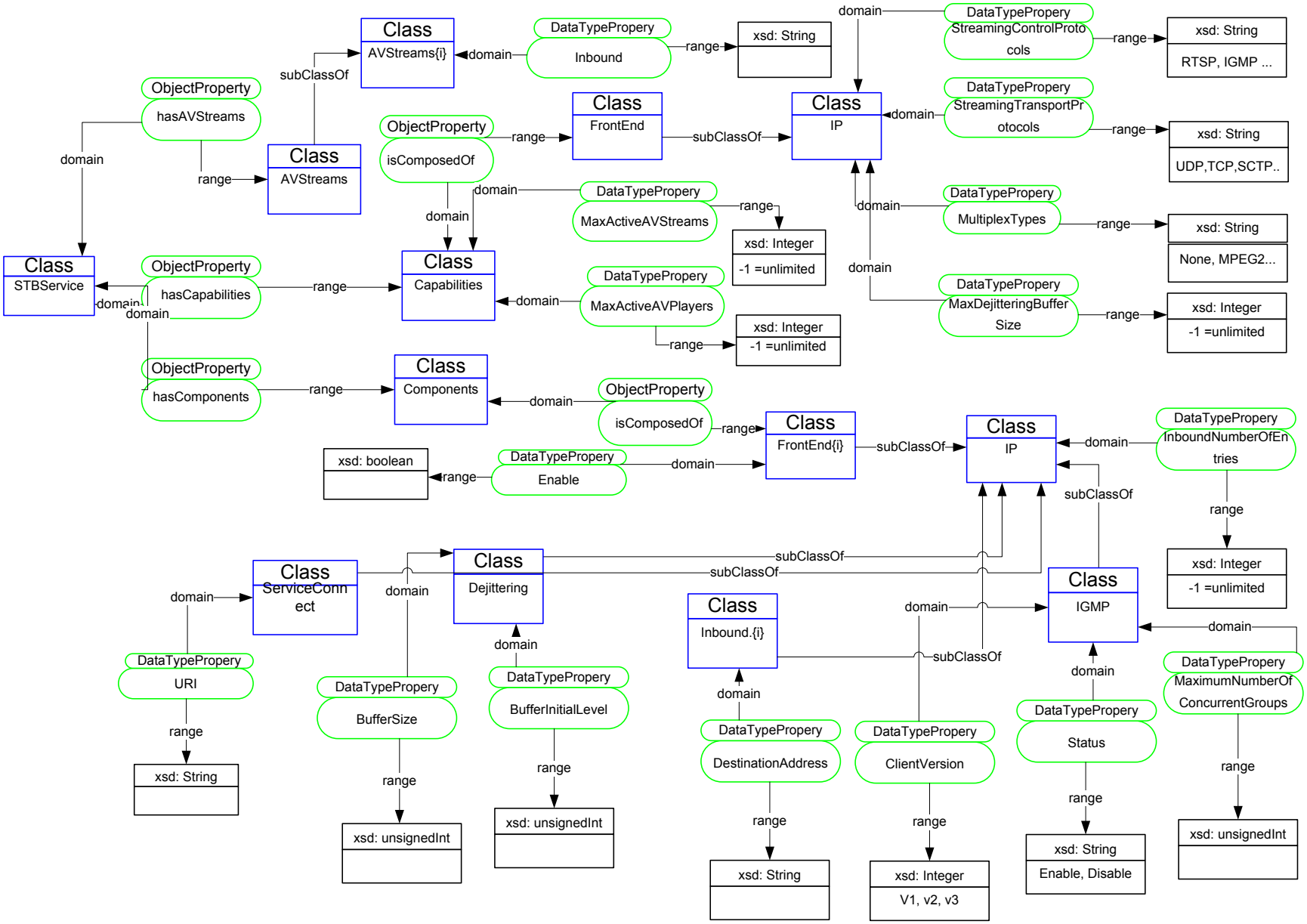

Figure 3-13: STB Ontology using OWL-

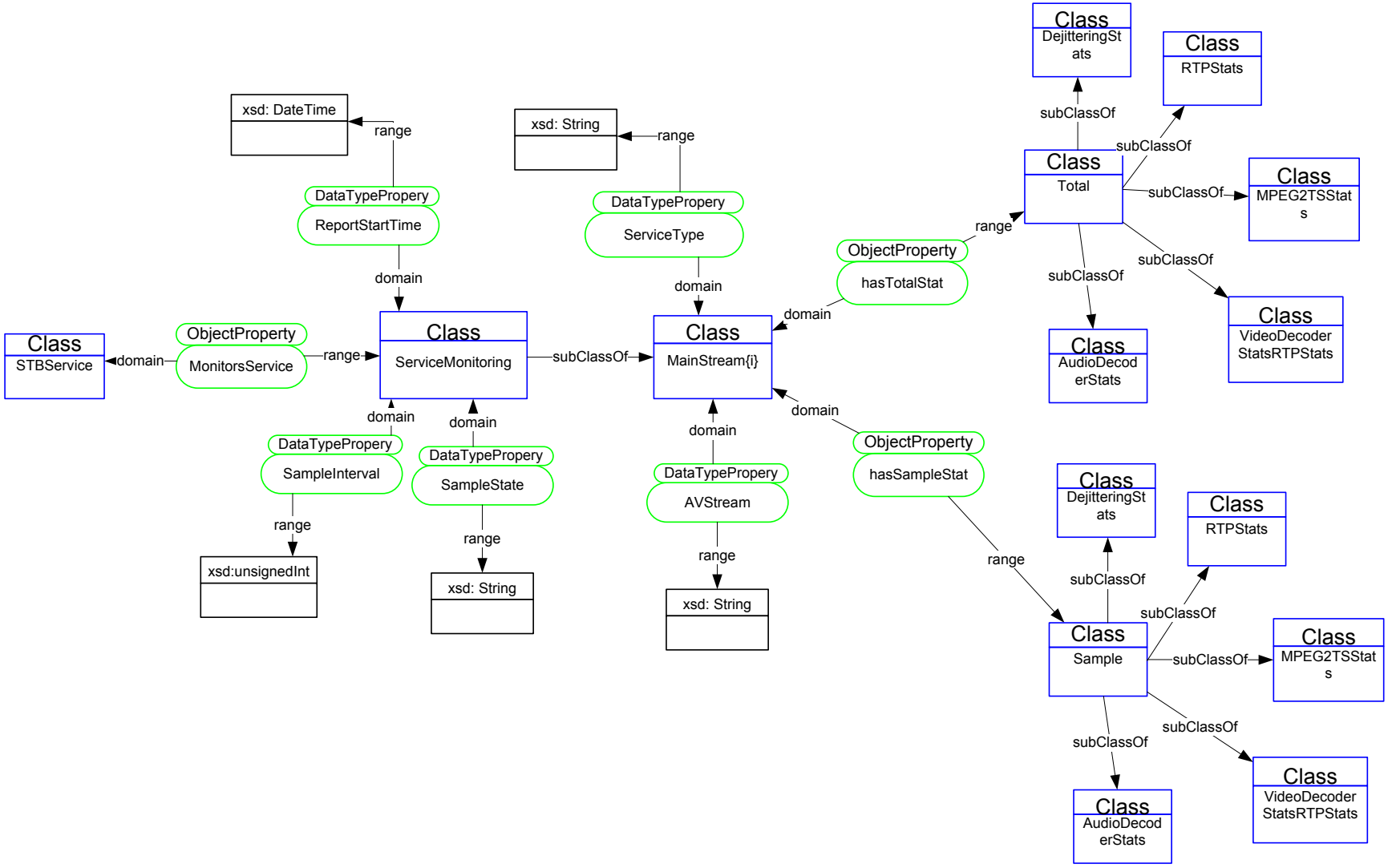

Figure 3-14: STB Ontology using OWL- BasicPerMon

The OWL specifications used are defined as follows:

ObjectProperty: represents a relationship between two classes it links two classes according to a domain and range.

DataTypeProperty: represents an instance of a data type, it links a class to a data type value according to a domain and a range.

Domain: built-in property that links a property to a class description, where the subject of the property should belong to the extension of the class description (i.e. inheritance)

Range: built-in property that links a property to either a class description or a data range, where the subject of the property either belongs to the extension of the class description or its data value is within the data range.

Figure **3-13** shows part of the objects required to deploy basic IPTV (IPTVBaseline profile) service within an STB device. *STBService* is the root class that has property relationships with classes constituting the building blocks for an STB Device. The *Capabilities* class inherits from this root object, and describes what each component (functional block) of the STB can perform (e.g. *FrontEnd* component), in other words, services that can be deployed within this STB object (e.g. streaming). The class also describes the supported streaming protocols and audio/video standards. The class *Components* also inherits from *STBService* class, it represents the STB device functional and computing components e.g. *FrontEnd*, *PVR*, *AudioOutput*, etc.

In Figure **3-14**, we present another example of the STB ontology representing the required objects for an STB offering a basic performance monitoring (BasicPerMon) service. *ServiceMonitoring* class inherits from the *STBService* class. As its name indicates, The *ServiceMonitoring* class holds statistics related to QoE/QoS of the audio/video streams. The statistics are divided in two types: *Total* (statistics are always collected) and *Sample* (statistics are monitored periodically by samples). Both types report on the same statistic parameters that are divided into categories: *DejitteringStats*, *AudioDecoderStats*, *VideoDecoderStats*, *RTPStats*, and others.

3.5. Summary

The device self-configuration framework architecture presented within this chapter reinforced the autonomic concepts necessary to build self-managing systems. We based our framework architecture on the IBM MAPE-K control loop that we adapted to the context of device self-configuration. The main components of the system were explained in details. The framework requires the deployment of an autonomic manager that represents the management entity for managed devices. The autonomic manager comprises the system intelligence within the control loop functional components.

Using the proposed framework architecture, application designers have flexibility in choosing the degrees of intelligence of their systems by using different approaches in the implementation of the components. The four main components are: Device Monitoring, Device Data Analysis, Device Configuration Planning, and Device Configuration Execution. These components collaborate with the Knowledge component to provide autonomous properties to the system.

The Knowledge base constitutes a key factor in realizing autonomic systems' management. It should have the ability of accurately describing the device models, all related environment information that affect their state, and rules that govern their operations. We described how a policy-based approach will fit in the system to translate the contextual information to a set of policies that will govern the user personalized services, and provide an autonomous and seamless management of devices.

We have chosen an ontology-based approach to build our system knowledge given the significance of ontologies in modeling knowledge and storing context information. The ontology described within this chapter was designed to enhance capabilities by providing a full view and understanding of the device components and capabilities. This allows the autonomic manager to provide the devices with a continuous performance support and control to achieve high-level goals and keep the user satisfied.

The proposed self-configuration framework was designed to include management for devices from a diverse range of vendors; therefore, our system is open to all industry standards and

device technologies. The control loop can collaborate with various devices using a matrix of management communication protocols.

In the next chapter, we illustrate the functionality of our proposed device self-configuration framework by an application prototype system. We implement the CPE WAN Management Protocol for communication between the management entity and the managed device.

Chapter 4

4. System Prototype Implementation

4.1. Objective

This chapter describes the implementation of the main components of our proposed self-configuration framework architecture and its use in reconfiguring devices. We implemented the TR-069 agent that should be available in the home device; it plays the role of the CWMP client that communicates with the ACS from one part, and with the user device application from another part. We exploit the functionalities of OpenACS, an open source project implementing the CWMP protocol based on the TR-069 auto-configuration server requirements.

Figure 4-1 depicts the main phases followed in the implementation process. The first phase regarded the installation and exploitation of OpenACS server functionalities. The second phase consisted of developing the TR-069 agent and establishing the first communications with the server. TR-098 as a device ontology was then integrated with the TR-069 agent for concrete communication with the server.

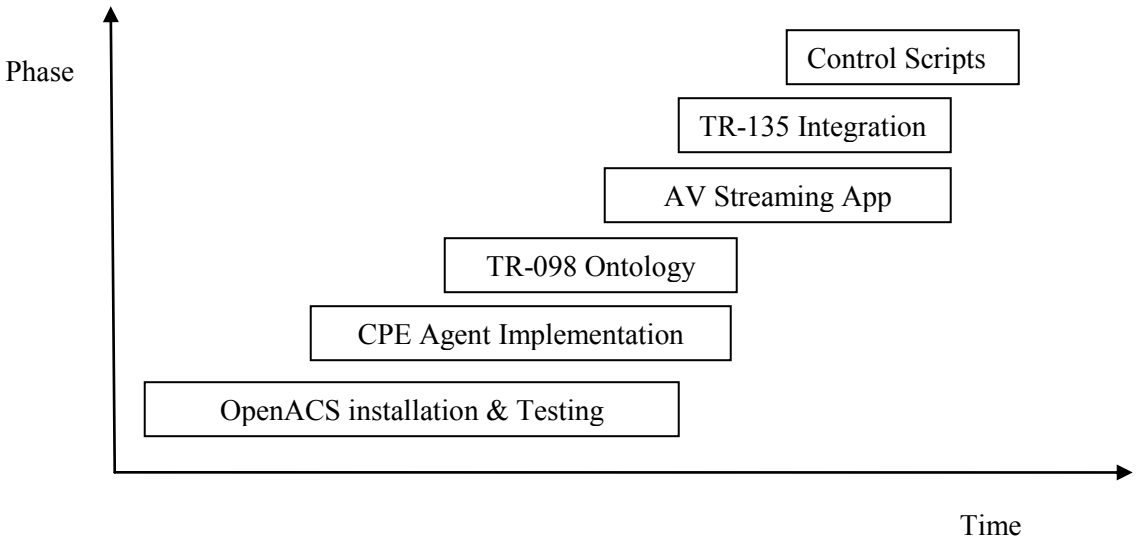

Figure 4-1: Main phases in the implementation process and application prototypes

An audio/video streaming application was designed using Xuggle API, followed by the integration of the STB TR-135, as service ontology that encompasses the video/audio parameters, to demonstrate the applicability and usability of the system in a real life scenario. The part of the control and management components was implemented as scripts run on the openACS.

4.2. Application Scenario

Our scenario is based on two users, they might be physically different users (as we are explaining it bellow in terms of Allan and Bob) or it might be the same subscribed user who can access the service from different locations (e.g. Allan accessing his HAN from a hotel room). In both cases, one entity is considered as a provider and the other is a user of the

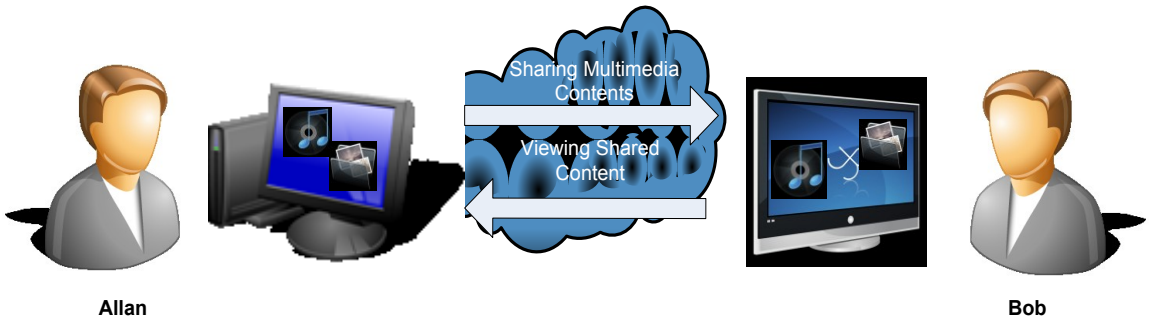

Figure 4-2: Overall Story Scenario

service. Following is a description of the overall story scenario illustrated in Figure 4-2.

Allan, an end-user, equipped of the state-of-the-art HAN, including multimedia devices. He is interested in latest technical developments supporting a digital lifestyle. Allan is open to any services offered by his service provider that promise to enhance his possibilities and/or improve his experience. Allan looks for the full range of consuming the media services, like watching external video content on some device inside his HAN or remotely accessing content stored in his HAN from his laptop in a hotel. But Allan is also an active web2.0

proponent, engaged in various communities, and avidly sharing multimedia content with his friends; both can be in the same town or in remote areas or abroad.

Bob, a pure consumer end-user, invited by Allan to look at shared video content. Even if Bob might not be equipped in the same way as Allan, and not as experienced in the underlying technologies, the quality of consumed service of course matters to him, and so does the usability. If he experiences a well-working service and easy interaction (to set up and troubleshoot the service) then he might be encouraged to use other services also, similar to Allan.

Problem: Let's consider the scenario where Bob selects a high-definition video clip to watch. The media content is then sensed as being of high quality. Appropriate QoS configuration is then set for involved devices (streaming server and rendering device) to consider that this stream traffic is of highest priority. However, this streaming configuration will potentially negatively affect a parallel ongoing video sharing session between Allan and Bob, since the first video streaming will consume most of the available bandwidth.

Challenge: Using the traditional network assurance approaches, it will be hard to check whether Allan and Bob are satisfied by the service quality they are receiving. Our system prototype that involves using the functionalities of the CWMP auto-configuration server will outline an approach that integrates the user devices in the HAN into the overall service assurance concept (see Figure **4-3**). This involves keeping track of the performance parameters on the HAN devices and also includes an optimized interaction with the end user. For the operation of the services this shall result in better customer satisfaction of the quality of the consumed service and will considerably reduced customer frustration and complaints.

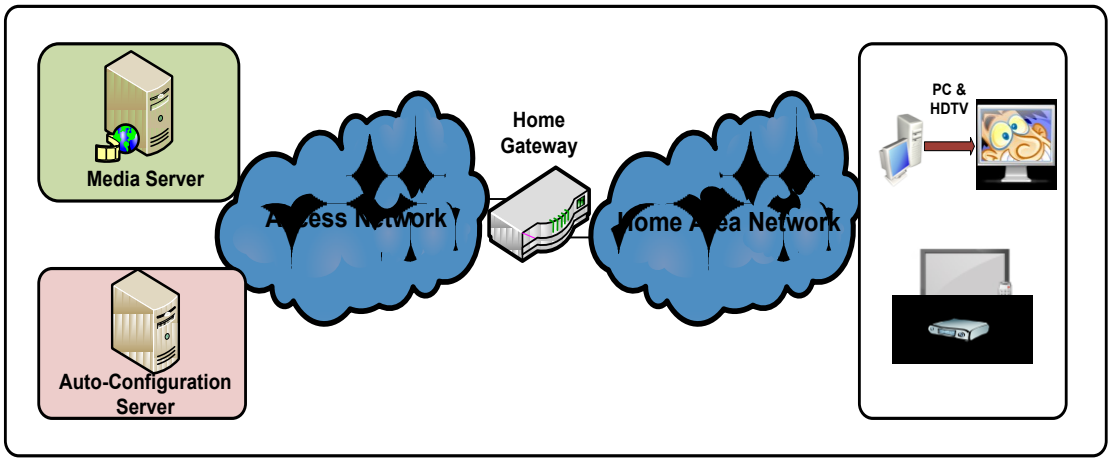

Figure 4-3: General Prototype scenario

4.3. Physical Architecture

This section describes the physical architecture for the scenario described in the previous section. Towards achieving the self-configuration of devices we consider using CWMP management protocol for communication between Customer Premises Equipment (CPE) and a management entity in the form of an Auto-Configuration Server (ACS) whose functionalities are summarized in: auto-configuration of user CPEs (including initial configuration), dynamic service provisioning, software/firmware image management, status/performance monitoring, and diagnostics. The ACS has two communication interfaces: southbound interface; for communication with user CPEs, and northbound for communication with third-parties such as service providers as depicted in Figure 4-4. In this work we are more concerned with the southbound interface.

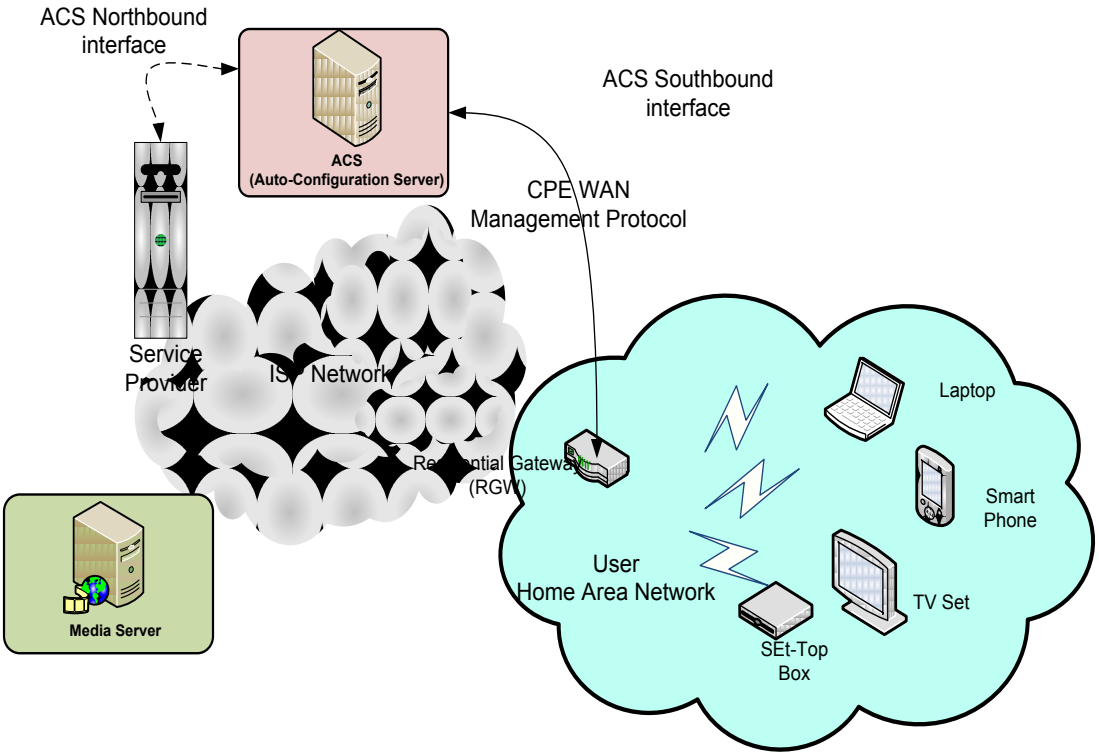

Figure 4-4: Scenario Physical Architecture

Streaming servers stream data to HAN devices. It is able to serve data either unicast or multicast. The multimedia content streaming is incorporated in this scenario for the purpose of demonstrating a real-life applicability of our device/service self-configuration system. These servers can exist inside a HAN or in the outer network.

The home residential gateway (RGW) is the entry point to the HAN. It is the networking device that connects the home devices to the WAN. Among others, it provides functionalities such as, Dynamic Host Configuration Protocol (DHCP) [38] functions, firewall functions, Local Area network (LAN) connectivity, Digital Subscriber Line (DSL) connectivity, etc. The residential gateway device should ideally have management capabilities and be able to run software in it.

The user home area network holds multiple digital devices. Different classes of devices might exist: workstations, multimedia players, communication devices, etc. In particular, we are interested in a multimedia player device that is capable of receiving streams via Internet Group Management Protocol (IGMP) [39] and Real Time Streaming Protocol (RTSP) [40]

and decode these streams whether it is a standard definition or high definition streams. This device can have either Ethernet or wireless connection by USB to wireless dongle. The device can also be managed remotely using Ethernet port of the device. The device must provide a user interface to display the messages regarding the network operations and any error condition. The device should also provide other home devices and the residential gateway device data about the QoS, fault diagnosis, etc. The device should be able to process and run applications, namely the CWMP client. Example of such devices would be PCs, smart phones, STB, etc.

CWMP Integration:

Figure 4-5 shows the integration of TR-069 family in the residential gateway and user CPEs. As it has been presented in chapter 2, BBF published a set of technical reports in addition to the TR-069 that describes the management protocol, namely:

- TR-135: Set-Top-Box,

- TR-104 for Voice over IP,

- TR-140 for storage,

- TR-106 for a general Data Model Template.

- TR-098 for residential gateway.

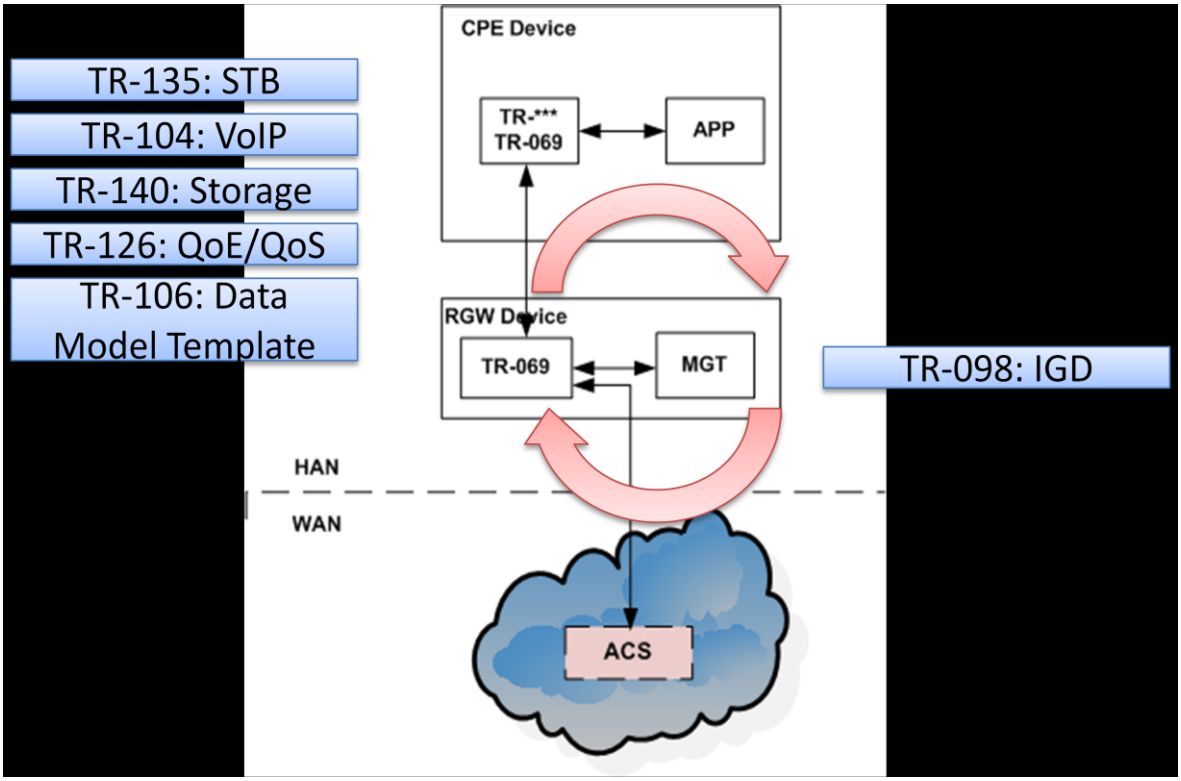

Figure 4-5: TR-069 Family Integration

The complete reports list is available at [41]. For devices to communicate with ACS using CWMP, they have to implement the corresponding device/service data model and respect requirements that come with technical reports. Every device should hold a TR-069 agent that constitutes a client/server process interface between the service and device applications we would like to manage (represented in Figure 4-5 as APP for a CPE device and MGT, referring to a management application, for a RGW device) and the management entity. The integration of CWMP goes hand in hand with our goal to achieve the autonomic control loop.

In our prototype implementation for the TR-069 agent, the data model of RGW [42] was adopted for proper communication with the ACS. Since the application we would like to evaluate is related to video on demand streaming, the STB data model [37] has been chosen to encapsulate the video/audio parameters using *STBService* object, and hence plays the role of a Set-Top-Box device behind RGW.

In developing the application prototype, we considered specific requirements necessary for an efficient parameter configuration of device application. We also considered the capabilities of the network and the devices, and the rules binding all these components together so that they consistently satisfy user needs. Such requirements include:

- Ability to seamlessly access the management entity.

- Authentication of users and personalized preferences configuration.

- Obtaining a seamless device and service management.

- Control and management of dynamic changes in the user preferences that may occur during service and device operation.

- Role of ACS: monitors the operation and performance of both RGW and STB, allows specific device parameters to be checked, and carries out necessary configuration of devices.

- CPE devices making use of an STBService object MUST adhere to all of the data-hierarchy requirements defined in TR-106 [43].

As it is stated in [37], there exist two types of management control for an STB with regards to an IPTV service:

- First, controlled by the IPTV Service Platform for all tasks related to provisioning, media processing and Conditional Access / Digital Rights Management.

- Second, controlled by the ACS for all tasks related to trouble management and monitoring of performance and operational status.

Based on our scenario, the goals of these specifications are as follows:

- Enable ***configuration*** by the ACS of those ***objects*** and ***parameters*** that are *not* the responsibility of the IPTV Service Platform.
- Enable operational status monitoring and checking of specific parameters of an STB from an ACS.
- Enable performance monitoring of an arbitrary set of STBs, from one to millions, through estimates of QoS (Quality of Service) and QoE (Quality of Experience), where QoS and QoE are defined in TR-126 [44].

In our implementation, the STB data model is embedded as part of the RGW data model.

In the following section we elaborate on how this control loop is achieved by presenting the architecture component diagram and explaining the self-configuration procedure.

4.4. Implementation component diagram

Figure 4-6 shows the component architecture for the device self-configuration framework presented in the previous chapter. The main components presented in Figure 3-4 (Framework Architecture) are included. We adopted the implementation architecture to the scenario described in the preceding section.

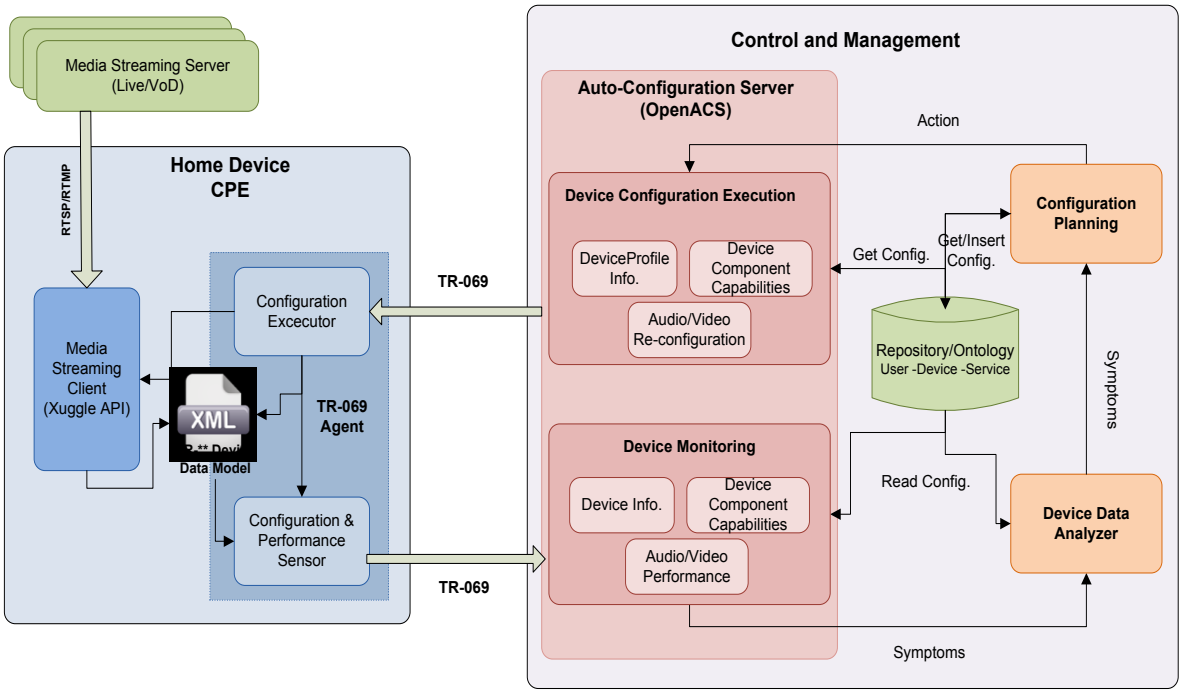

Figure 4-6: Architecture for device Self-configuration framework

Media streaming server is used to stream videos using an RTSP/RTMP protocol. We used Xuggle API to implement the media streaming client. Video/audio general and performance parameters are extracted in TR-135 schema xml files. The TR-069 client agent is composed of two components: *Configuration & Performance Sensor* and *Configuration Executor* that represent the device *Sensor* and *Effector* operations presented in chapter 3. It uses TR-098,

symbolizing a RGW, to communicate with OpenACS since no direct communication with other type of devices is permitted in the current version. The communication between the two entities is done through an exchange of Simple Access Object Protocol (SAOP) messages conform to TR-69 protocol specifications.

Both the *Device Monitoring* and *Device Configuration Execution* components are implemented as part of openACS. Default implementation of these modules comes with the open source server. However openACS gives a possibility for their control through scripts, in terms of the selection of parameters to monitor. Configuration parameters that we aim to monitor and reconfigure concern variety of device/service parameters. We group these parameters in the following categories:

- Device/DeviceProfile Info.: Monitoring of parameters related to device and to the periodic sample intervals needed for monitoring (an example is presented in Figure 4-7 e.g. Tree root is *InternetgatewayDevice*)
- Device Component & capabilities: Draw the tree of device components objects and parameters that are enabled to be monitored (it should be similar to the device ontology at the server level).
- RGW Info.: Information related to the RGW management (e.g. Management Server URL, net. Info about forwarding)
- Audio/Video Performance: audio and video periodic statistics (e.g. *FrameRate*, *DecodedFrames*, *LostFrames*, etc.)

The *Device Data Analyzer* and *Device Configuration Planning* modules are logically kept outside of the openACS as their functionalities are not included in the roles of ACS. However, implementation of these two modules is injected in the openACS using scripts and it is kept simple by hard-coding the policies logic.

Figure 7 shows the TR-098 XML data model that is used in the client. All the objects are encapsulated within the *InternetGatewayDevice* object that refers to the residential gateway

```
<InternetGatewayDevice>
   <DeviceSummary Write="true">InternetGatewayDevice (Baseline:1,
                              EthernetLAN:1, ADSLWAN:1)</DeviceSummary>
       <DeviceInfo>
              <Manufacturer xsi:type="xsd:string(64)">Cisco</Manufacturer>
              <ManufacturerOUI xsi:type="xsd:string(6)">00000C</ManufacturerOUI>
              <ProductClass xsi:type="xsd:string(64)">Networking
                                                    Device</ProductClass>
              <SerialNumber xsi:type="xsd:string(64)">4231278</SerialNumber>
              <SpecVersion xsi:type="xsd:string(16)">1.0</SpecVersion>
              <HardwareVersion xsi:type="xsd:string(64)">HDDR_V1.0</HardwareVersion>
              <SoftwareVersion xsi:type="xsd:string(64)">SDDR_V1.0</SoftwareVersion>
              <ProvisioningCode xsi:type="xsd:string(64)">1.0</ProvisioningCode>
       </DeviceInfo>
       <ManagementServer>
              <URL xsi:type="xsd:string(256)" Write="true">
                                    http://137.122.88.66:8080/openacs/acs</URL>
              <Username xsi:type="xsd:string(256)" Write="true">openacs</Username>
              <Password xsi:type="xsd:string(256)" Write="true">openacs</Password>
              <PeriodicInformEnable xsi:type="xsd:boolean" Write="true">
                                                     true</PeriodicInformEnable>
              <PeriodicInformInterval xsi:type="xsd:unsignedInt[1:]" Write="true">
                                                   1800</PeriodicInformInterval>
              <PeriodicInformTime xsi:type="xsd:dateTime" Write="true">
                                                        </PeriodicInformTime>
              <ConnectionRequestURL xsi:type="xsd:string(256)">
                       http://137.122.88.66:8080/openacs/acs</ConnectionRequestURL>
              <ParameterKey xsi:type="xsd:string(32)">
                                                 onTransferComplete</ParameterKey>
       </ManagementServer>
</InternetGatewayDevice>
```

Figure 4-7: Example of TR-098 XML data model used

device. *DeviceSummary* refers to a summary of the device capabilities and services deployed. (should add the STBService). The attribute Write, if set to true, means that this parameter can be written or modified by the ACS. The *DeviceInfo* object gives metadata information about the device: *Manufacturer*, *ManufacturerOUI* (Organization Unique Identifier), *SerialNumber*, etc. *ManagementServer* object provides information related to the management process, where the server *URL* and authentication parameter (*username* and *password*) are specified. Within this object, we also state whether periodic monitorin is enabled for this device or not. This is done by affecting *true* value to *PeriodicInformEnable*. We also set the periodic interval for this monitoring in the parameter *PeriodicInformInterval*.

4.5. Device Self-configuration Procedure

The scope of device self-configuration is defined by three stages: *initial configuration phase*, *monitoring*, and *re-configuration phase* (Figure 4-8). Our proposed model can be applied to a wide selection of home devices such as PDA, set-top box, computers, etc. The goal behind the self-configuration of these devices is to ensure a proper operation according to desired quality required by the user. Initial configuration phase is the first phase upon a new installation, resetting, or powering of the device. The device enters then in an operational mode. During this time of operation, performance parameters of the device/service are monitored, either periodically or upon a change of value. These values are then analyzed and if improper state is detected, we proceed into the re-configuration of corresponding parameters, which consists of changing accordingly the value of these parameters. We iterate between monitoring and reconfiguration as long as the device is in operation.

Figure 4-8: Self-configuration scope

Figure 4-9 outlines the device self-configuration procedure in details. In the following we describe each phase and explain the logic behind it:

Initial Configuration: As a device is powered on and discovered inside the home (steps 1~4), it is essential to perform initial configuration to prepare it for further management actions. First, a session is opened between the CPE device and openACS (steps 5~6). The initial configuration can be specific to the device functionalities/behavior, for example, providing specific services scanning, management server related parameters configuration, periodic interval time for monitoring, etc. As well as it can be related to the user preferences; in the case of different user profiles using the device, such as preferred language, favorite services.

It is also vital that the management entity first knows the supported management functions; this can be expressed in TR-069 as *GETRPCMethods* request (steps 7~14). After the initial configuration, the device then enters the operational mode (step 15), and services, video streaming in our scenario case, is then being used.

Monitoring: Different monitoring approaches have been introduced in the literature. We adopt an active and passive monitoring approach as explained in chapter 3. In the latter, monitoring is either organized to receive passively periodical messages summarizing the current state of the device or upon a change of value of some parameters that we first configure their notification as being active or passive. In terms of TR-069, these two actions correspond to *PERIODIC* and *VALUE CHANGE* events. As for the former, we actively trigger monitoring to reinforce data analysis. Monitoring uses ontologies to check the conformity of parameters' name, data values, and relationships (step 16). The monitoring module gets a feedback from the analysis module to request a parameter value (i.e. *GetParameterValues*) (steps 17~18) from the device to complete the problem identification and target the re-configuration planning. For example, degradation in the video quality resulting in QoE issue can trigger a *VALUE CHANGE* event upon a significant change in the visibility indicator, for instance.

Re-configuration: This phase represents the core phase of the self-configuration process, as all the modules collaborate together to resolve any performance issues. While the device is in operation, continuous monitoring is carried out; in particular we are interested in performance related parameters for media applications. If we consider again the degradation in video quality scenario; as the analysis module receives the monitored parameters (step 19~20), to identify the source of the problem, it makes use of the predefined policies available in the base to check on different possibilities of the cause, it can contact the monitoring module back in case it needs to know about the value of parameters that can potentially affect the parameter in question, parameter values related to QoS (packet loss, jitter, etc.) are relevant in this scenario. Once the cause is identified, appropriate configuration plans (i.e. parameters names and values according to device ontology) are created by the configuration planning module (steps 21~22), and sent back to the device for re-configuration using the method *SetParaameterValues* (steps 23~27).

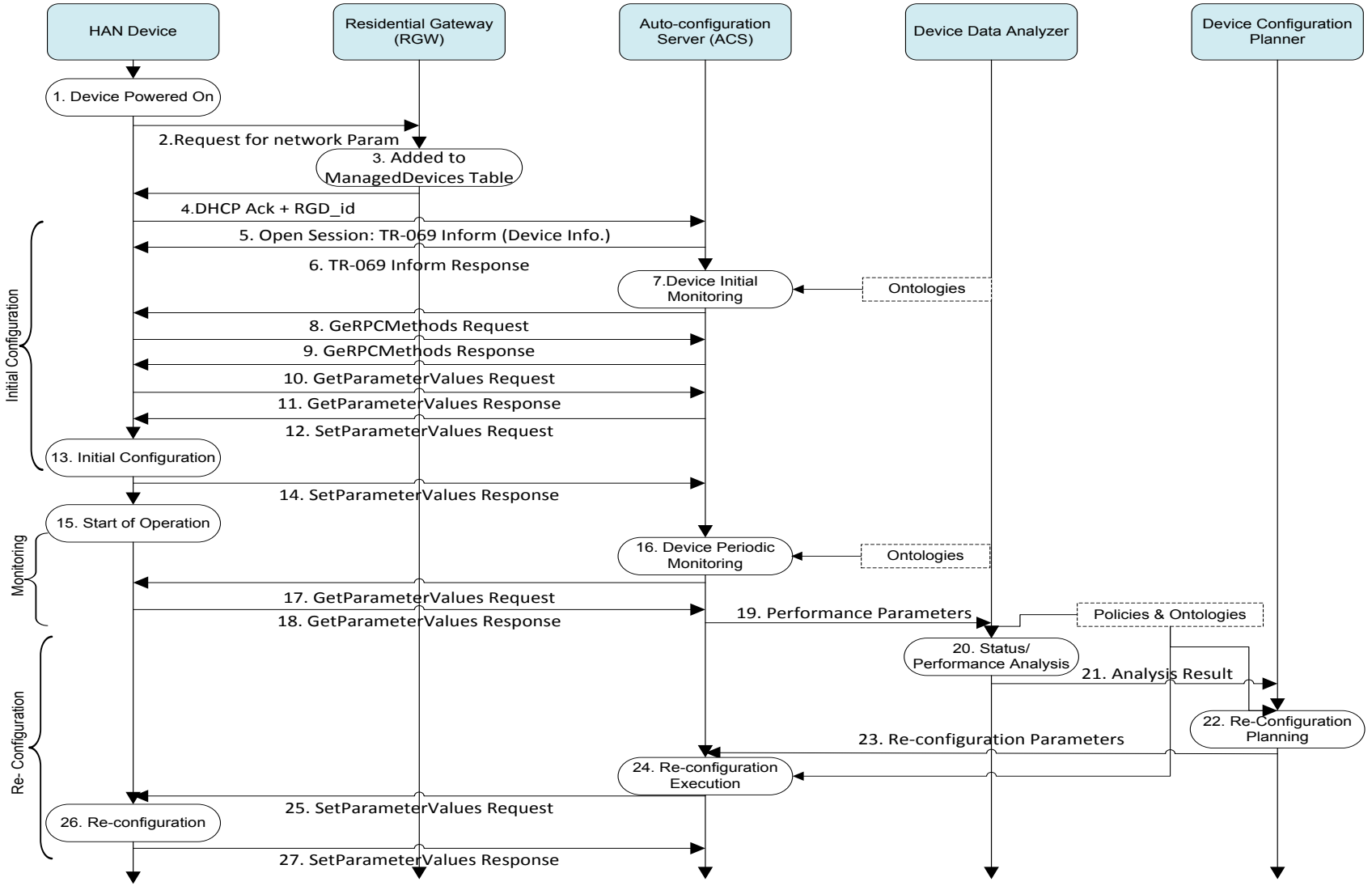

Figure 4-9: Device Self-configuration Procedure

4.6. TR-069 Client

The implementation of the TR-069 client was done in two steps. The first step concerned the implementation of the core classes to realize communication with openACS. We based our implementation on an open source code that defined the general structure of the client [45], but did not have a complete working code; because the project was abandoned. We debugged the code and completed its implementation. The second step concerned the integration of TR-135 and video streaming application. Since the two applications (streaming and client) should be run simultaneously, we used Java Threads to enable a concurrent programming.

The TR-069 client was developed using Java language. Additional open source libraries were used such as: dom4j [46], Apache Axiom [47] both make working with XML easy on a Java platform, and Jakarta Commons *HttpClient* component [48] and its dependencies for the implementation of the HTTP protocol client side.

Figure 4-10 shows the UML representation for our TR-069 Client. The *CPEController* class is the core class that implements the *sensor* and *effector* operations as described in

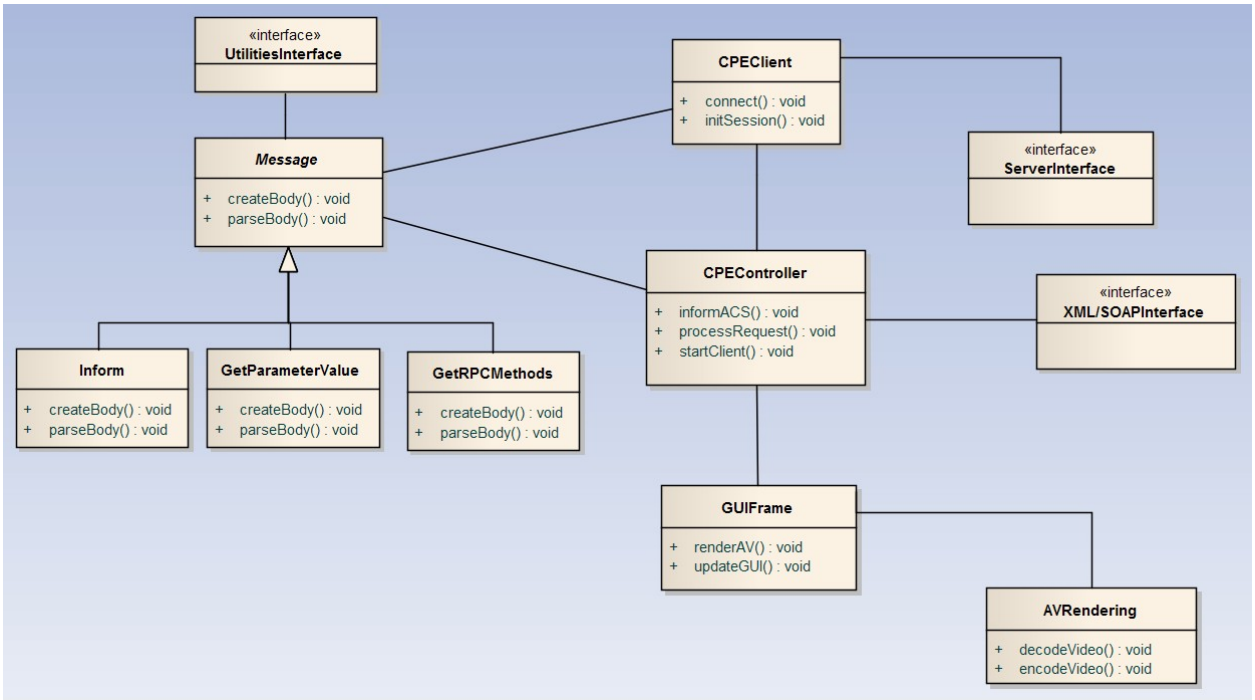

Figure 4-10: TR-069 Client Class Diagram

chapter3. The class is instantiated in a separate thread as the application is powered on from the *GUIFrame* class.

The *CPEController* class launches the *CEPClient* class that initializes the session with the openACS. *CPEClient* class implements the methods of interaction with the server, i.e. sending and receiving messages using HTTP protocol. To contact the openACS, there are three ways to know about the server URL, the most interesting and more realistic method is based on DHCP (more details about this procedure can be found in TR-111 [49]). The two other ways would be having the URL built-in in the CPE client, or allowing a local configuration through the user interface. We opted for this latter for its simplicity, but still it allows a minimum of control. Client/Server first contact is done trough sending an empty HTTP Get request. In our implementation we only deal with the case where the user CPE first contacts the server. All the cases of session initialization are listed in [14].

TR-069 session establishment is done though sending a SOAP *Inform* message. The *Inform* message contains information about the device, an example of the Inform message is illustrated in Figure 4-11. As all applications based on web services, our application uses SOAP/HTTP binding to allow a synchronous message exchanging between the TR-069 client and openACS. The cwmp:ID used in the SOAP header represents the identification of the current message. When the server replies back to the device it should use the same ID. Likewise, the device when replying back to a server message, it has to use the ID of the request message. The SOAP body provides a serialization of the XML schema representing the TR-069 specific device data model in a way that respects the CWMP protocol. The first element of the body provides the name of the message (e.g. *Inform*). The second element is *DeviceID* that encapsulates the device information as we have seen for *DeviceInfo* element in Figure 4-7. Then the message specifies the *Event* type for which the device is notifying the management server. This is expressed in the parameter *EventCode*. There exist some predefined event codes such as "1 *BOOT*" that indicates that the session is being established upon a powering on of the CPE or reboot, "2 *PERIODIC*" indicates that the session is being established on a periodic inform interval, etc. TR-069 also allows a customization of these codes as needed by the manufacturer (the detailed list can be found in [14]). The

third element of the *Inform* message is *ParameterList*. This element contains the list of

```
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:cwmp="urn:dslforum-
org:cwmp-1-0" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <SOAP-ENV:Header>
   <cwmp:ID>123</cwmp:ID>
 </SOAP-ENV:Header>
 <SOAP-ENV:Body>
   <cwmp:Inform>
     <cwmp:DeviceID xsi:type="cwmp:DeviceIdStruct">
      <DeviceIdStruct>
        <Manufacturer xsi:type="xsd:string(64)"> Cisco</ Manufacturer>
        <OUI xsi:type="xsd:string(6)"> 00000C</OUI>
        <ProductClass xsi:type="xsd:string(64)">NetDevice</ProductClass>
        <SerialNumber xsi:type="xsd:string(64)"> 4231278</SerialNumber>
      </DeviceIdStruct>
     </cwmp:DeviceID>
     <cwmp:Event SOAP-ENV:arrayType="xsd:EventStruct[3]">
      <EventStruct>
        <EventCode>0 BOOTSTRAP</EventCode>
        <CommandKey/>
      </EventStruct>
      <EventStruct>
        <EventCode>1 BOOT</EventCode>
        <CommandKey/>
      </EventStruct>
     </cwmp:Event>
     <cwmp:MaxEnvelopes xsi:type="xsd:int">1</cwmp:MaxEnvelopes>
     <cwmp:CurrentTime xsi:type="xsd:dateTime">Fri Aug 13 10:42:32 EDT
                                                2010</cwmp:CurrentTime>
     <cwmp:RetryCount xsi:type="xsd:int">0</cwmp:RetryCount>
    <cwmp:ParameterList SOAP-ENV:arrayType="xsd:ParameterValueStruct[1]">
      <ParameterValueStruct>
        <Name>InternetGatewayDevice.DeviceSummary</Name>
        <Value xsi:type="xsd:string"> InternetGatewayDevice (Baseline:1,
                                      EthernetLAN:1, ADSLWAN:1)</Value>
      </ParameterValueStruct>
    </cwmp:ParameterList>
   </cwmp:Inform>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 4-11: Example of CPE Inform Message

parameters (along with their corresponding *names* and *values*) we would like to notify the server about. This list changes significantly according to the event and type of message.

In our prototype application the SOAP messages get manipulated according to their type at the level of the classes that inherit from the *Message* abstract class (see Figure 4-12). Message class implements the core methods that are being used by all the sub-classes, and gives the methods' definition for those to be customized to the specific type of message. For example, the method create(), that creates a new SOAP message envelop with a header, has the same implementation for all the messages, as opposed to createBody() that differs from a message to another (the SAOP body of an *Inform*

message is different from the one of *GetParameterValues* message, that is different from *GetRPCMethods* message).

```
import jcpe.src.cn.edu.njupt.sc.GUI.CPEFrame;
import org.apache.axiom.om.OMNamespace;
import org.apache.axiom.soap.impl.dom.soap11.SOAP11Factory;

public abstract class Message {
      public Message() {

      }

      abstract protected void createBody(SOAPBodyElement body, SOAPFactory spf)
                  throws SOAPException;

      abstract protected void createBody(SOAPBodyElement body, SOAPFactory spf,NameValue[] nameValue)
      throws SOAPException;

      abstract protected NameValue[] parseBody(SOAPBodyElement body, SOAPFactory f)
                  throws SOAPException;

      public byte[] create() {

            try {
                  MessageFactory mf = MessageFactory.newInstance();

                  SOAPFactory spf = SOAPFactory.newInstance();

                  SOAPMessage msg = mf.createMessage();
                  SOAPEnvelope env = msg.getSOAPPart().getEnvelope();

                        env.addNamespaceDeclaration(CWMP, URN_CWMP);
                        env.addNamespaceDeclaration("SOAP-ENC",
                                    "http://schemas.xmlsoap.org/soap/encoding/");
                  env.addNamespaceDeclaration("xsi",
                              "http://www.w3.org/2001/XMLSchema-instance");
                  env.addNamespaceDeclaration("xsd",
                              "http://www.w3.org/2001/XMLSchema");
                  msg.getSOAPHeader().addHeaderElement(
```

Methods to be overridden

Same definition for all sub-classes

Figure 4-12: Java code extract of the Message Abstract Class

The implemented client offers a friendly user interface to manage the device self-configuration parameters either by an admin or a user (for the user it is optional as it

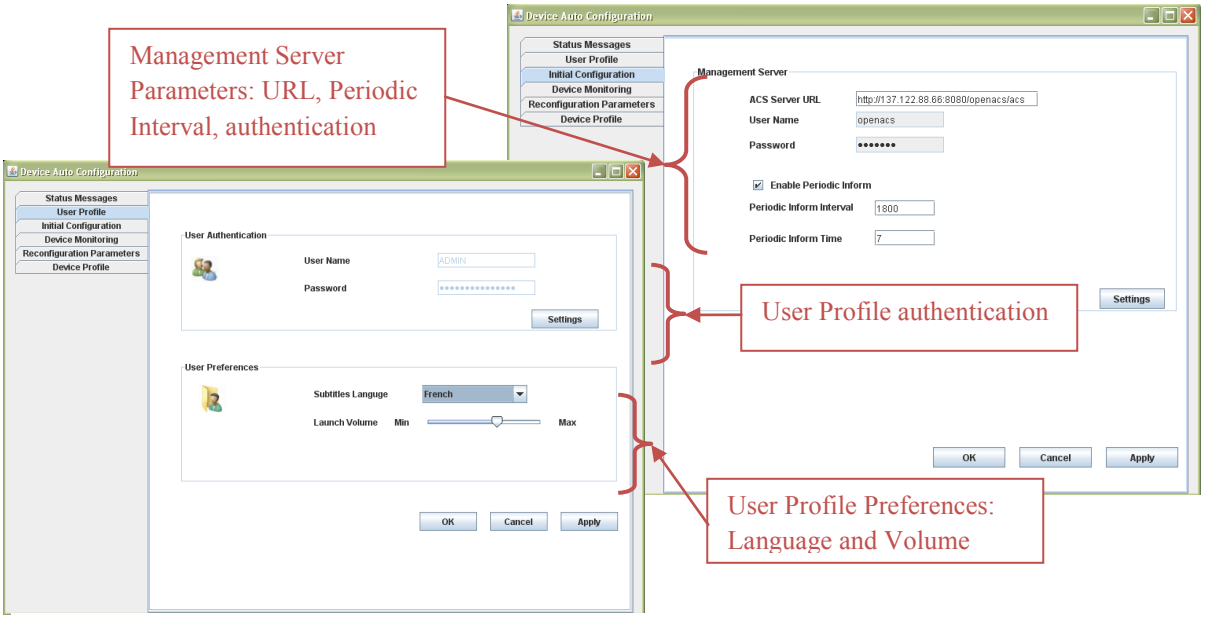

Figure 4-13: User Profile Preferences and Initial Configuration Settings

depends on the level of technical knowledge). It also provides a possibility to customize the user preferences.

Figure 4-13 shows the user interface for both user profile preferences and Initial configuration parameters settings. As different user profiles can exist, it is necessary to provide an authentication option that takes into consideration the user name and password, so that it loads the possible preferences settings. Among these settings we include in this interface the subtitling language and the media volume. The subtitling language exists as parameter in TR-153, whereas the volume can be flexibly added to the data model by adding a new object with its related parameter list elements. The initial configuration parameters include the management server parameters necessary for the CPE/ACS communication. These parameters are the same as the ones they appear in Figure 4-7. These values of these parameters are read form the UI to create and/or update the data model XML files. We also provide an interface that gives the user a feedback about the communication status between the CPE and openACS, and another one that gives notification and history of parameter re-configurations performed.

In the following section, we present the video streaming application that was used for a more real-life usability testing of our device self-configuration system.

Video Streaming Application:

For the implementation of our streaming application we used Xuggler API [50] that is compatible with the Java platform. Xuggler is an open source API that presents an efficient way to decode and modify media files or streams using Java. For the purpose of the experiment, both the TR-069 client and the streaming application were integrated in the same project application, and were launched in different threads using Java threads. The choice of Xuggler was based on our need for manipulation of the media, to facilitate the monitoring and re-configuration of parameters. Xuggler allows an easy and complete control over the media files containing audio, video, and data. Hence provides the perfect tool to fine tune the user preferences streaming settings. We gave the example of the media volume that can be customized according to the user profile. Other parameters can involve the visual quality of video received such as resolution and Frame Rate Per

Second (FPS) that can be fine tuned according to the codec used (e.g. MPEG-4, H.264, etc.). Such parameters can be reconfigured in case bandwidth issues are encountered or there is a storage limitation, and also accordingto the user preferences. All these parameters can be easily extracted (to be monitored) and written back (for re-configuration purposes) using Xugger API.

To communicate the audio/video parameters to openACS, we used TR-135 that specifies the data model for an STB as media steaming is among the services offered within an STB. Hence it provides parameters that are suitable for our streaming application.

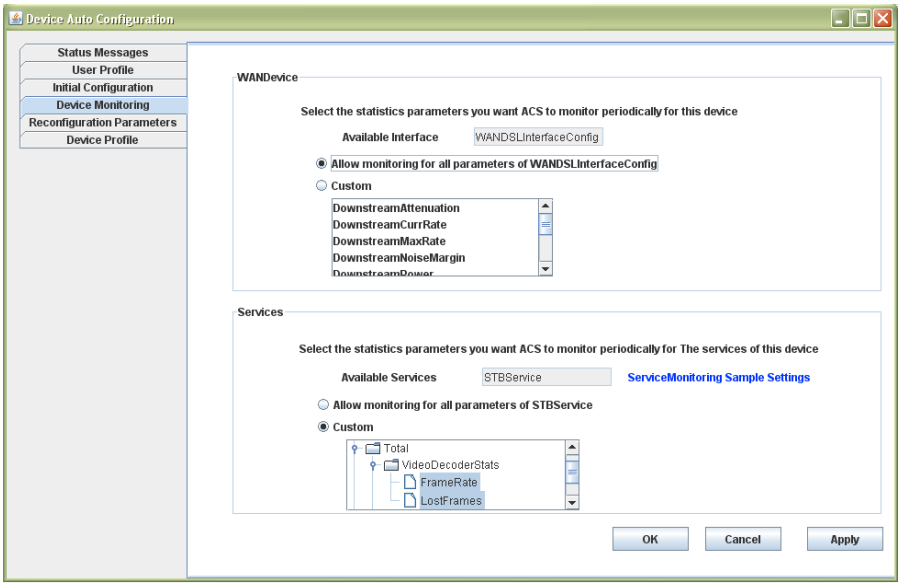

Figure 4-14: Screenshot of the Client Devicce Monitornig Interface

Figure 4-14 shows the interface that allows the user to choose the performance parameters to be monitored and evaluated at the level of openACS. The list of parameters is extracted from the XML file describing the device ontology. Figure 4-15 depicts snapshots of the video streaming application with different frame rates.

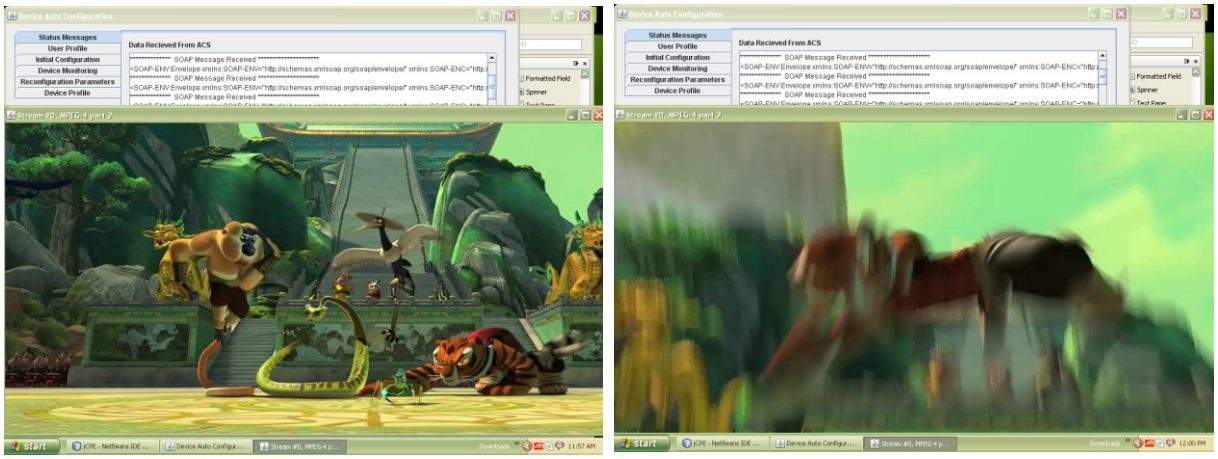

Figure 4-15: Screenshots of the Video Streaming Application

4.7. OpenACS

In the control and management part, we use OpenACS [51], an open source project developed by Audrius Valunas. It implements the CWMP protocol based on the TR-069 auto-configuration server requirements. The project is implemented as a J2EE application and it is deployed using JBoss application server. The code is published on source forge. Currently, almost all ACS functionalities are implemented. In addition to default configuration functionalities, OpenACS allows a customization of the control and management logic, using scripts, as explained in section 4.4.

As OpenACS receives the *Inform* message, it runs a *Default* script, in this script are called appropriate scripts (*monitoring* and/or *analysis*) according to the *Event* we get in the Inform message. Figure 4-16 shows a snapshot from the server default script. For a *BootStrap* event, the

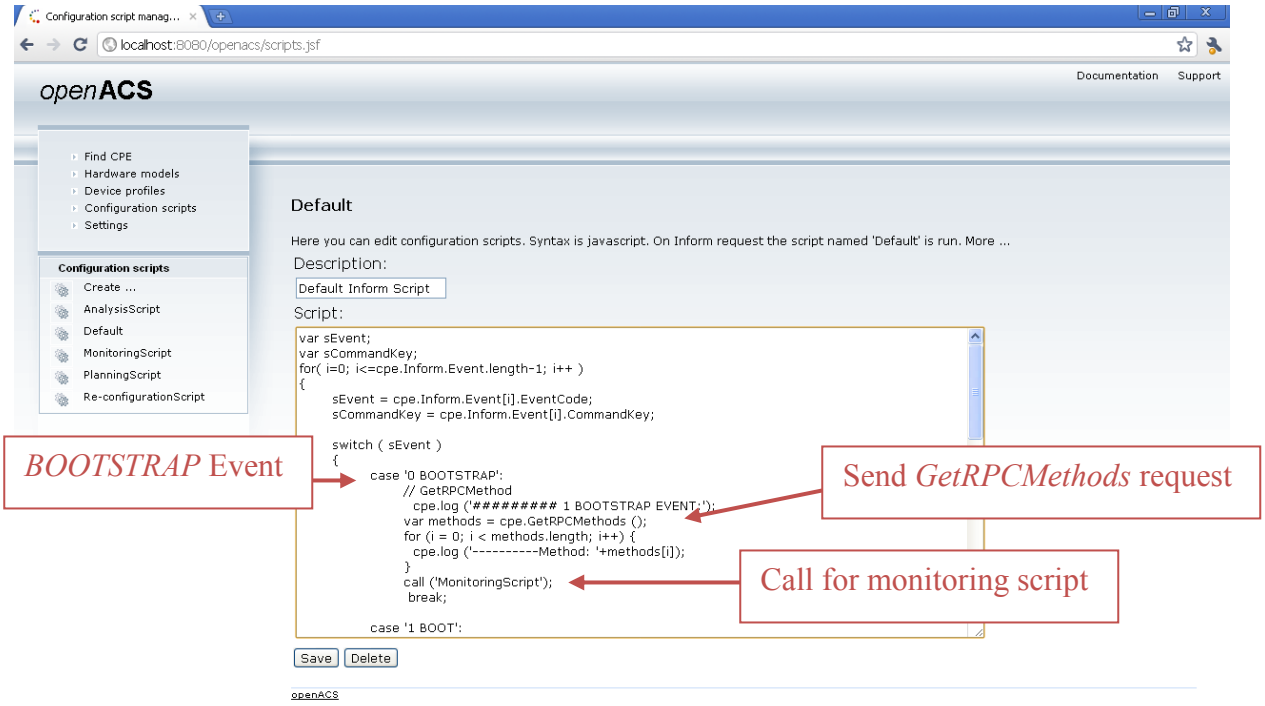

Figure 4-16: Snapshot of OpenACS script interface

script sends a *GETRCPMethods* request to discover the methods that are handled by the device. Then it calls the *MonitoringScript* for monitoring of parameters. For a *PERIODIC* event we call *AnalysisScript*.

OpenACS uses a MySQL-based database into which all acquired and inferred device information is saved. We used reverse engineering of the software MicroOLAP Database Designer for MySQL to generate the Entity Relationship (ER) diagram presented in Figure 4-17.

The hostsbean table is the core table that represents device information such as url, the URL that the server uses to contact CPE, as well as information about the software (currentsoftware) and its update times (sfwupdtime) and configuration update time (cfgupdtime). Hostbean table has references to tables:

- *deviceprofilebean* table, where values are affected to records about data such as periodic interval.

- *hardwaremodelbean* table that describes details about the device model such as *displayName*, hardware class *hclass*, etc.

- *softwarebean* and *softwaredetailbean* tables, both have related records about the software version (*version*), parameter name (*paramName*), url, etc.

- *Configurationbean* table that has the configuration file (*config* expressed as a *longblob* data type), the *filename*, and *version* of configuration. It is important to note that this configuration is related to the configuration that the server receives from the northbound interface (from service providers and third party players, for instance) and that is different from the configuration we are addressing in this work. However similar approach could have been carried out.

- *dsldatabean* table, stores records related to device performance statistics related to downstreamAttenuation, downstreamCurrRate, etc.

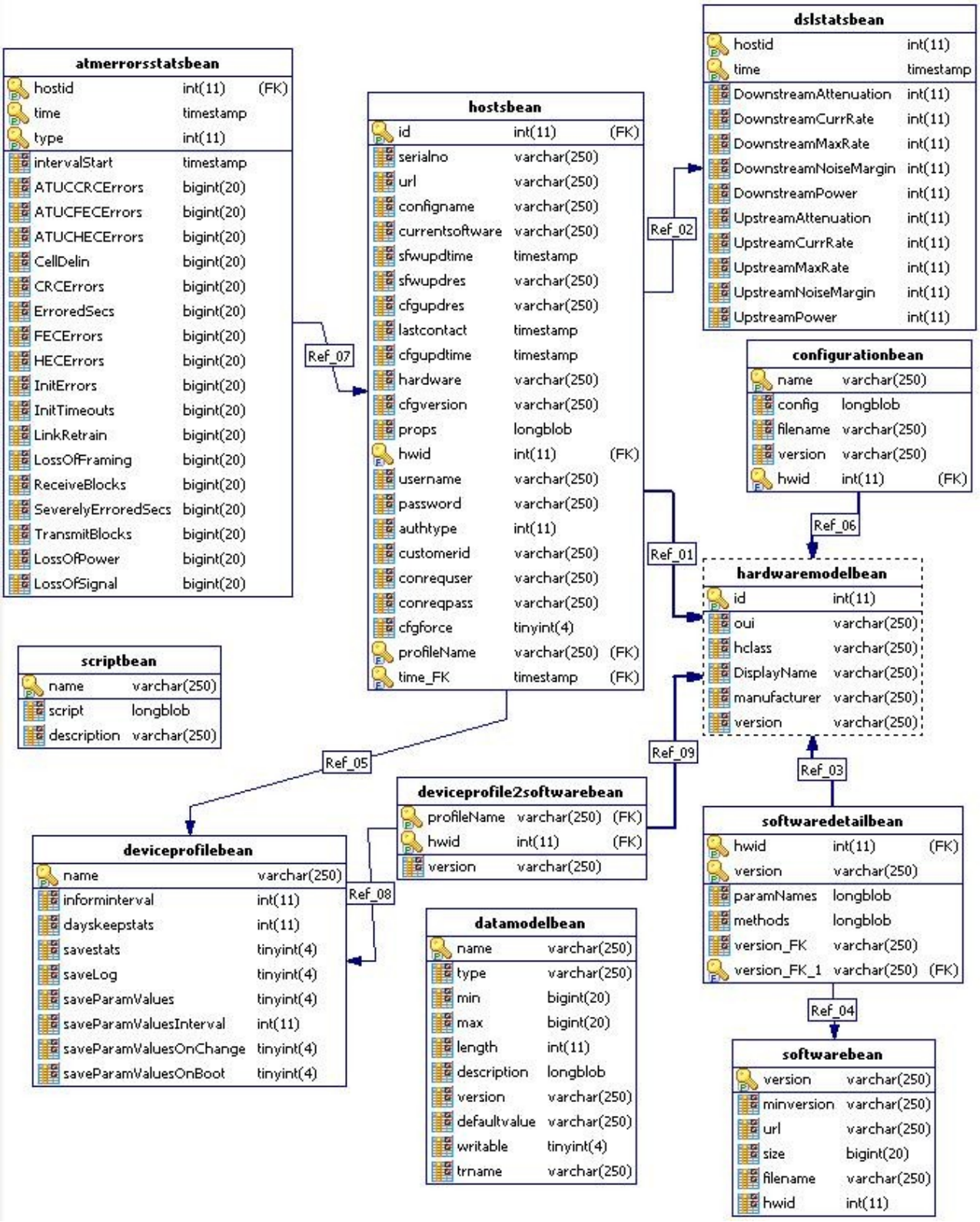

Figure 4-17: OpenACS ER Diagram

The database provides an ontology-independent storage for the device knowledge. OpenACS server does not use any API to access and manipulate device/service ontologies as it is the case to access the XML files in the TR-069 client. SQL capabilities are rather exploited when performing ontology-based approach.

Single domain specific ontology (Service/Device) showed in Chapter 3 is handled by a set of tables (*datamodelbean* table). Figure 4-18 is an extraction of the table results where column *name* refers to the name of objects and parameters as described in TR-135. The data type is specified in column type, and then the minimum and maximum allowed values are expressed in *min* and *max* columns, respectively.

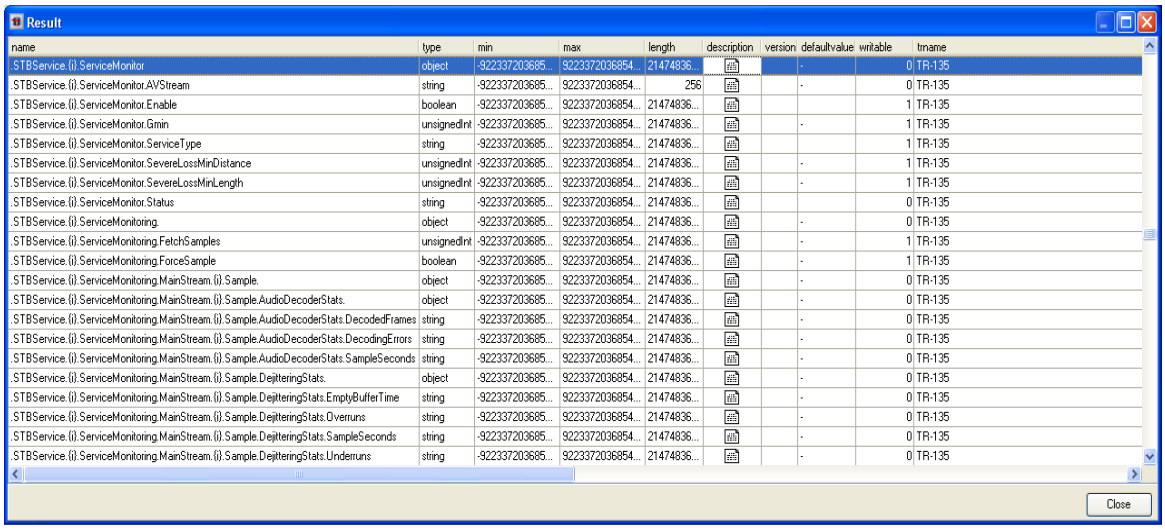

Figure 4-18: Example of datamodel table results

4.8. Summary

In this chapter provided we presented a prototype system implementation representing the two main entities of our architecture: control and management and the managed device. We used OpenACS as server that implements functionalities of the TR-069 auto-configuration server, and that allows advanced management functionalities by adding our logic in terms of scripts. We implemented TR-069 CPE that is conforming to the protocol specifications, and that carries out a complete communication with the server. Focus was made more on the feasibility and usefulness of the framework by implementing the TR-

069 client, using functionalities of OpenACS server, and possible scenarios (qualitative analysis) rather than measurements regarding multiple criteria (quantitative analysis). Our system prototype was tested according to the TR-069 CPE conformance and functionality test suites done in the interOperability Laboratory of University of New Hamsphire found in [52]. The tests have been carried out in parallel with implementation to check and validate parameter re-configuration.

Chapter 5

5. Conclusion and Future Work

5.1. Conclusion

In the current time where systems are evolving rapidly along with the technologies associated to them, it is crucial to invest efforts towards fulfilling the IBM autonomic vision to offer a wide possibility of management and control for equipments. Such autonomic management systems should communicate successfully with a variety of customer equipments regardless of their technical details regarding design and implementation. Therefore, standard compliance with management protocols is of great importance.

The goal of this thesis has been to develop a self-configuring framework for devices that provides reconfiguration based on dynamic changes occurring either to the device components or services offered by these devices. The main issue with current related device self-management work is that they provide extremely vague and abstract architectures without providing convincing implementation and/or testing details to assess system validity and usability. In this thesis, we present a general architecture for device self-management, and we give a general control loop implementation for device self-configuration using the Broadband Forum CPE WAN Management Protocol that mapped adequately with our system requirements.

The device self-configuration framework architecture presented within this thesis reinforced the autonomic concepts necessary to building self-managing systems. We based our framework architecture on the IBM MAPE-K control loop that we adapted to the context of device self-configuration. The main components of the system were explained in details. The framework requires the deployment of an autonomic manager that represents the management entity for managed devices. The autonomic manager comprises the system's intelligence within the control loop functional components.

Using the proposed framework architecture, application designers have the flexibility of choosing the degrees of intelligence of their systems by using different approaches in the implementation of the components. The four main components are: Device Monitoring, Device Data Analysis, Device Configuration Planning, and Device Configuration Execution. These components collaborate with the Knowledge component to provide autonomous properties to the system.

The Knowledge base constitutes a key factor in managing autonomic systems. It should have an ability of accurately describing the device models, all related environment information that affect their state, and rules that govern their operations. We described how a policy-based approach will fit in the system to translate the contextual information to a set of policies that will govern the user-personalized services, and provide an autonomous and seamless management of devices.

We have chosen an ontology-based approach to build our system knowledge given the significance of ontologies in modeling knowledge and storing context information. The ontology described was designed to enhance the capabilities of the system in providing a full view and understanding of the device components and capabilities. This allows the autonomic manager to provide the devices with continuous performance support and control to achieve high-level goals and keep the user satisfied.

To prove the validity of our system architecture, we have also developed a prototype system based on our proposal. The prototype represents the two main entities of our architecture: control and management and the managed device. We used OpenACS as a server that implements functionalities of the TR-069 auto-configuration server, and that allows advanced management functionalities by adding our logic in the form of scripts. We implemented TR-069 CPE that is conforming to the protocol specifications, and that carries out a complete communication with the server. Bed-tests have been carried out to prove the feasibility, usability, applicability, and validity of our system. The work done of this thesis has been accepted for publication in [1].

5.2. Future Work

There are several remaining issues that require closer investigation. Given the difficulty of the concepts introduced within our device self-configuration architecture, the device analysis and configuration planning components remain in need of deeper research, where more efficient algorithms should be developed. This represents a potential research perspective of this work.

Introducing a policy-driven autonomic device self-configuration would be convenient, in which reconfiguration decision making can be based on delivered policies and pertinent reasoning/learning of knowledge. Future directives would be to further develop the project by integrating other devices and services (such as IPTV service) in order to provide a wider variety of device and service management.

References

[1] Gu Xiaoyuan, J. Strassner, Xie Jiang, L.C. Wolf, T. Suda, "Autonomic Multimedia Communications: Where Are We Now?," *Proceedings of the IEEE* , vol.96, no.1, pp.143-154, Jan. 2008

[2] IBM, Autonomic computing initiative, 2001. [Online]. Available: http://www.research.ibm.com/autonomic/manifesto/autonomic_computing.pdf

[3] D.A. Menasce, J.O.Kephart, "Guest Editors' Introduction: Autonomic Computing," Internet Computing, IEEE , vol.11, no.1, pp.18-21, Jan.-Feb. 2007.

[4] E. Manoel, M. J. Nielsen, A. Salahshour, S. Sampath K.V.L., S. Sudarshanan, "Problem Determination Using Self-Managing Autonomic Technology", IBM Redbooks, June 2005.

[5] IBM, The autonomic computing edge: Can you CHOP up autonomic computing? 2008. [Online]. Available: http://www.ibm.com/developerworks/autonomic/library/ac-edge4/

[6] IBM, Autonomic computing and Web Services Distributed Management, 2005. [Online]. Available: http://www.ibm.com/developerworks/autonomic/library/ac-architect/

[7] N. Bicocchi, F. Zambonelli, "Autonomic communication learns from nature," *Potentials, IEEE* , vol.26, no.6, pp.42-46, Nov.-Dec. 2007Fff

[8] M.G. Hinchey, R. Sterritt, "Self-managing software," *Computer* , vol.39, no.2, pp. 107-109, Feb. 2006

[9] B. Jennings, S. van der Meer, S. Balasubramaniam, D. Botvich, M.O. Foghlu, W. Donnelly, J. Strassner, "Towards autonomic management of communications networks," *Communications Magazine, IEEE ,* vol.45, no.10, pp.112-121, October 2007

[10] UPnP, "Universal Plug and Play" [Online]. Available: http://www.upnp.org

[11] R. Kutschke, *Devices and services in UPnP, Bluetooth, TR-069, and OMA DM*, 2005. [Online]. Available: http://www.linecity.de/downloads/Dipl_arbeit_anhang_rk.pdf

[12] OMA-DM, "Open Mobile Alliance - Device Management" [Online]. Available: http://www.openmobilealliance.org/Technical/DM.aspx

[13] Mobile device management, "The Critical Role of Interoperability Standards for Mobile Device Management" [Online]. Available: http://www.innopath.com/pdf/mobile_device_management_standards.pdf

[14] J. Bernstein *et al*. “CPE WAN Management Protocol,” absorption in the earth’s atmosphere,” DSL Forum, Tech. Rep. TR-069, May, 2004

[15] M. Smith, C. Welty, and D.McGuinness, “OWL Web Ontology Language Guide. W3C Recommendation.”, W3C Recommendation, February 2004, [Online]. Available: http://www.w3.org/TR/2004/REC-owl-guide-20040210/

[16] Y. Al Ridhawi, A. Karmouch, “Ontology-Based Context-Level Agreements and Negotiation Protocol,” in *5th International Workshop on Next Generation Networking Middleware*, 2008

[17] I. Al Ridhawi, M. Aloqaily, A. Karmouch, N. Agoulmine, "A location-aware user tracking and prediction system," *Information Infrastructure Symposium, 2009. GIIS '09. Global* , vol., no., pp.1-8, 23-26 June 2009

[18] M. Strimpakou, I. Roussaki, M. Anagnostou, "A context ontology for pervasive service provision," *Advanced Information Networking and Applications, 2006. AINA 2006. 20th International Conference*, vol.2, April 2006.

[19] X. Wang, D. Zhang, T. Gu, H. Pung, "Ontology based context modeling and reasoning using OWL," *Pervasive Computing and Communications Workshops, 2004. Proceedings of the Second IEEE Annual Conference on* , pp. 18-22, March 2004.

[20] M. Ganna, E. Horlait, "Toward secure autonomic pervasive environments," *Global Telecommunications Conference, 2005. GLOBECOM '05. IEEE* , vol.2, no., pp. 6 pp., 28 Nov.-2 Dec. 2005

[21] B. Steinke, K. Strohmenger, "Advanced Device self Management through Autonomics and Reconfigurability," *Mobile and Wireless Communications Summit*, 2007. 16th IST , vol., no., pp.1-4, 1-5 July 2007

[22] E²R2, “End-to-End Reconfigurability Phase II” [Online]. Available: http://www.e2r2.motlabs.com

[23] V. Alonso, M.A. Fernandez, P. Davila, D. Gallegos, A. Castillo, E. Paniego, "New Self-* Paradigms for Managing Customer Networks," *GLOBECOM Workshops, 2008 IEEE* , vol., no., pp.1-5, Nov. 30 2008-Dec. 4 2008

[24] A.E. Nikolaidis, S.S. Papastefanos, G.I. Stassinopoulos, M.-P.K. Drakos, G.A. Doumenis, "Automating remote configuration mechanisms for home devices," *Consumer Electronics, IEEE Transactions* on , vol.52, no.2, pp. 407- 413, May 2006

[25] A.E. Nikolaidis, G.A. Doumenis, G.I. Stassinopoulos, M.-P. Drakos, M.P. Anastasopoulos, S. D'Haeseleer, "Management traffic in emerging remote configuration mechanisms for residential gateways and home devices," *Communications Magazine, IEEE* , vol.43, no.5, pp. 154- 162, May 2005

[26] A.E. Nikolaidis, S. Papastefanos, G.A. Doumenis, G.I.Stassinopoulos, M.P.K. Drakos, , "Local and remote management integration for flexible service provisioning to the home," *Communications Magazine, IEEE* , vol.45, no.10, pp.130-138, October 2007

[27] J.O. Kephart, D.M. Chess, "The vision of autonomic computing," *Computer* , vol.36, no.1, pp. 41- 50, Jan 2003

[28] H. Harroud, M. Ahmed, A. Karmouch, , "Policy-driven personalized multimedia services for mobile users," *Mobile Computing, IEEE Transactions on* , vol.2, no.1, pp. 16- 24, Jan.-March 2003

[29] N. Samaan, A. Karmouch, "Towards Autonomic Network Management: an Analysis of Current and Future Research Directions," *Communications Surveys & Tutorials, IEEE* , vol.11, no.3, pp.22-36, 3rd Quarter 2009

[30] S. Zhang, I. Cohen, M. Goldszmidt, J. Symons and A. Fox, "Ensembles of Models for Automated Diagnosis of System Performance Problems", *Dependable Systems and Networks, 2005. DSN 2005. Proceedings. International Conference on*, pp. 644–653, 2005.

[31] S. Duan, S. Babu, "Proactive identification of performance problems", in *SIGMOD '06: Proc. the 2006 ACM SIGMOD international conference on Management of data*, pp. 766–768, New York, NY, USA, 2006, ACM Press

[32] A. Ranganathan, R.H. Campbell, "Autonomic pervasive computing based on planning," *Autonomic Computing, 2004. Proceedings. International Conference on* , vol., no., pp. 80- 87, 17-18 May 2004

[33] E.Miller, *An introduction to RDF, online Computer Library Centter, Inc.,* Dublin, ohio, May 1998, [Online]. Available: http://www.dlib.org/dlib/may98/niller/05miller.htm.

[34] W3 Consortium, "Composite Capability/Preference Profiles (CC/PP)" [Online]. Available: http://www.w3.prg/TR/NOTE-CCPP/.

[35] T. Gruber. "A translation approach to portable ontology specifications." *Knowledge Acquisition*, 1993, [Online]. 5: 199-220. Available: http://tomgruber.org/writing/ontolingua-kaj-1993.pdf

[36] Broadband Forum [Online]. Available: http://www.broadband-forum.org/

[37] DSL Forum, "Data Model for TR-069 Enabled STB", Tech. Rep. 135, December 2007

[38] Dynamic Host Configuration Protocol, RFC 2131, March 1997

[39] Internet Group Management Protocol, RFC 3376, October 2002

[40] Real Time Streaming Protocol, RFC 2326, April 1998

[41] Broadband Forum CWMP Technical Reports [Online]. Available: http://www.broadband-forum.org/cwmp.php

[42] DSL Forum, "Internet Gateway Device data model for TR-069", Tech. rep.098, December 2006

[43] DSL Forum, "Data Model Template for TR-069 Enabled Devices", Tech. rep.106, September 2009

[44] DSL Forum, "Triple-play Services Quality of Experience (QoE) Requirements", Tech. rep.126, December 2006

[45] JCPE open source [Online]. Available: http://code.google.com/p/jcpe/

[46] Domj4 Library [Online]. Available: http://www.dom4j.org/dom4j-1.6.1/

[47] Apache Axiom Library [Online]. Available: http://ws.apache.org/axiom/

[48] Apache HTTPClient Library [Online]. Available: http://hc.apache.org/httpclient-3.x/

[49] DSL Forum, "Applying TR-069 to Remote Management of Home Networking Devices", Tech. rep.111, December 2005

[50] Xuggler API: Xuggle, "Xuggler API" [Online]. Available: www.xuggle.com/xuggler/

[51] OpenACS [Online]. Available: http://sourceforge.net/projects/openacs/

[52] Interoperability Laboratory of University of Hampshire, "Broadband Forum TR-069 CPE Functionality Test Suite", May 2007, [Online]. Available: http://www.iol.unh.edu/services/testing/tr069/testsuites/

Publications

[1] H. Rachidi, A. Karmouch, " A Framework for Self-configuring Devices using TR-069," To appear in *The 2nd International Conference on Multimedia Computing and Systems (ICMCS'11)*, Ouarzazate, Morocco, April 7-9, 2011

[2] I. Abdeljaouad, H. Rachidi, S. Fernandes, A. Karmouch, "Performance analysis of modern TCP variants: A comparison of Cubic, Compound and New Reno, " *Communications (QBSC), 2010 25th Biennial Symposium on* , vol., no., pp.80-83, 12-14 May 2010