LAISA CAROLINE COSTA DE BIASE

A computational architecture for organic and heterogeneous networks: the swarm operating system control plane

São Paulo

2015

LAISA CAROLINE COSTA DE BIASE

A computational architecture for organic and heterogeneous networks: the swarm operating system control plane

Text submitted to pursue the degree of Doctor of Sciences to the *Escola Politécnica da Universidade de São Paulo*

Area
Electronics Systems

São Paulo
2015

LAISA CAROLINE COSTA DE BIASE

A computational architecture for organic and heterogeneous networks: the swarm operating system control plane

Text submitted to pursue the degree of Doctor of Sciences to the *Escola Politécnica da Universidade de São Paulo*

Area
Electronics Systems

Advisor
Prof. Dr. Marcelo Knörich Zuffo

São Paulo
2015

Este exemplar foi revisado e corrigido em relação à versão original, sob responsabilidade única do autor e com a anuência de seu orientador.

São Paulo, _____ de _____ de _____

Assinatura do autor: _____

Assinatura do orientador: _____

Catalogação-na-publicação

De Biase, Laisa Caroline Costa
A computational architecture for organic and heterogeneous networks: the swarm operating system control plane / L. C. C. De Biase -- versão corr. -- São Paulo, 2015.
135 p.

Tese (Doutorado) - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Sistemas Eletrônicos.

1.Redes de computadores 2.Arquitetura de software 3.Arquitetura orientada a serviços 4.Middleware 5.Internet das coisas I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Sistemas Eletrônicos II.t.

This work is dedicated to my family.

ACKNOWLEDGEMENTS

I would like to express my deep gratitude to Professor Marcelo Knorich Zuffo, my research supervisor, for his patient guidance and enthusiastic encouragement of this research work. I would also like to thank to Professor Jan Rabaey for receiving me at the Swarm Lab and for all the time dedicated to our long discussions, providing guidance and valuable and constructive suggestions to the work. I would like to offer my special thanks to Professor Adam Wolisz for the fruitfull discussions about this work.

My grateful thanks are also extended to Mr. Max Rosan for his help with the code implementation, to Mr. Pablo Calcina, who helped me with the WSMX framework, to Ilge Akkaya and Nau Ozaki for their contributions with the Smart Jukebox demonstration development, to Mr. Rafael Sales who helped with the tests of the Control Plane in embedded systems and to Mr. Luis Gustavo Taboada who contributed supporting the Android application development to execute the Personal Agent. As well as to Professor John Kubiatiwicz, Professor Edward Lee and Professor John Wawrzynek for the inspirations from the Swarm OS group meetings.

I would also like to extend my thanks to the team at CITI-USP (Centro Interdisciplinar de Tecnologias Interativas da USP) and for the team at the Swarm Lab at UC Berkeley, for their friendship and support, especially to Maria Francesca Neglia and Ken Lutz.

Finally, I wish to thank my husband, Luiz Fernando, my parents, Marina and Claudio, my brothers, Rodolfo e Claudio, and my mother-in-law, Noemi, for their support and encouragement throughout my study.

This work was partially supported by: Associação do Laboratório de Sistemas Integráveis Tecnológico (LSI-TEC); Financiadora de Estudos e Pesquisas (FINEP) / FUNTTEL through the BRIPTV Project; Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) through the *Rede de Cooperação Universitária para Ensino Superior e Pesquisa Avançada em TV Digital e seus Cenários Interdisciplinares de Evolução* project; University of Sao Paulo; and Banco Santander.

ABSTRACT

Computational swarms, consisting of integrating smart networked sensors and actuators into our connected world, enable an extension of the info-sphere into the physical world. We call this extended cyber-physical info-sphere as the Swarm. This work proposes a Swarm vision with computational devices cooperating dynamically and opportunistically, generating organic and heterogeneous networks. This thesis proposes the computational architecture of the Swarm Operating System Control Plane that is the distributed software layer, embedded in all Swarm devices, responsible for managing Swarm resources, defining actors, how to describe and use services and resources, how to advertise and discover them, how to do transactions, content adaptation and multi-agent cooperation. The design of the architecture started with the review of the Swarm characterization itself, revisiting term definitions and establishing a terminology to be used. Requirements and challenges were identified and an operational vision was designed. This operational vision has been exercised with use case scenarios. The architectural elements were extracted from this vision and organized into an architecture that was tested against use cases, generating architectural reviews. Each of the architectural elements generated a state of the art review. A proof of concept of the framework was implemented and a demonstration was proposed and implemented. The selected demostration was the Smart Jukebox that exercises the distributed aspect and the dynamicity of the system. This work presents the vision of the emerging computing Swarm and presents a suitable framework. The evolution of this architecture may be the basis of a global heterogeneous and organic computer network leveraging cyber-physical systems to the cloud, and allowing the emergence of scalable and flexible systems to interoperate to achieve common goals.

Key words: Computer networks. Software architecture. Service oriented architecture. Middleware. Internet of things.

RESUMO

Computational Swarms (“enxames computacionais”), consistindo da integração de sensores e atuadores inteligentes no nosso mundo conectado, possibilitam uma extensão da info-esfera no mundo físico. Nós chamamos esta info-esfera extendida, cíber-física, de Swarm. Este trabalho propõe uma visão de Swarm onde dispositivos computacionais cooperam dinâmica e oportunisticamente, gerando redes orgânicas e heterogêneas. A tese apresenta uma arquitetura computacional do Plano de Controle do Sistema Operacional do Swarm, que é uma camada de software distribuída embarcada em todos os dispositivos que fazem parte do Swarm, responsável por gerenciar recursos, definindo atores, como descrever e utilizar serviços e recursos (como divulgá-los e descobrí-los, como realizar transações, adaptações de conteúdos e cooperação multiagentes). O projeto da arquitetura foi iniciado com uma revisão da caracterização do conceito de Swarm, revisitando a definição de termos e estabelecendo uma terminologia para ser utilizada. Requisitos e desafios foram identificados e uma visão operacional foi proposta. Esta visão operacional foi exercitada com casos de uso e os elementos arquiteturais foram extraídos dela e organizados em uma arquitetura. A arquitetura foi testada com os casos de uso, gerando revisões do sistema. Cada um dos elementos arquiteturais requereram revisões do estado da arte. Uma prova de conceito do Plano de Controle foi implementada e uma demonstração foi proposta e implementada. A demonstração selecionada foi o Smart Jukebox, que exercita os aspectos distribuídos e a dinamicidade do sistema proposto. Este trabalho apresenta a visão do Swarm computacional e apresenta uma plataforma aplicável na prática. A evolução desta arquitetura pode ser a base de uma rede global, heterogênea e orgânica de redes de dispositivos computacionais alavancando a integração de sistemas cíber-físicos na núvem permitindo a cooperação de sistemas escaláveis e flexíveis, interoperando para alcançar objetivos comuns.

Key words: Redes de computadores. Arquitetura de software. Arquitetura orientada a serviços. Middleware. Internet das coisas.

FIGURE LIST

TABLE LIST

ACRONYMS AND ABREVIATIONS

ADPDU	AVDECC Discovery Protocol Data Unit
AF	Application Formats
API	Application Programming Interface
AV	Audio and Video
AVB	Audio Video Bridging
AVDECC	AV discovery, enumeration, connection management and control
AVTP	Audio Video Transport Protocol
CAN	Controller Area Network
CoAP	Constrained Application Protocol
CoRE	Constrained RESTful Environments
CORBA	Common Object Request Broker Architecture
CRUD	Create, Retrieve, Update and Delete
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
DNS-SD	DNS Service Discovery
DTLS	Datagram Transport Layer Security
DTV	Digital Television
DIDL	Digital Item Declaration Language
DLNA	Digital Living Network Alliance
DTCP	Digital Transmission Content Protection
GDP	SwarmOS Global Data Plane
GENA	General Event Notification Architecture
GPRS	General Packet Radio Services
gPTP	Generalized Precision Time Protocol
HAN	Home Area Network
HAVI	Home Audio and Video Interoperability
HN	Home Network
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
HTTPU	HTTP/1.1 UDP
HTTPMU	HTTPU Multicast

ID	Identification
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
IoT	Internet of Things
IOPEs	Inputs, Outputs, Preconditions, and Effects
IP	Internet Protocol
ISO	International Organization for Standardization
JAR	Java ARchive
JID	Jabber ID
JSON	JavaScript Object Notation
LAN	Local Area Network
M2M	Machine-to-Machine
MAC	Media Access Control
MDF	Module Description File
Middleware	Intermediate software layer
MoCA	Multimedia over Coax Alliance
MPEG	Moving Picture Experts Group
OS	Operating System
OSI	Open Systems Interconnection
OSGi	Open Services Gateway Initiative
OWL	Web Ontology Language
OWL-S	Web Ontology Language for Services
P2P	Peer-to-Peer
PC	Personal Computer
PhD	*Philosophiae* Doctor
QoS	Quality of Service
REST	Representational State Transfer
RDF	Resource Description Framework
RMI	Remote Method Invocation
RTP	Real-time Transport Protocol
SAWSDL	Semantic Annotations for WSDL
SCP	Simple Control Protocol
SLA	Service-level Agreements
SOA	Service Oriented Architecture

SOAP	Simple Object Access Protocol
SRP	Stream Reservation Protocol
SSDP	Simple Service Discovery Protocol
TAO	The ACE ORB
TCP/IP	Transmission Control Protocol
TS	Transport Stream
TV	Television
UDDI	Universal Description, Discovery, and Integration
UDP/IP	User Datagram Protocol
UPnP	Universal Plug-n-Play
URI	Universal Resource Identifier
URL	Uniform Resource Locator
UPnP	Universal Plug and Play
VSCP	Very Simple Control Protocol
XML	eXtensible Markup Language
XMPP	eXtensible Messaging and Presence Protocol
W3C	World Wide Web Consortium
WSDL	Web Services Description Language
WSMO	Web Service Modeling Ontology
WSML	Web Service Modeling Language
WSMT	Web Service Modeling Toolkit
WSMX	Web Service Modeling eXecution environment

SUMMARY

1 INTRODUCTION

In the past years, computational devices have become faster, smaller, connected and cheaper. It is expected to have 50 billion connected devices by 2020 (CISCO, 2011) providing valuable information to consumers, manufacturers and utility providers. We call these pervasive computing devices working together a Computational Swarm, or just Swarm, which is allowing new scenarios composition leveraging a breakthrough in human-computer symbiosis (LICKLIDER, 1960). As an example, we can imagine a personal area network composed by sensors embedded in clothes or in the body, seamlessly cooperating with each other and with remote cloud systems in order to improve humans' quality of life. This work presents a computational architecture proposal to the Swarm Operating System Control Plane, leveraging global scale organic and heterogeneous networks.

1.1 SWARM CHARACTERIZATION

The Merrian-Webster dictionary[1] defines the word swarm as a body of aggregated bees, as in a hive, or as a great number of things or persons, especially in motion. Undoubtedly our pervasive computing scenario can be described as a great number of things in movement (adapting and interacting). On the other hand, the hive, or the body of bees, definition offers interesting opportunities of reasoning.

The computing swarm vision proposed by Rabaey (2011) builds on the belief that future information processing will be performed on a vastly distributed platform of collaborating components. The Swarm represents a much broader vision than the wireless sensor networks, potentially connecting trillions of sensory and actuating devices worldwide into a single platform abstraction. This enables the true emergence of concepts such as cyber-physical and cyber-biological systems, immersive computing, and augmented reality, and will have a tremendous impact in domains such as advanced healthcare, improved energy efficiency, environment-friendly living, mobility management, enhanced security, and many others.

Although in a hive there are only bees, there are different kinds of bees (queens, drones and workers), and they do cooperate in an organized way: each member works in cooperation with the others in order to achieve common goals (FRISCH, 1956). They work

[1] http://www.merriam-webster.com

cooperatively and communicate using a range of techniques, e.g. pheromones and dances. Besides, a same bee can act offering distinct services during its life, especially the worker bee, that can have different functions, such as honey sealing, drone feeding, queen attending and honeycomb building. Thus, we could consider that it is a heterogeneous system, even though there are all bees, as well as in the Swarm, where there are devices with computing and communication systems, equal in its core, but each one with a different capability, offering different services to be composed to achieve a common goal.

Dorigo (2009) defines swarm intelligence as the discipline that deals with natural and artificial systems composed of many individuals that coordinate using decentralized control and self-organization. In particular, swarm intelligence focuses on the collective behaviors that result from the local interactions of the individuals with each other and with their environment. The characterizing property of a swarm intelligence system is its ability to act in a coordinated way without the presence of a coordinator or of an external controller. The swarm intelligence discipline studies many bio inspired algorithms, such as: ant and bacterial colony optimizations, bee and bats algorithms.

A concurrent term to the Swarm is the Internet of Things (IoT) or the Internet of everything. IoT has become a buzzword being used in many contexts, but a definition that is often used is “technologies and research disciplines that enable the Internet to reach out into the real world of physical objects”. This nomenclature downside is the implication of having the Internet as glue. The Swarm envisions a heterogeneous and organic network, counting on machine-to-machine communications, on gateways facilitated intranets, as well as on Internet communications. It couples with the swarm intelligence term, as well, in which many individuals cooperate using decentralized control and self-organization. Hence, we will adopt the term Swarm, in order to designate our scenario. The Swarm term envisions an embedded centric approach, where each device has its own processing power and specialized set of features that can be combined in order to achieve a goal.

The Swarm vision leverages the advancement of the pervasive computing scenarios, looking forward the set up of a framework that allows cooperation between heterogeneous devices. It would extend the cloud computing vision including smart objects to the network in a seamless way, dematerializing devices boundaries and creating a dynamic and auto-adaptable system. This dematerialization can be achieved by the fusion of service oriented architecture with the cyber-physical computing, allowing devices to offer its resources as services to explore the system synergies. The Computational Swarm is a network of

heterogeneous embedded devices that organic and opportunistically can perform complex tasks synergistically.

- **Heterogeneous**: multiple devices with different functions, complexity, processing and communication capabilities, operating systems, etc.;

The organic and opportunistic characteristics could be mapped in some more concrete requirements:

- **Distributed**: there are no central entity controlling the swarm, it is intrinsically distributed;
- **Autonomous**: does not depend on a human operator to work, to maintain and to set-up;
- **Cooperative**: devices work synergistically, through resources sharing, allowing the achievement of common goals;
- **Dynamic**: the swarm is organic, allowing dynamicity where the heterogeneous devices adapt to the context, cooperating to the available devices at the moment. Using semantic computer, it will allow for improving the descriptions and commands on the swarm environment reducing the standardizations constraints;
- **Scalable**: this requirement refers to the adaptability of the system, being able to infer knowledge, being less dependent from standards.

In addition, in order to think about a common framework, it is essential to have communications flexibility. An organic communication is peer to peer, but we cannot neglect the Internet and Local Area Networks available for communications. On the other hand, to have a common platform, it should be flexible regarding communications standards, supporting gateway implementations. Thus, two more requirements could be added:

- **Virtualized**: this heterogeneous system will allow interoperability using gateways / adaptors. It will allow, for example, the composition of virtual networks that connects physical networks through these gateways;
- **Wide communication scope**: refers to global and local communication capability, the swarm should be capable of allowing services cooperation in a global way, allowing from machine-to-machine up to over the Internet interoperation;

The main challenges to implement the Swarm ecosystem are the interoperability and scalability. Given the heterogeneity of devices, network technologies and applications, connecting from resource-constraint embedded devices to high-processing-power general-

purpose computers, the optimized solution for some scenarios are not optimal to others. It generates difficulties to establish a consensus on a common open communication system.

The need of standards to define this communication creates a lack of flexibility, limiting the system scalability. The most commonly used definition of flexibility is “the ability to change or react with little penalty in time, effort, cost, or performance” (UPTON, 1994). As the standards require a process that demands time to have changes published, standardization flexibility is highly desired.

In order to allow interoperability some current solutions rely on complex user configuration with low-level control and commands, as well as, on closed systems. This is a challenging area that has been receiving many efforts over the years. The first commercial systems were closed and high cost, they required specialized labor to do the initial setup and to do maintenances. Newer systems have arisen, such as the DLNA (Digital Living Network Alliance) (DLNA, 2014) that is open, cheaper and easier to use. Nevertheless, it lacks of usability and flexibility. There is a mismatch between user desires and the available resources – there is a long way for mapping desires, goals, plans, execution and binding.

Challenges to the Swarm are the technologies complexity due to the multiple fields of knowledge involved: the Swarm framework and applications design require the use of different technologies, such as networking and communications, computer learning, embedded computing, services oriented architecture, semantic computing, grid computing, sensor network, cloud computing, among others.

The semantic localization is an interesting feature because many of the user goals are related to some geographic location, that are not coincident to computer defined domains, such as a room in a house. Simple solutions to define the semantic of devices location will require user configurations, what is against our usability principle. Finally, the final challenge is to design a robust open architecture with security and dynamically available resources that can leverage more powerful, capable and resilient systems enabling applications that have not been realized yet.

In addition, there are the unpredicted Swarm requirements. Although designers can foresee many scenarios, real requirements about user desires and concerns are difficult to define, given the current society values dynamism.

1.2 HYPOTHESIS

Heterogeneous networked computational devices can behave like Swarms, they can auto-configure themselves to perform tasks synergistically and organically.

1.3 JUSTIFICATION

Smart networks are leveraging multiple projects and standards. This theme is a good opportunity to technologic and scientific research and development. The research in Swarm Technologies allows a wide range of applications with huge impact potential in society. Examples of application areas are:

- **Smart cities**: a swarm-enabled "Smart City" helps to run the city infrastructure more effectively and empowers its occupants by providing more effective interfaces and better mobility. In Smart Cities, the Swarm framework would make possible to aggregate information from multiple sources, identify individuals who can benefit from information that has been gathered, and notify them using local resources such as cell phones, nearby displays, or audio systems. For example, maintenance crews may recruit sensors from underground utilities, and combine that sensor data with data from pipe-crawling robots combining this information to guide maintenance operations using overlay displays (LEE et al., 2012).
- **Natural resources consumption efficiency**: this technology is synergic to the smart grid; supporting a smarter distribution, automation and monitoring systems; and extending the benefits to management systems for in-home and in building use. It will help consumers monitor their own usage and adjust behaviors, regulating devices to automatically operate during off-peak energy hours and connect to sensors to monitor occupancy and lighting conditions, configuring the lighting, heating, laundry times and domotic systems in general focusing in economy and efficiency.
- **Health**: the Swarm can bring huge benefits to healthcare, where it may improve access, increase the quality of care and reduce the cost of care. Hospitalized patients can be constantly monitored using IoT-driven, noninvasive monitoring. Employing sensors to collect physiological information and gateways and the

cloud to analyze and store the information and then send the analyzed data wirelessly caregivers for further analysis and review. It replaces the process of having a health professional come by at regular intervals to check the patient's vital signs, instead providing a continuous automated flow of information. In this way, it simultaneously improves the quality of care through constant attention and lowers the cost of care by eliminating the need for a caregiver to actively engage in data collection and analysis (NIEWONLY, 2013). The technology may be used for remote monitoring too, improving quality of life and prevention of health complications. Huo et al. (2009) presents a remote monitoring system to the elderly, where two sensor networks monitor the person and the home environment respectively. It allows understanding whether the person is doing his or her usual activities. An important capability is the identification of critical situation and the generation and transmission of alerts with reliability and privacy concerns (SURIE; LAGUIONIE; PEDERSON, 2008; SUH; KO, 2008).

- **Home networking**: the Swarm can be applied to the domestic environment too, improving comfort, safety, convenience and efficiency in homes. The home remote control may be done using the Internet as well as cell phone networks, as presented by Silva et al. (2008) and by Hornsby, Belimpasakis and Defee (2009). The home networking is revolutionizing the entertainment too, blurring the boundaries between devices, and allowing contents to be exchanged seamlessly between devices (CHENG; KUNZ, 2009). Important features are: search, localization, sharing, distributed storage, streaming, playing and transcoding.
- **Accessibility**: over a billion people are estimated to be living with disability. The lack of support services can make handicapped people overly dependent on their families, which prevents them from being economically active and socially included. The Swarm can improve people with disabilities quality of life and their independence. In Costa, Almeida and Zuffo (2013) an accessible interface was developed to allow people with disabilities to control the domestic environment. In Taleb et al. (2009), a home networking framework was developed to assist elderly people in home – this work focus in visual impairment and in fell-down monitoring.

- **Disasters monitoring and prediction**: in emergency scenarios, the swarm-enabled Smart City is able to safely and securely align both stationary (e.g. biohazard detection sensors) and mobile (e.g. Unmanned Aerial Vehicles and robots) resources needed to protect itself and its inhabitants. Environmental sensors will focus on detecting and alerting inhabitants of dangerous chemical and biohazards, while immersive environments created on the fly can enable teams to deploy cleanup and security forces. The goal of the network under emergency conditions is to adapt, coordinate, respond, and resolve dangers appearing in the environment effectively, efficiently, and as autonomously as possible. It may automatically reroute traffic and identify health and safety threats, such as those created by an earthquake or by a terrorist attack (LEE et al., 2012).

1.4 OBJECTIVES

This work aims to investigate and develop a flexible and scalable computational architecture that dynamically and opportunistically integrates heterogeneous networked devices leveraging the interoperability among them so they can synergistically and organically auto configure themselves to perform complex tasks.

The specific goals are:

- Investigate and present the theoretical foundations and the state of the art of the Swarm;
- Consolidate a computational architecture to the Swarm OS Control Plane;
- Demonstrate the functional viability and to evaluate the Swarm Control Plane proposal by developing a proof of concept.

1.5 METHODOLOGY

The seamless and consistent execution of a broad range of applications on the emerging and evolving Swarm platform requires a framework – which we call the Swarm OS (Operating System) – that helps to construct applications from a multitude of functions that are running on distributed modules, while ensuring that resources are dynamically balanced.

Accomplishing the latter is the function of the Control Plane of the Swarm OS, whose architecture is the central contribution of this thesis. The design of the architecture used a top-down approach. It started with the review of the Swarm characterization itself, revisiting term definitions and establishing a terminology to be used. Requirements and challenges were identified and an operational vision was designed. This operational vision has been exercised with usage scenarios. The architectural elements were organized in architecture, and each of them generated a state of the art review. A proof of concept of a proposed application was developed. The architecture has been revisited during the proof of concept analysis phase as shown in Figure 1.

The proof of concept was developed applying the architecture to implement a smart jukebox. The jukebox is an automatic music composer. It identifies the user that approaches an area of interest, and based on the user's music preferences, it composes a new song dedicated to the user. The audio transmission is over Ethernet, with quality of services guarantee – time synchronization and bandwidth reservation. Essential requirements of the Swarm framework exercised on this proof of concept are: time synchronization, resource management and allocation, context awareness and multi-agent systems. It also uses the Audio Video Bridging (AVB) technologies for network resources management and time synchronization, image sensors for face recognition (in order to provide Personas retrieval) and factor oracles for varied realization of musical structures.

Figure 1 - Smart Jukebox Swarmlet Illustration

Source: Author

The composer and the face recognition processes integrated to the smart jukebox were developed by other researchers and are not part of this thesis contribution. The face recognition code was developed by Mr. Nau Osaki using the Open CV library and programmed using C++. The composer was developed by Ms. Ilge Akkaya and was

developed using Ptolemy II. The integration of these codes to the smart jukebox example required development and was done under the scope of this thesis. The remainder of the proof of concept was developed using the Java Programming Language. The proximity identification was done using Personal Agents hosted on Android OS devices. More details about the proof of concept implementation will be presented in item 4.2.

Flow summary:

- State of the art review;
- High level architecture proposal;
- Segmented state of the art review;
- Architecture detailing;
- Proof of concept specification;
- Proof of concept implementation;
- Results analysis.

Given the importance of this topic, an internship with international collaboration was considered relevant. Thus, the candidate engaged herself with the TerraSwarm Research Center and the Swarm Lab at the University of California at Berkeley, where important discussions where held, improving this work aggregated value.

1.6 THESIS STRUCTURE

This thesis has five chapters:

- Chapter 1: Introduction – presents the context, justification, goals, methodology and the document outline;
- Chapter 2: State of the art – this chapter investigates the fundamental problems in the field and the related works;
- Chapter 3: The Swarm OS Control Plane – this chapter presents the Terminology proposed, Swarm OS Control Plane architectural proposal and use cases description;
- Chapter 4: Proof of concept – presents the proof of concept description and its analysis;
- Chapter 5: Conclusion – this chapter presents the summary of the result analysis, contributions of the work, comments and future work.

2 STATE OF THE ART

Considering the Swarm principles previously presented, related technologies will be described and discussed in this chapter, these technologies are: service-oriented architecture, RESTful web services, agent-based computing, semantic computing and middlewares for distributed networks.

In order to achieve a cooperative and granular system, the Service-Oriented Architecture (SOA) approach will be applied. SOA organizes computational functionalities as components that can be connected composing more complex behaviors. In order to have a distributed and dematerialized system, with a wide communication scope, web services will be considered in the Swarm OS. Web services allow the application of SOA through computer networks. RESTful web services will make possible to have heterogeneous environments, establishing a simple communication protocol combined with document passing messages for web services communications. The REST resources orientation approach, leverages virtualized and dematerialized principles as well. Semantic Computing will provide the organic behavior to the system, allowing adaptation and no pre-programmed cooperation. Finally, applying a multi-agent approach will provide an autonomous and dynamic behavior.

The Swarm OS Control Plane is a middleware for distributed networks, so we are presenting and discussing some of these technologies. Semantic middlewares were presented and discussed as well.

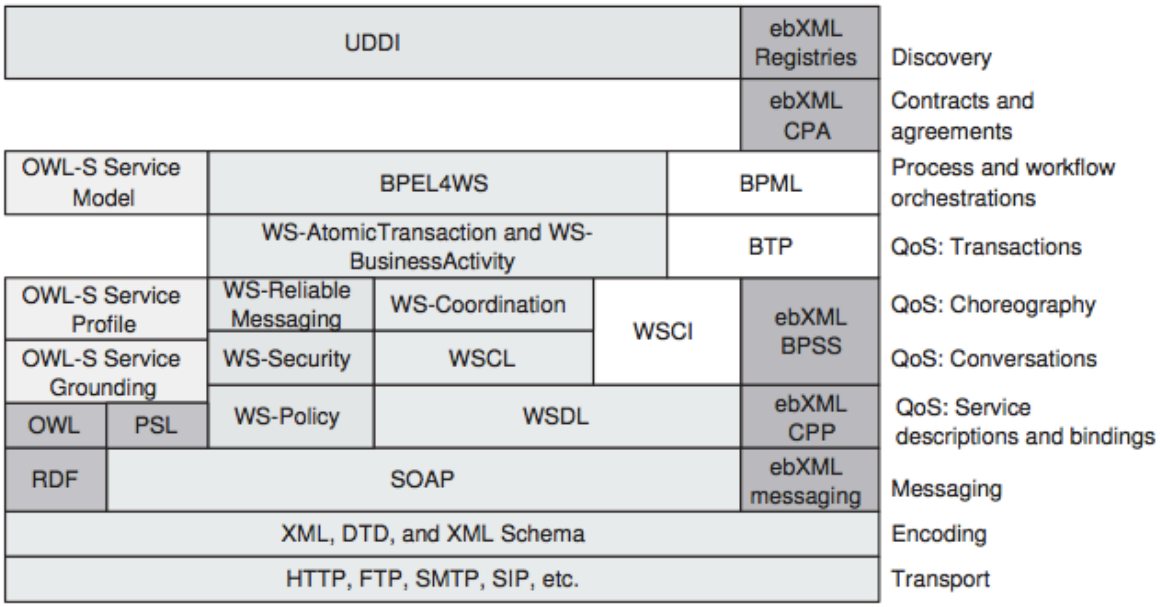
2.1 SERVICE-ORIENTED ARCHITECTURE

The Service-Oriented Architecture (SOA) is a software design pattern based on the dynamic selection of software pieces that provides functionalities as services to other applications. Services provide higher-level abstractions for organizing applications in large-scale, open environments, based on quality-of-service criteria that each party can customize for itself. Moreover, these abstractions are standardized, what enables the interoperation. SOA impose the following requirements (SINGH and HUHNS, 2005), (BOOTH et al., 2004):

- **Loose coupling and implementation neutrality:** it means that the interface matters, not the details of the implementations of the interacting components. It can be called as message orientation: the service is formally defined in terms of the messages exchanged between provider agents and requester agents.

- **Flexible configurability**: different components are bound to each other in run time, changing dynamically.
- **Coarse Granularity**: low dependency and few messages of greater significance between the participants.
- **Teams**: computation is realized as autonomous parties cooperating.
- **Logical view**: the service is an abstracted, logical view of actual programs, databases and business processes.
- **Description orientation**: the service is described by machine-processable metadata containing only the details important for the use of the service.
- **Network orientation**: Services tend to be oriented toward use over a network, though this is not an absolute requirement.

Figure 2 – The relationship of standards for automating electronic business



Source: SINGH and HUHNS (2005)

Singh and Huhns (2005) proposed the architecture presented in Figure 2 that presents the SOA layers and the related standards relationship. The stacks make use of the following abstraction levels:

- **Transport**: provides the fundamental protocols for communicating information among the components in a distributed system of services;
- **Encoding**: standards at this level describe the grammars for syntactically well-formed data and documents, and how to validate them;
- **Messaging**: describes the formats and document templates;

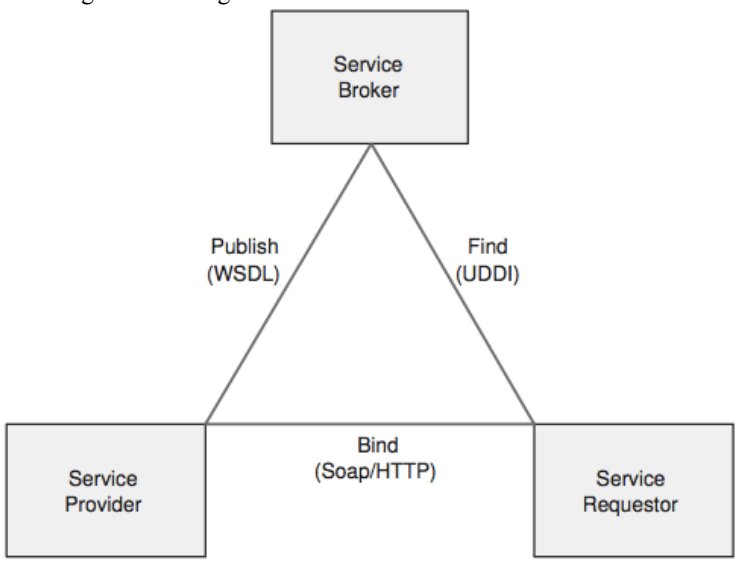
- **Service descriptions and bindings**: describe the functionality of Web services in terms of their implementations, interfaces, and results;
- **Conversations**: it is an instance of a protocol of interactions among services, describing the sequences of documents and invocations exchanged by an individual service;
- **Choreography**: coordinates collections of Web services into patterns that provide a desired outcome;
- **Transactions**: specify behavioral commitments of the autonomous components, but also the means to rectify failures;
- **Process and workflow orchestration**: specify the control flows and data flows needed for composite services to be executed correctly;
- **Contracts and agreements**: formalize commitments among autonomous components in order to automate electronic business and provide outcomes that have legal force and consequences;
- **Discovery**: specifies the protocols and languages needed for services to advertise their capabilities and for clients that need such capabilities to locate and use the services.

Current implementations of web services emphasize a single provider offering a single service to a single requester. This is in keeping with a client-server architectural view of the Web. On the other hand, another important trend relevant to the Swarm is peer-to-peer computing (P2P). The Web today is used for interactions in which most of the information resides on one side (server) and most of the intelligence on the other (client). In P2P different components are peers, each one has aspects of being a server and a client. Guinard et al. (2010) presented a discussion about how to connect SOA concepts with the Internet of Things.

The architecture for Web services is founded on principles and standards for connection, communication, description, and discovery. Service providers, who create Web services and advertise them to potential users by registering the Web services with service brokers. Service brokers, who maintain a registry of advertised (published) services and might introduce service providers to service requesters. Service requesters, who search the registries of service brokers for suitable service providers, and then contact a service provider to use its services.

For providers and requesters of services to be connected and exchange information, there must be a common language. A common protocol is required for systems to communicate with each other, so that they can request services, such as to schedule appointments, order parts, and deliver information. The Simple Object Access Protocol (SOAP) (MITRA and LAFON, 2007) currently provides a common communication protocol. The services must be described in a machine-readable form, where the names of functions, their required parameters, and their results can be specified. SOAP uses Web Services Description Language (WSDL) to describe services that uses eXtensible Markup Language (XML). Finally, clients – users and businesses – need a way to find the services they need. This is provided by Universal Description, Discovery, and Integration (UDDI), which specify a registry or "yellow pages" of services. Besides standards for XML, SOAP, WSDL, and UDDI, there is a need for broad agreement on the semantics of specific domains. In fact, that is where the deeper challenges lie.

Figure 3 – The general architectural model for Web services



Source: SINGH and HUHNS (2005)

The service-oriented architecture is being pointed out to be used in Internet-of-Things frameworks by some works. Singh, Tripathi and Jara (2014) present a survey where the Service Oriented Architecture is used in IoT middlewares for access to heterogeneous sensor resources.

2.1.1 RESTful web services

SOAP web services support remote procedure calls using XML documents in a generic way, being flexible to define any commands. On the other hand, a simpler strategy for web services that is currently more popular is to use the REST (REpresentational State Transfer) approach. REST is a programming style that applies the following constraints: replicated repository, cache, client-server, layered system, stateless, virtual machine, code on demand, and uniform interface (FIELDING and TAYLOR, 2002). An important concept in REST is the existence of resources referenced with a global identifier. In order to manipulate these resources, the communication uses a standardized interface and exchange representations of these resources (via documents). The RESTful interface takes advantage of the HTTP commands and applies them to resources (things pointed by URIs) using the REST architectural style. RESTful uses the CRUD (Create, Retrieve, Update and Delete) basic methods, designed to operate persistent storage mapping them to the HTTP methods (GET, PUT, POST, and DELETE) (HELLER, 2007).

RESTful service implementations are more lightweight, since it uses standard Web technologies such as URI, HTTP and JSON, in contrast to WS-* services which are based in XML and RPC. This makes the RESTful approach of particular importance for resource-constrained services. Some previous works (FIELDING and TAYLOR, 2002; WILDE, 2007; and GUINARD et al., 2011) envisioned the advantages of applying RESTful architectural principles and the World Wide Web technologies in order to represent the things in the IoT, calling this approach the Web of Things.

In order to fully interact with devices using URIs, two additional elements are defined in REST besides operations (GET, POST, PUT and DELETE): content-negotiation, which is the capability of choosing the response format based on weights; and status codes which are a standardized way of notifying about exceptions in the main execution flow.

Although RESTful web services are simple in concept, HTTP is usually used over TCP/IP what is not suitable for constrained networks and devices, especially wireless battery power ones (SHELBY, 2010). Modern HTTP servers, clients and proxies, may employ optional headers increasing its complexity and the overhead. HTTP has evolved into a highly complex protocol as used between modern servers and browsers. In sensor networks wireless nodes are typically sleeping over 90 percent of the time, making the HTTP request/response pull model inappropriate. TCP has performance problems over lossy links, sensitivity to

mobility, it doesn't support multicast and has high overhead for short-lived transactions (EGGERT, 2010).

The Constrained RESTful Environments (CoRE) IETF (Internet Engineering Task Force) Working Group developed CoAP (Constrained Application Protocol) that aims at realizing the REST architecture in a suitable form for the most constrained nodes and networks, considering machine-to-machine applications. CoAP compacts the HTTP protocol and offers features for M2M such as built-in discovery, multicast support and asynchronous message exchanges. CoAP defines an UDP (POSTEL, 1980) binding with optional reliability (with exponential back-off) supporting unicast and multicast requests, support to request actions (GET, PUT, POST and DELETE) over resources using URI, simple proxy and caching capabilities, a stateless HTTP mapping and security binding to Datagram Transport Layer Security (DTLS) (RESCORLA; MODADUGU, 2012).

2.2 MULTI-AGENT SYSTEMS

Agent-based Computing (JENNINGS, 1999) is a large and widely spread scientific domain. An agent can range from a "software agent" or "service/daemon", to an intelligent agent, which is based on models of artificially intelligent behavior (WOOLDRIDGE, 2009; WEISS, 1998).

Booth et al. (2004), defines an agent as the concrete implementation of a Web service. The agent is the concrete piece of software or hardware that sends and receives messages, while the service is the resource characterized by the abstract set of functionality that is provided. Singh and Huhns (2005) define an agent as an active computational entity that has a persistent identity, that can perceive, reason about, and initiate activities in its environment and can communicate with other agents, including humans. Because services are best modeled as being autonomous and heterogeneous, they can be naturally associated with agents. From the point of view of an agent, if there are other agents that can affect its environment and of which the agent is aware, then the environment is considered to be multi-agent.

From an implementation standpoint, environments for agents consist of: a communication infrastructure and protocols for interaction, security services for authentication and authorization, remittance services for billing and accounting, and operations support for logging, recovery, and validation.

The state of the environment according to the agent can be termed as its knowledge or a set of its beliefs. An agent's desires correspond to the state of the environment the agent prefers. And, an agent's intentions correspond to the state of the environment the agent is trying to achieve, which should be a consistent subset of the agent's desires and directly connected to the agent's actions. The agent must infer the beliefs, desires, and intentions of other agents in order to cooperate or compete with them.

2.3 SEMANTIC COMPUTING

Semantic computing is a field that addresses the derivation and matching of the semantics of computational content and that of naturally expressed user intentions to help retrieve, manage, manipulate, or even create contents (SHEU et al., 2010). It uses ontologies to represent a knowledge domain, defining concepts and the relationship between them. It generates a vocabulary that has meaning to computers, supporting automatic reasoning. Figure 4 presents the layers to connect content and the user:

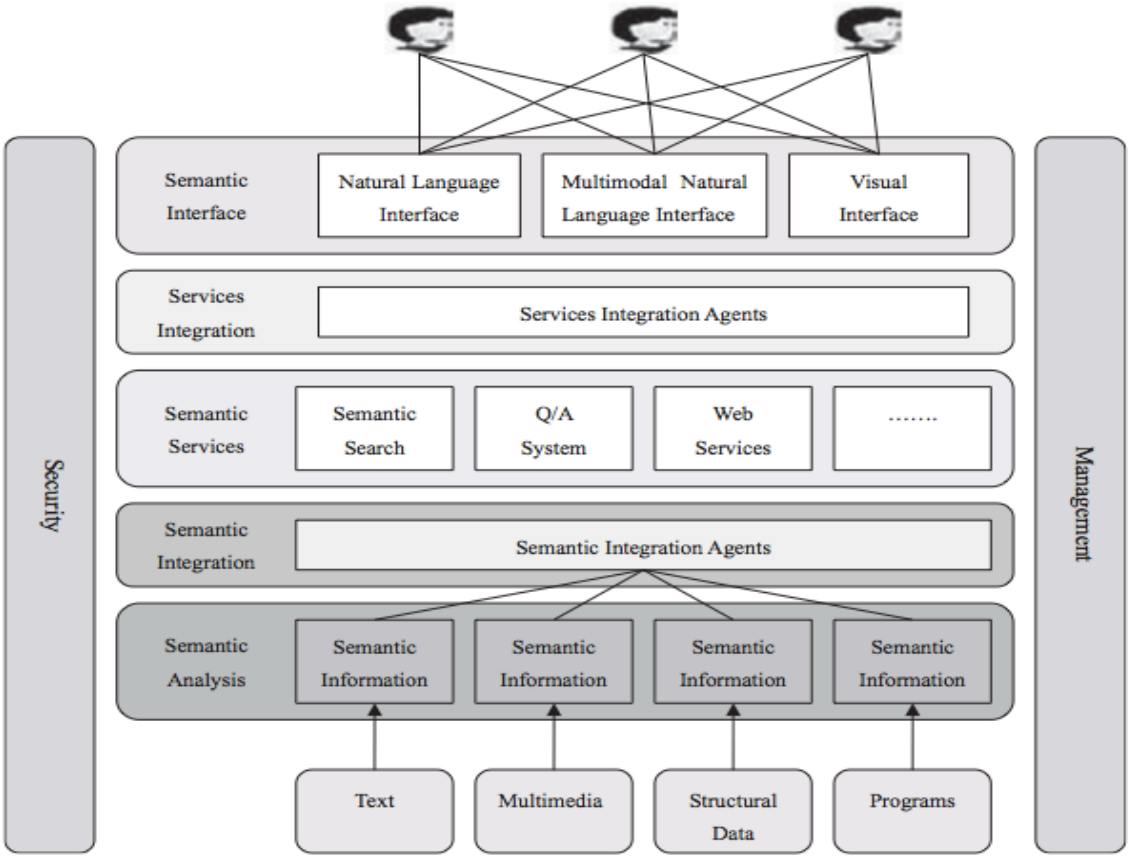
- **Semantic analysis**: analyzes content to convert it to a given description;
- **Semantic integration**: integrates content/semantics from multiple sources;
- **Semantic services**: utilizes content and semantics to solve problems;
- **Service integration**: integrates services to provide more complex services;
- **Semantic interface:** interpret naturally expressed user intentions.

Ontology is a kind of knowledge representation describing a conceptualization of some domain. It is often captured in some form of a semantic network — a graph whose nodes are concepts or individual objects and whose arcs represent relationships or associations among the concepts. It specifies a vocabulary including the key terms, their semantic interconnections, and some rules of inference.

The Semantic Web is an effort to express information in the World Wide Web using formal semantics in order to ease its automated processing, minimizing human intervention (LASSILA, 2005). Semantic web services use the semantic web to extend the web services, in order to automate its discovery, execution, composition, and interoperation (MCILRAITH; SON; ZENG, 2001). In order to embed semantic information in web services, specific standards have arisen – such as RDF-S, OWL-S and WSDL. The semantic web was originally developed to address client-server systems, allowing business to offer services to people and

the semantic information used to improve searching mechanisms. The Swarm vision requires machine-to-machine communications.

Figure 4 - Semantic computing architecture proposal



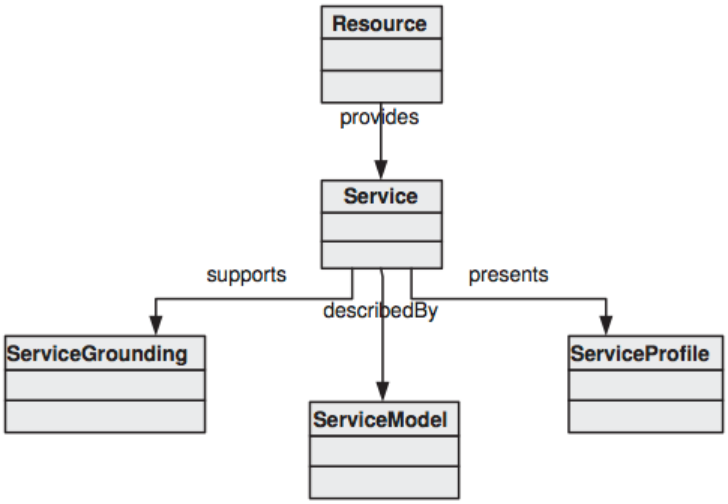
Source: SHEU et al. (2010)

The two main ontology standards currently developed for services are the Web Ontology Language for Services (OWL-S) (SMITH et al., 2004; MARTIN et al., 2004) and the WSMO (Web Service Modeling Ontology) (ROMAN et al., 2005). They were developed to enable reasoning about web services, planning compositions of web services, and automating the use of services by software agents. Another option is the use of the Web Services Description Language (WSDL) (BOOTH; LIU, 2007) associated with the Semantic Annotations for WSDL (SAWSDL) (FARREL; LAUSEN, 2007; KOPECKY et al., 2007).

The OWL-S is a set of ontologies that describes services, implemented in OWL language that uses XML. An OWL-S description for a service consists of three components, as presented in Figure 5: service profile, service model, and service grounding. The service profile defines the service, using its IOPEs (Inputs, Outputs, Preconditions, and Effects). The service model provides a description of how and when to interact with the service, via the

sending and receiving of messages. A service behavior is described in terms of its sub processes, where each sub process can be described by its IOPE and is linked to other sub processes by control constructs (such as: sequence, split, split and join, choice, iteration, and if-then-else). The service grounding describes how to access and use a service. It includes descriptions for message formatting, transport mechanisms, protocols, and serializations of types. The OWL-S service-grounding model builds upon WSDL. An OWL-S atomic process corresponds to WSDL operation, and inputs and outputs correspond to WSDL messages.

Figure 5 - OWL-S Class Diagram



Source: SINGH and HUHNS (2005)

An example that represents the IOPE concept is: "… an online bookstore could have as inputs the book title, customer's address, and customer's credit card number, with the precondition being the validity of the credit card. The output would be an electronic receipt, and the effects would be a debiting of the credit card and the shipping of the book with a corresponding transfer of ownership. The functional attributes might describe the quality of the service, such as the bookstore's average time-to-ship and the fee for an order." (SINGH; HUHNS, 2005)

The WSMO is composed by four sub ontologies:

- **Ontology**: describes the underlying concepts associated to the web service domain.
- **Web Services**: describes the web service details, the capabilities and interfaces. The capabilities define the preconditions and post conditions of the service. The interface defines how the functionality can be achieved including the

choreography (interaction with the web service) and orchestration (functionality required from other services).

- **Mediator**: its purpose is to resolve the heterogeneities between ontologies, there are four types of mediators: goal-goal, web service-web service, ontology-ontology and web service-goal.
- **Goal**: expresses the objective of the service consumer: this information will be used to look for a service.

OWL-S takes a service point of view to describe service activities while WSMO takes a client point of view to describe the consumer's goals – the client may be either a human user or an agent acting on behalf of a user. The main difference between the two approaches is that OWL-S does not separate what the user wants from what the service provides. The Profile ontology is used as both an advertisement for the service and as a request to find a service. In WSMO, a Goal specifies what the user wants and the Web service description defines what the service provides through its capability. Other differences are that WSMO defines vocabularies for non-functional properties (Dublin Core and Friend of a Friend), but OWL-S doesn't standardize it. OWL-S defines only one way to interact with the service and WSMO allows the definition of multiple interfaces for a single service. Finally, WSMO provides a family of layered logical languages which enables to combine conceptual modeling with rules on only the required expressiveness level, allowing descriptions to stay more explicitly within efficiently computable fragments. OWL-S can be considered more mature in certain aspects, including the choreography and grounding specifications. However, WSMO provides a more complete conceptual model as it addresses aspects such as goals mediators (KOPECK, 2005). In addition, the WSMO project has a broader scope than OWL-S, it not just specifies the ontology to model a web service, but defines a whole framework with its own language, execution environment and development tool. The Web Service Modeling Framework (WSMF) (DE et al., 2011) includes:

- **WSML**: a language for describing ontologies, its syntax is based in F-Logic (BAADER, 2003) and Description Logic (KIFER; LAUSER, 1989). It is comparable to OWL.
- **WSMO**: a set of ontologies that aim to describe all necessary aspects of a service in order to facilitate services discovery, invocation and composition, comparable to OWL-S. WSMO inherits the concept of URI (Universal Resource

Identifier) for unique identification of resources and adopts the concept of Namespaces for denoting consistent information spaces.

- **WSMT**: a design tool based on Eclipse that serves to describe a service using WSML, comparable to Protégé associated with the OWL-S Editor.
- **WSMX**: an execution environment for the framework. WSMX currently does not implement the entire WSMO specification but serves as a reference implementation, being a good start point to further development. It is based on SOAP web services.

The SAWSDL, different from the others, is based on an annotation approach that consists on enriching and supplementing the service description and is restricted to functional aspects. It extends WSDL, defining new attributes and a XML schema definition language. SAWSDL is independent from the semantic description language, being possible to use WSDL or OWL, for example. In SAWSDL the semantic annotations are related to the interface, operation, input and output elements (BITAR; BELOUADHA; ROUDIES, 2013).

2.4 MIDDLEWARES FOR DISTRIBUTED SYSTEMS

Middlewares are software layers that are used to content, application and services interoperation in distributed and heterogeneous systems. *Middleware* is a generic term used to designate the software layer that intermediate the applications execution over operating systems. The middleware shall facilitate the applications development and to facilitate the integration of systems.

Many generic middlewares have been developed in distributed heterogeneous computing: CORBA (NARASIMHAN, 2007; PARK, 2005) and OBIWAN (FERREIRA; VEIGA; RIBEIRO, 2003) are examples. CORBA allows the use, through a computer network, of programming objects that are running in different programs (running in the same machine as well as in different ones). It creates abstractions to objects using a standardized declarative language that defines how to use the object and its features. CORBA allows the interaction between objects implemented in any language if this declarative language is being used. OBIWAN is similar but adds some extra features, such as a distributed *Garbage Collector* and a security policy language.

This kind of middleware suits well to the scenario where one application is running in multiple devices. It differs from the swarm-computing scenario, where there is a set of devices, from different manufacturers, each one with its own software, having to find each

other and to work together. In addition, the swarm has requirements, such as: streaming, billing and quality of service.

On the other hand, there are many middlewares developed for specific proposes, as home area network, for example. We can mention there are HAVI, Jini and Salutation, for example. Jini (JINI, 2012) is a coordination language developed by Sun Microsystems, based on the Gelernter's work (1985). Jini was developed to be used with the Java programming language combined with RMI (*Remote Method Invocation*). It does not define an addressing system, requiring the use of another specification. HAVI (*Home Audio and Video Interoperability)* (HAVI, 2012; LEA et al., 2000) was developed to the Firewire interface (INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, 2008b). Salutation defines mechanisms for advertisements, discovery, description services request and security. Addressing and transport are not defined. Commercially, the most successful middleware for home networking, with devices from diverse manufacturers is the DLNA that is based on UPnP (Universal Plug-n-Play). AllJoyn and AVB are examples of emerging commercial middlewares, targeting to remote control in local area networks. DLNA, AllJoyn and AVB will be described in the next sessions.

Given the tendency of middleware segmentation by scope (e.g. sensor, multimedia, security, automotive), some works intends to leverage middlewares interoperability. In addition to the approach to develop specific bridges from one middleware to another, there are another approach – e.g. Oh et al. (2003), Tokunaga et al. (2002) and Moon et al. (2005) – that creates a meta-middleware to integrate many middlewares. These meta-middlewares have to create virtual devices that when receives a message, translates it to a neutral language, and then, to the available middleware interfaces.

There are some efforts to develop semantic middlewares to solve the inherent heterogeneity of standards, protocols and data formats. One of the first projects was Task Computing (MASUOKA et al., 2003), which used standard Semantic Web (RDF (KLYNE, CARROLL and MCBRIDE, 2014), OWL, DAML-S (COALITION et al., 2002)), Web Services (SOAP, WSDL) and pervasive computing (UPnP) technologies (MASUOKA, PARSIA and LABROU, 2003). This system aimed to allow users with no programming experience to use the devices around, with the assistance of the semantic environment, but it requires user interaction, not providing a seamless solution. Another effort built upon Task Computing was to construct a semantic middleware, which automates the creation of a semantic service description for the service associated to a device (SONG; CARDENAS;

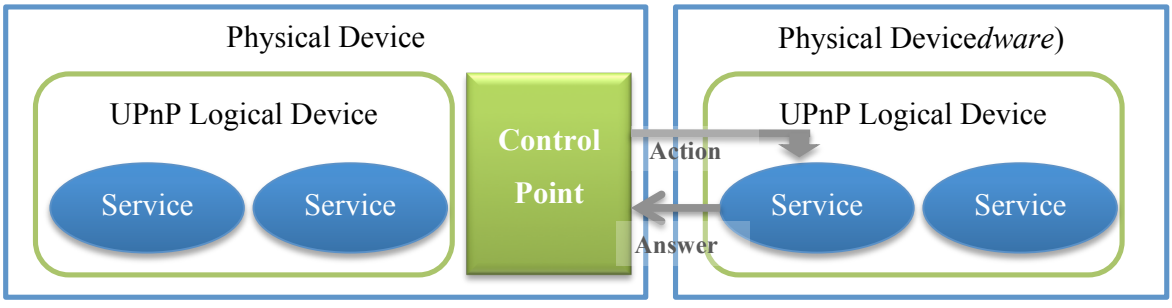
MASUOKA, 2010). To achieve this they implemented a mapping utility that translates each standard specification into a common ontology.

RUNES (Reconfigurable Ubiquitous Networked Embedded Systems), by Costa et al. (2005) and SOCRADES (Service-Oriented Cross-Layer Infrastructure for Distributed Smart Embedded Devices), by Cannata, Gerosa, and Taisch (2008), were attempts to build a generic middleware for embedded systems, despite they offer solutions for the automation of interconnectivity, this project lacks a solution for interoperability (automatic discovery and composition of services). UBIWARE project, by Katasonov et al. (2008), aims to solve the interoperability problem by using a decentralized infrastructure of software autonomous and proactive agents. Each agent represents a correspondent device and monitors its state and makes decisions on behalf of the device, agents can also discover and request other services if needed. The proposed middleware has a "repository of behaviors" that can be chosen by the agent according to the circumstances.

2.4.1 Universal Plug-n-Play - UPnP

UPNP Forum was formed in October 1999. UPnP specifications (UPNP, 2012) define a device as a logical entity that implements the UPnP protocols. Each UPnP device is related to a category that have a standardized services set. Each service is a feature that shall offer well-defined methods, parameters, states and values.

Figure 6 – Control Point, Logical and Physical Devices in UPnP



Source: Author

As described by Jeronimo and Weast (2003) and by Culler, Estrin and Srivastava (2004), UPnP defines Control Points that are entities in the network that use the services offered by UPnP devices and their notifications. UPnP Devices and UPnP Control Points are logical entities that are embedded in physical devices, as presented in Figure 6. If two UPnP

devices are embedded in the same hardware a hierarchy is defined, there is one UPnP Root Device and the others are UPnP Embedded Devices.

UPnP (*Universal Plug and Play*) is a decentralized system oriented to peer-to-peer connections, what allows devices to automatically and dynamically register to the home network, find the devices available in the network and to communicate in a distributed fashion. UPnP is independent of platform; it can be implemented in any Operating System or hardware (with enough processing power). UPnP does that using interoperable technologies, such as: IP, HTTP, XML and SOAP. UPnP was published in December 2008, as the ISO/IEC 29341 standard, composed by 73 parts.

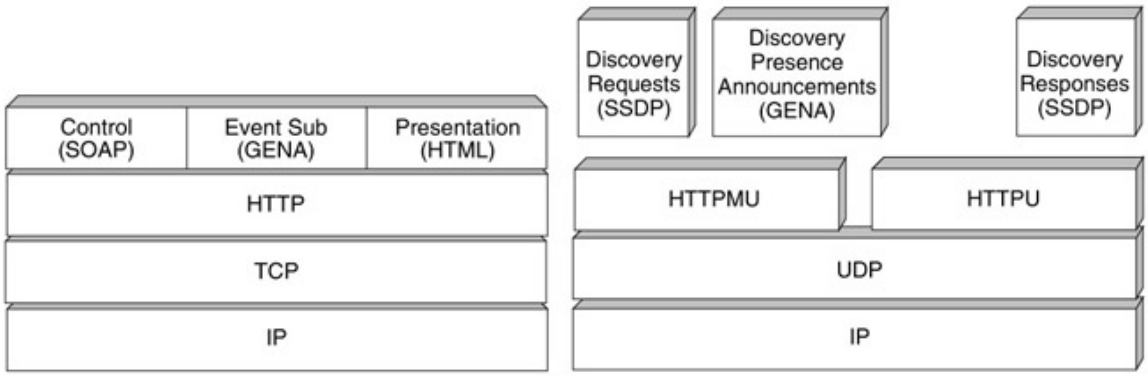
Six layers compose the UPnP architecture: addressing, discovery, description, control, events and presentation.

- **Addressing** uses IP, where each device acquires the address using DHCP (*Dynamic Host Configuration Protocol*). If DHCP is not available, auto-IP is used for the IP address definition.
- **Discovery** is the process in which devices are identified by the Control Points. UPnP uses SSDP (*Simple* Service *Discovery Protocol*) to do discovery. The discovery may be done in two ways: a Control Point is looking for a service or a Device advertises its services when connecting to the network. When a Control Point is looking for a service, it uses multicast queries (HTTPMU) to advertise that it is trying to locate a specific service. The devices that have the service available will answer using HTTPU (it is an extension of HTTP1.1 using UDP instead of TCP). Each service receives category identification and a unique ID number. Hence, UPnP requires the capability to send and receive HTTP messages with 100 bytes and to process XML files.
- UPnP **Description** is based on the use of a standardized XML over HTTP (*Hypertext Transfer Protocol*) with *unicast* (peer-to-peer) transmission. Each UPnP peer shall be able to receive and transmit HTTP and to handle descriptors of some kilobytes. The UPnP XML is the DIDL-Lite and is based on the DIDL (*Digital Item Declaration Language*), proposed in MPEG-21 (Brunett et al., 2006). MPEG-21 does not define the resources or metadata to be included in the DI; they are defined by the application (UPnP in this case).
- The **Control** Layer allows Control Points to send commands to devices. The UPnP control uses SOAP (Simple Object Access Protocol) that is based on

XML exchanges over HTTP. These messages sizes are around some kilobytes as well.

- The **event** signaling is based on a notification architecture using XML over HTTP: GENA (*General Event Notification Architecture*). This structure allows that Control Points register themselves to receive events from devices, allowing synchronous event signaling in the network. The peers shall be capable of process HTTP messages with about some kilobytes as well.
- Finally, the UPnP **presentation** makes an HTML administrative interface available, that allow users to monitor and control services and devices. In order to do this, the Control Point shall recover the HTML pages of each device in the network and load them to an Internet browser.

Figure 7 – UPnP protocols: peer-to-peer (left) and point-to-multipoint (right)



Source: JERONIMO and WEAST (2003)

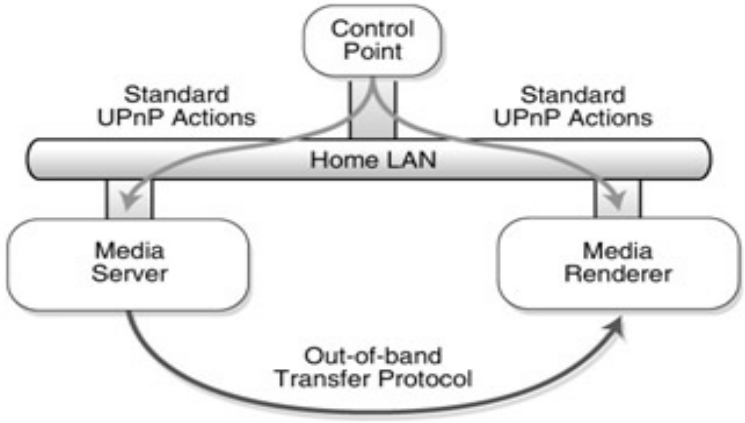
UPnP has a specific architecture to audio and video, the UPnP A/V. Its first version specifies two UPnP device types: Media Server and Media Renderer. The Media Server is responsible to store content and the Media Renderer is responsible to play the content. The end user interacts only with the Control Point that shall discover and configure the UPnP A/V devices in the local area network. The Control Point shall analyze the content format compatibility too. When Media Server and Media Renderer are set up, the content is transmitted directly, what can be done out-of-band.

The UPnP A/V version 1 includes:

- UPnP AV architecture;
- Media Server Device Template;
- Media Renderer Device Template;

- Reproduction Control Service Template (controls the brightness, volume and mute/unmute, for example);
- Connection Management Service Template;
- A/V Transport Service Template (controls the: play, stop, pause and seek);
- Content Directory Service Template (list the available content);
- Content Recording Control Service Template.

Figure 8 - UPnP A/V Architecture



Source: JERONIMO and WEAST (2003)

After UPnP definition, some pilot implementations were developed. Kim, Lee and Kwon (2002) present an implementation with client-server applications, domestic appliances emulators, a scheduler to energy management to control the devices to be on or off, and embedded devices with Windows CE and Linux OS.

Culler, Estrin and Srivastava (2004) shown that UPnP is applicable to high processing power devices with power plug, but to integrate wireless sensors and actuators, many computational constraints make the use of UPnP a challenge. In order to address this issue, Microsoft developed SCP (Simple Control Protocol).

2.4.2 Simple Control Protocol - SCP

SCP (*Simple Control Protocol*) was designed to be used in communications of low bitrate to the home automation market. Microsoft (2005) stands that SCP complement UPnP, allowing bridging UPnP networks to networks that do not support TCP/IP. As presented in Table 1, SCP divides the home network in two domains, automation and multimedia, inserting a gateway between them. These bridges would allow creating a unified logical network to the end users. The bridges shall discovery SCP devices, at the SCP network, to

retrieve their description and to generate a correspondent XML document compliant to UPnP. The bridge would allow a unidirectional discovery and communication between SCP and UPnP networks. UPnP devices would be able to discover and control SCP devices, but not the opposite.

SCP maps technologies directly with UPnP, simplifying protocols and replacing CML by binary codes. UPnP uses a similar device model, but it is possible to connect devices using a Control Point, as in UPnP, or using manufacturer defined connections.

Table 1 - SCP and UPnP comparison

Feature	SCP	UPnP
Addressing	An addressing node (ASA – *Address Space Arbiter*) defines the Network ID and Node ID. If ASA is not available, an arbitrary address is used.	IP - DHCP or *AutoIP*
Discovery	Looks for nodes without addressing.	Uses advertisements - SSDP to retrieve a URL.
Description	Binary codes (16 bits)	XML
Control	Control points or hard coded.	Control Point.
Events	Based in the properties: level or level change	Events based on services
Presentation	HTML	HTML
Security	3 levels: no security, cryptography with open key exchange or cryptography with protected key exchange.	DTCP

Source: Author

2.4.3 *Very Simple Control Protocol* - VSCP

Very Simple Control Protocol (VSCP), firstly proposed in 2000, is a really simple solution to monitor and control; designed to be used with low cost microprocessors. Ardakani, 2012 presents VSCP that is based in events, what allow intelligence distribution between the network devices. VSCP does not specify the lower network layers. It can be used with Ethernet, TCP/IP, Wireless, ZigBee, Bluetooth, CAN, GPRS, RS-232 and USB.

VSCP events are not addressed, they are broadcasted or multicast instead. Considering the VSCP controlling, a common interface to nodes control is defined and a specific class defines the units being used allowing nodes interoperability. The VSCP nodes implement decision matrixes that configure the actions initiated by a set of events.

VSCP has two node description systems: by registers and by XML files (MDF - *Module Description File*). As the MDF are too big to most of the VSCP devices the XML file may be stored inside the device or may be on the cloud, and just the URL is exchanged in the

VSCP network. VSCP uses unique identifiers to the nodes. The possibility of the nodes description transmission using a URL, instead of the XML file is really interesting in achieve a common system to heterogeneous networks.

2.4.4 *Digital Living Network Alliance - DLNA*

The *Digital Living Network Alliance,* founded in 2003*,* develops standards to home area networks. More than 200 consumer electronics companies joined DLNA with the goal to create interoperable devices, seamlessly to the end users, leveraging new services (DLNA, 2007). DLNA allows the media sharing, home automation and communications. There are more than 12.000 certified products.

The DLNA guidelines address all home networks connection levels, selecting existing standards, already well established, and defining how to integrate them. Table 2 summarizes the standards adopted by DLNA.

Table 2 – DLNA standards.

Layer	Role	Standards
Stream protection	How commercial content is protected in the home area network?	DTCP/IP WMDRM-ND *
Media Format	How multimedia content is coded?	MPEG2, MPEG4, AVC/H.264, LPCM, MP3, AAC LC, JPEG, XHTML-Print-
Transport	How multimedia content is transferred?	HTTP, Wi-Fi Multimedia (WMM) QoS
Media Management	How content is identified, managed and distributed?	UPnP AV 1.0, 2.0*, 3.0* UPnP Print Enhanced 1.0
Discovery and control	How devices discover, identify and control each other?	UPnP Device Architecture 1.0
IP Networking	How wired and wireless devices connect and communicate?	IPv4 Protocol Suite**
Connectivity		Wired: >Ethernet 802.3, MoCA (coaxial cable) Wireless: >Wi-Fi 802.11 n/b/g/a, Wi-Fi Protected Setup, Bluetooth*, WifiDirect*, HPNA*

* Optional

** IPv6 is foreseen for future use

Source: Author

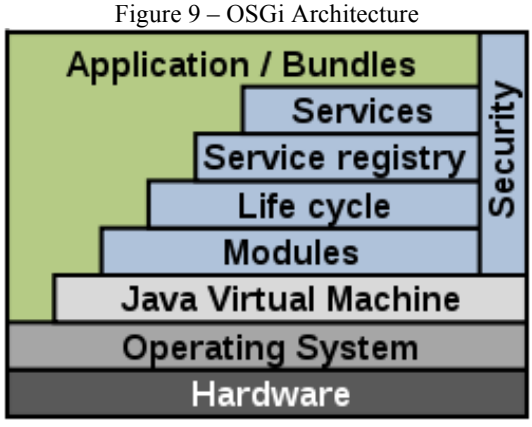
DLNA defines twelve device classes and a set of well-defined functions to each one. For example: a device from the Server class has to be capable of acquire, record, protect and to provide contents; on the other hand, a Player device shall be capable of playing contents from a Server after being configured by a Controller. DLNA builds interoperability with the connection of functional blocks. DLNA is a commercial standard, it consolidates the well-established contributions, limiting the possibilities of the systems that originated it, but looking forward to achieve devices interoperability.

The 2009 and 2011 DLNA guidelines require UPnP AV version 2 and 3 to support more advanced features: content synchronization, scheduled recording, electronic programming guide and digital rights interoperability.

2.4.5 Open Services Gateway Initiative - OSGi

OSGi (Open Services Gateway Initiative) (THE OSGI ALLIANCE, 2012), founded in 1998, is a services platform that makes available a common open architecture to deploy and manage services coordinated. OSGi framework was developed in Java. The OSGi specifications, had it first release in 2000 and its last release in 2014, have a wide range of target platforms: TV set-top boxes, gateways, modems, consumer electronics, PCs, cars, cell phones and industrial computers, for example.

Figure 9 – OSGi Architecture



Source: THE OSGI ALLIANCE (2012)

OSGi initial focus, was in home networks, but the specifications were used in other markets, but they have never been used in home networking. Still, OSGi developed new guidelines to home networking, focusing in fixed devices (e.g. kitchen appliances, home gateways, sensors and actuators for home automation). The guidelines include requirements

and functional specifications, APIs, a model for local and remote gateway management. Figure 9 presents OSGi architecture.

OSGi had contributions to home networks but uses two concepts not aligned with this thesis: Java language and centralized control (with a gateway).

2.4.6 JXTA Protocols

JXTA (SUN MICROSYSTEMS, 2007) is a protocol specification, released in 2002, based on XML that defines how nodes in a peer-to-peer network can organize themselves in groups, advertise and discover resources, communicate and monitors each other. It is an open specification, language independent and suitable to any network technology. JXTA is a micro-core-architecture, i.e. it is organized with a minimum core and services implemented as additional modules. JXTA was designed to solve the communication between cellphones, PDAs/Tablets, personal computer and servers.

The specifications, defines a JXTA virtual network as peers that associate with each other becoming groups. These groups are used to define a set of resources and services, to provide a region with access control, to define scope and to monitor members. Peers categories are:

- **Edge peer**: a simple peer that may be a device with transient, low bandwidth network connectivity.
- **Minimum peers**: peers with constrained resources that do not implement the full JXTA specification.
- **Proxy peers**: act as proxy to minimum nodes or to nodes behind firewalls;
- **Rendezvous peers**: these peers cache information about the peers available in the network, working as a registry. They facilitate discovery and solve names, mapping peers to IP addresses.
- **Relay peers**: they retrieve routing information and forward messages to peers behind firewalls, NATs or routers. Usually the same devices play the *rendezvous* and *relay* roles.

All JXTA entities are made available to the network using advertisements, which are XML documents that designate URNs and information of each entity. Advertisements have pre-defined lifetime, because they can be stored in the devices local cache. There are six advertisement types: peer, group, connection, service, content and connection point.

Peers send messages using connections. Connections have unique identifiers and are related to two connection points: one input and one output. Connections may be unidirectional not-reliable virtual channels (*pipes*) or bidirectional reliable virtual channels (*Channel*). JXTA allows to set up a virtual network over a physical network, allowing peers to interact regardless their location and physical connections.

Messages are XML documents that have the routing defined in the header, informing the peer sequence to be used for routing. There are seven basic services provided by JXTA that should be provided by a JXTA group:

- **Peer Resolver Protocol**: sends a generic query to peers (unicast or multicast). It supports the other protocols.
- **Peer Discovery Protocol**: dynamic discover JXTA resources. It sends a multicast message through the local network and to the rendezvous peers networks. This protocol is implemented by the discovery service. Some peers available in the network may not answer one consult message to a resource, because the protocol is not reliable. *Rendezvous* peers are used to store advertises of resources that they already know.
- **Peer Information Protocol**: collects information about a peer state. It makes polling to verify specific peers connectivity.
- ***Rendezvous* Protocol**: propagates messages inside a group. It is the base to the *Peer Information Protocol* and *Pipe Binding Protocol* protocols.
- **Pipe Binding Protocol**: associates a connection to two peers.
- **Endpoint Routing Protocol**: specifies a set of messages to define routing, previously to the actual messages transmission. The routes are stored locally, including the sender and receiver identification, the message lifetime and the ordered sequence to the peers ID.
- **Association protocol**: used to validate peers to register in groups. It uses a unique password to authenticate peers.

Protocols are independent to each other and it is not necessary to have all protocols implemented in each peer. For example: if a peer uses pre-defined peers to route its messages, it does not need to fully implement *Endpoint Routing Protocol*. Or, if a peer does not use other peer information, it shall not implement *Peer Information Protocol*. The mandatory protocols in every peer are: *Peer Resolver Protocol* and *Endpoint Routing Protocol*.

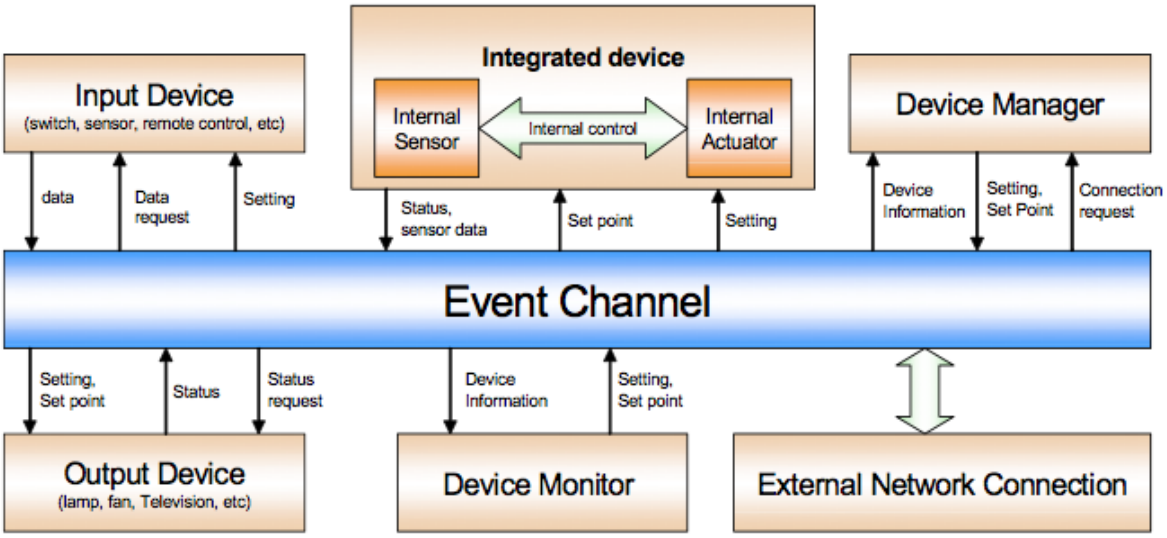
Services specification and use are not defined in JXTA, the use of WSDL, UPnP and SOAP are suggested.

JXTA brought many contributions to the home networks. It is independent of specification, language and network and it uses a virtual network model. Despite adopting XML, the micro kernel architecture may expand the system scope, allowing the integration of lower processing power devices. JXTA optimizes peer-to-peer communication allowing dynamic routing with multiple hops.

2.4.7 TAO Middleware

TAO (The ACE ORB) is an open-source CORBA v3.0 implementation specific to high bandwidth communication with support to Quality of Service (TAO WEBSITE, 2012). TAO is based on the ACE framework (*Adaptive Communication Environment*), which is a C++ framework used to simplify network programming and code portability (HUSTON, JOHNSON, SYYID; 2003).

Figure 10 – TAO Middleware Example



Source: BINUGROHO, SEO and CHOI (2008)

Binugroho, Seo and Choi (2008) developed an implementation using TAO to deploy home network. As CORBA is a solution to distributed objects computing, one object represents each appliance. The network architecture includes a server that is connected to the

home network and with the Internet, acting as a gateway. In addition, this server manages the devices and distributes content among the client devices.

The main architectural element is the Event channel, as presented in Figure 10. It manages objects and events between providers and consumers. Providers generate events to the channel and consumers receive events. Events may be received asynchronously (*push)* or synchronously (*pull), it waits for the read command from the consumer.* TAO's event channel delivers events to all consumers. Binugroho, Seo and Choi (2008) developed a routing decision process that takes load off the Events Channel, directing events to the related consumers only.

2.4.8 Extensible Messaging and Presence Protocol - XMPP

The Extensible Messaging and Presence Protocol, XMPP (SAINT-ANDRE, 2011), is a set of open technologies for instant messaging, presence, multi-party chat, voice and video calls, collaboration, lightweight middleware, content syndication, and generalized routing of XML data. XMPP was originally developed in the Jabber open-source community, back in 1999, to use in instant messaging services and continue currently active, with a broader scope.

XMPP provides a technology for the asynchronous, end-to-end exchange of structured data by means of direct, persistent XML streams among a distributed network of globally addressable, presence-aware clients and servers. XMPP architecture is similar to the email architecture, with modifications to facilitate communication in close to real time.

XMPP introduces the "Availability for Concurrent Transactions" (ACT) architectural style that is based on:

- **Global Addresses**: as with email, XMPP uses globally unique addresses (based on the Domain Name System) in order to route and deliver messages over the network. All XMPP entities are addressable on the network, most particularly clients and servers but also various additional services that can be accessed by clients and servers. XMPP addresses are called Jabber IDs or JIDs.
- **Presence**: XMPP includes the ability for an entity to advertise its network availability or "presence" to other entities.
- **XML streams over long-lived TCP connections**: enable each party to push data to the other party at any time for immediate routing or delivery.

- **Structured Data**: the basic protocol data unit in XMPP is a fragment of XML that is sent over a stream, called XML "stanza". The root element of a stanza includes routing attributes (such as "from" and "to" addresses), and the child elements of the stanza contain a payload for delivery to the intended recipient.
- **Distributed Network of Clients and Servers**: end-to-end communication in XMPP is logically peer-to-peer but physically client-to-server-to-server-to-client. Each client can communicate exclusively with the server that it is registered and the communication between any two given deployed servers is strictly discretionary and a matter of local service policy.

In XMPP clients establishes an XML stream with a server by authenticating using the credentials of a registered account (via SASL negotiation) and that then completes resource binding in order to enable delivery of XML stanzas between the server and the client over the negotiated stream. The client then uses XMPP to communicate with its server, other clients, and any other entities on the network, where the server is responsible for delivering stanzas to other connected clients at the same server or routing them to remote servers. An XMPP server may store data that is used by clients; in this case, the relevant XML stanza is handled directly by the server itself on behalf of the client and is not routed to a remote server or delivered to a connected client.

XMPP host many add-on services that provide additional functionality; examples include publish-subscribe services and the Sensor Andrew extensions (Rowe et al., 2011). Sensor Andrew is a large-scale effort to widely deploy sensing devices across Carnegie Mellon University with a broad set of applications ranging from infrastructure monitoring, first-responder support, quality of life for the disabled, water distribution monitoring, building power monitoring and control, social networking and biometric systems for campus security.

2.4.9 Audio and Video Bridging - AVB

An “Audio Video Bridging” (AVB) network implements a set of protocols being developed by the IEEE 802.1 Audio/Video Bridging Task Group, this effort started on 2005. AVB implements extensions to standard layer-2 MACs and bridges, what make AVB devices to be able to reserve a portion of network resources through the use of admission control and traffic shaping, to send and receive the new timing-based frames and to coordinate the network**.**

AVB motivation was to develop a system focusing in Time-sensitive applications. That means that the data must be delivered within a certain window (typically before a specified maximum delay) and the connected devices need to have a common sense of wall clock time for synchronization, coordination and phase locking. Professional audio/video applications have a 2ms maximum transport delay and 10μs maximum synchronization error (1μs is desirable). Control and sensor networks may have even more stringent requirements, while home networks are typically more relaxed. (TEENER et al., 2013)

IEEE 802.1BA (INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, 2011a), Audio Video Bridging Systems standard, specifies the default configuration for AVB devices in a network. Since the whole AVB scheme depends on the participation of all devices between the talker and listener, any network element that does not support AVB (including so-called “unmanaged bridges”) must be identified and flagged.

The protocol used for maintaining timing synchronization is specified in IEEE 802.1AS, (INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, 2011b), also known as gPTP, which is a subset of IEEE 1588 (INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, 2008a). Accordingly to this protocol, there is a single device called the grandmaster that provides a master timing signal, to which all other devices synchronize their clocks. The source of gPTP time need not be the source of all or even any media streams, since the notion of a Presentation Time abstracts Network Time from Media Time. gPTP defines hardware time stamping mechanisms for each LAN technology that propagates the time with very little degradation from hop to hop. gPTP creates a clock tree from the Grand Master through all paths of the LAN that support gPTP. Many different LAN transports (e.g. Wi-Fi, MoCA, and G.hn) in combination and still maintain accurate time. Significant changes to the original IEEE 1588 definition, among others, are a limitation of the scope to support only Layer 2 Transport for Ethernet, and to limit the devices defined to ordinary clocks called ‘Time Aware End Stations’ and specifically adapted boundary clocks called ‘Time Aware Bridges’.

A multimedia application can implement synchronized distributed rendering using 802.1AS and higher layers. Specific audio samples and/or video frames carried by higher-layer protocols are given an associated presentation time (in terms of the shared 802.1AS clock) by the media source that is also an AVB talker. Each media renderer, that is also an AVB listener, renders the referenced audio sample or video frame at the 802.1AS presentation time.

The IEEE 802.1Qav (INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, 2010a) describes a credit-based scheduling algorithm, which is used in AVB switches instead of the usual strict priority-handling schemes. The use of this scheduler permits bridges to additionally reduce latency for specific data streams, which have high real-time requirements.

IEEE 802.1Qat (INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, 2010b) describes the Stream Reservation Protocol (SRP). This protocol allows network stations to establish, control and tear down data streams with defined Quality of Service (QoS) requirements. End stations are divided into ‘talkers’, which transmit stream frames to the network and act as a communication source and ‘listeners’, which receive the stream frames and therefore act as communication sinks. A device can implement talker and listener functionality at the same time. Talkers advertise streams by sending Talker messages, and listeners subscribe to streams by sending Listener messages. Talker messages are flooded over the ports on which SRP is enabled, while Listener messages are forwarded only back to the source of the Talker. Talker messages contains information about the bandwidth required for the stream, as Bridges forward the Talker Advertise messages across the network, they evaluate whether a reservation can be successfully made (e.g. verifying if it is enough bandwidth). As each node forwards the Talker message, it updates the Accumulated Latency field in the message with the worst-case latency for the given hop. Reservations are made as the Listener messages are propagated back toward the Talker – before propagating these messages to the Talker, Bridges make a reservation on that port by updating the bandwidth on the shaper for the queue associated with the traffic class, updating available bandwidth for the given port, and adding the port to allow it to flow.

IEEE 1722 (INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, 2011d) and IEEE 1722.1 (INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, 2013) define higher layer communication protocols. IEEE 1722 defines the Audio Video Transport Protocol (AVTP). The AVTP defines a data formatting that make use of the IEC 61883 (INTERNATIONAL ELETROTECHNICAL COMISSION, 1998) audio and video formats (that have been used for years on IEEE 1394 – Firewire (INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, 2008b)). In order to maintain real time performance it is critical that the source and sink of audio/video data maintain synchronized media clocks. AVTP allows each stream to maintain a separate media clock. This means that a single AVB network can accommodate multiple clock rates. AVTP uses the wall clock defined by 802.1AS to create cross timestamps with designated media clock edges. By

transporting these cross timestamps along with the associated samples it is possible to precisely recreate the original media clock with the correct sample and clock alignment. On the other hand, presentation time is key to normalize the network latency and to maintain sample coherence along multiple network paths. Presentation time is expressed as an offset that is added to the AVTP cross timestamps. IEEE 1722.1 presents a way to transport media streams over a LAN. It defines four steps to connect end stations: audio/video discovery, enumeration, connection management and control (AVDECC). The application that will use these individual steps is called an AVDECC Controller and is the third actor in the AVDECC Talker, AVDECC Listener and AVDECC Controller device relationship. An AVDECC Controller may exist within an AVDECC Talker, an AVDECC Listener, or exist remotely within the network in a separate end station. The AVDECC Controller can use the individual steps to find, connect and control entities on the network but it may choose to not use all of the steps if the AVDECC Controller already knows some of the information (e.g. hard coded values assigned by user/hardware switch or values from previous session establishment) that can be gained in using the steps. The only required step is connection management because this is the step that establishes the bandwidth usage and reservations across the audio video bridging (AVB) domain.

The steps are broken down in discovery, enumeration, connection management and control. Discovery is the process of finding AVDECC Entities on the LAN that has services that are useful to the other AVDECC Entities on the network. Enumeration is the process of collecting information from the AVDECC Entity that could help an AVDECC Controller to use its capabilities. The AVDECC Entity Model is used to describe the internal structure of an AVB device. An AVB audio/video device is comprised of network streaming port, other external ports or jacks, and internal ports. In order to intelligently manage an audio/video system a controller needs to be aware of and in control of all these paths. Simply routing audio from a networked media player to an amplifier doesn’t solve the problem if the controller cannot then create the connection from the amplifier to the speaker. The AVDECC Entity model allows end-to-end routing of audio/video signals (TEENER et al., 2013). Connection management is the process of connecting or disconnecting one or more streams between two or more AVDECC Entities. Finally, control is the process of adjusting a parameter on the AVDECC Entity from an AVDECC Controller.

IEEE 1722.1 messages are organized as binary tables, similar to MPEG standards (ANGELIDES; AGIUS, 2011). It is based on the AVDECC Discovery Protocol Data Unit (ADPDU) that follows the IEEE Std. 1722-2011 control AVTPDU header.

Finally, IEEE 1733 (INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, 2011c) correlates the IETF Real-time Transport Protocol (RTP) timestamp with the 802.1 AS presentation time, allowing the renderers to start playing at the same time and keep playing at the same rate.

2.4.10 The AllJoyn software framework

AllJoyn (ALLSEEN ALLIANCE, 2014) is an open source project, which provides a software framework to create dynamic proximal networks with connected products and software applications of various manufacturers. AllJoyn was initially developed by Qualcomm Innovation Center, Inc. in 2014, and is hosted by the AllSeen Alliance. The AllJoyn software framework and core system services let compatible devices and applications find each other. AllJoyn vision is that devices can interact regardless of how they are connected, but currently, the communication layer is limited to Wi-Fi.

AllJoyn core building blocks and services are: discovery, connectivity, security and management of ad-hoc proximal networks among nearby devices.

The AllJoyn Core Framework includes a set of service frameworks. The initial set of service frameworks include:

- **Device Information and Configuration**: allows the device to broadcast information such as device type, manufacturer and serial numbers; also allows the user to assign a name and password to the device;
- **Onboarding**: allows devices to borrow others interface in the network (e.g. an air conditioner can be controlled by a smart TV);
- **Notifications**: products can broadcast and receive basic communications (text, image/video, audio);
- **Control Panel**: enables a device like a smartphone or tablet to control another product via a graphical interface;
- **Audio**: so any device with audio can stream it to any set of Alljoyn-enabled speakers, audio receivers and other audio playback devices.

2.5 STATE OF THE ART DISCUSSION

Ubiquitous computing is a hot topic for many years and intense research is being developed in this topic, especially when segmented in specific fields of applications. The transformation power of the swarm computing is in defining a common platform that can blur the field applications and maximize its transformation power, taking advantage of the synergies among these fields. As we observed, many solutions and standards are being developed, but they fail to achieve this goal and to be established as a *de facto* solution.

In order to achieve a cross-field architecture, it is necessary to have flexibility regarding standards, due to the constant systems evolution when considering a wider scope of the system architecture. Therefore, it is fundamental to consider an easy to develop, gateway-friendly architecture, with physical communication abstractions as well as intelligent services that allows extrapolating standards expanding their original design, receiving new ontologies.

In the state of the art review, some solutions proposed to heterogeneous networks as well as the theoretical foundations of the Swarm were presented. In order to achieve collaboration between different devices in the Computing Swarm, the Service Oriented Architecture and Web Services are pertinent. They allow connected devices to use features from each other, independently of the implementation, standardizing the interfaces. RESTFUL simplifies the implementation, making SOA easier to develop. The entities using and offering services, as well as the entities that will manage Swarm Computing resources may be faced as agents, putting together a multi-agent system, as described in the state of the art review. Semantic computing is a powerful technology that we propose in this thesis to be used in order to achieve interoperability and flexibility when using evolving, live standards.

Regarding middleware solutions, we have presented some of the current systems related to the work theme. We could identify distributed processing middleware that have been used in ubiquitous computing, and we have explored middlewares developed for local networks, as a domestic environment. In this field we analyzed AllJoyn and DLNA (with UPnP and SCP or VSCP to make it lightweight). These are interesting solutions, DLNA is disseminated in the market, but suffers with interoperability issues and it has a really strict scope of application. AllJoyn has a broader scope and presents a really simple and useful programming framework, but has a local network focus and requires the developer to know before hand what the devices that will communicate are and how, not presenting a dynamic and opportunistic approach.

OSGi is a middleware solution that has good infrastructure and a consistent framework. Its main problems are the requirement to use Java language and its complexity to use low power nodes. JXTA is a multi-purpose middleware based on XML messages. JXTA architecture is organized in layers, allowing the existence of simpler nodes co-existing with more complexes ones. On the other hand, JXTA has a complex specification and is not complete, requiring work in order to develop application standards, the same way as UPnP required DLNA in order to evolve.

AVB is a solution originally developed for audio and video applications, but wich is being expanded to other fields, such as industrial applications. It presents an interesting entity model, and defines quality of service tools for communications and timing synchronization. On the other hand, AVB uses a description method based on tables, wich is a rigid structure highly dependent on standards and that makes it difficult to implement semantic reasoning.

XMPP is currently being used as a lightweight multi-purpose middleware. Implementations for sensors and Internet of Things are emerging. XMPP is based on streaming XML and has many useful extensions. The main downside with the XMPP is its centralized architecture, based on the e-mail architecture. The discovery system is really interesting and can be used as reference to the Swarm application.

The semantic middlewares analysis offered great insights in order to achieve a real world implementation of a semantic-aided Swarm; however, many of the project homepages are not available in current days, including all associated resources such as implementation software, examples, tutorial, API, etc.

As we could verify, all the analyzed systems have strong and weak points; and there is still space for a proposition for the Swarm Computing Control Plane, with a broader scope of communication, flexibility to standards and capable of working in heterogeneous nodes.

3 THE SWARM OS CONTROL PLANE

This chapter presents the Swarm OS Control-Plane proposal. It is organized in seven sections. The first section contains an overview of the Swarm Control Plane and its taxonomy. The second section presents the scenarios for the Swarm application in order to provide a context to the work. The third section presents the layers of the Swarm Control-Plane model, presenting its grounding definitions. The fourth section presents the Control-Plane architecture proposal, discussing the platform services. The fifth discusses the Control-Plane Broker. The sixth section presents the sequence diagrams for the control plane services usage. Finally, in seventh section the Control-Plane interfaces proposals are presented.

3.1 CONCEPTUALIZATION AND TERMINOLOGY

To ensure a seamless and consistent execution of a broad range of applications on the emerging and evolving Swarm platform, an essential item is the Swarm Operating System, i.e. Swarm OS. The framework shall offer open APIs, protocols and templates to ease the development and deployment of these applications ensuring interoperability, but also provides the necessary execution mechanisms. The Swarm OS is a distributed framework combining two elemental components: data plane and control plane. The **Data Plane** manages the run-time execution of the various services composing the Swarm, so that all performance requirements are met. At its core, it is a distributed storage system (the Global Data Plane) that is responsible for passing data efficiently and securely between sources and sinks, producing a variety of data organizations with varying levels of data persistence (from transient to deep archival) and security. The **Control Plane**, on the other hand, is essential in recruiting the necessary resources so applications can perform their tasks within the agreed quality and assurance levels. To do so, it needs to be aware of the resources available, negotiate the usage of those resources, and ensure they are provided. This work focuses on the Swarm OS Control Plane that offers protocols definition, open APIs and templates to allow applications and services development and devices interoperability. The control plane is the part of the system that is responsible for managing Swarm resources – defining the Swarm ecosystem actors, how to describe and use services and resources, how advertise and discover them, how to do transactions, how to adapt content and how multi-agents cooperate.

In order to accomplish the Swarm vision, where autonomous and heterogeneous devices cooperate through resource sharing to accomplish a set of goals, a service-oriented

architecture was adopted. In this scenario, we are introducing the concept of Swarmlets. A swarmlet is composed as a graph of interacting services, i.e. Swarmlets are applications in the Swarm.

As the Swarm is an open system with a variety of services being provided and consumed, the establishment of Service-Level Agreements (SLA) (contracts) between the service provider and the consumer is essential to the system. These transactions define an agreed level of Quality of Service or even a credits system, such as micropayments or reputation. The contract ensures to the consumer an agreed fraction of the resources to be available to the service. Contracts typically are scoped (that is, they are valid for a certain time and space).

The contract is negotiated between two agents, the service provider and the service consumer. The Control Plane considers yet a broker; a third party that may provide assistance in the negotiation and other functions between service providers and consumers.

The Swarm platform leverages a scenario where scarce resources are dynamically traded between competing interests, where diverse agents – that compete for resources – have to actively exchange information to perform joint optimization being aware of the changing environment and workload conditions**.** The two common paradigms in decision-making are localized (distributed) or centralized. The centralized decision making approach can achieve better system optimization, due to the overall vision. The problem is that it is impracticable having information about all Swarm ecosystem. On the other hand, to use localized decision making, in which each participant follows a local strategy, to arrive at some globally acceptable result, the individual policies should be harmonized in some degree, as discussed in Parsa et al. (2010). The Swarm OS has a hybrid approach, taking advantage of unstructured hierarchies of agents. This hybrid approach relies on the agents' independence and potential of cooperation. The possibility to represent entities with agents and Personas, allow the existence of a stakeholder to deal with system optimizations under its scope. For example, an agent could represent an ambient and deal with the optimizations in this space.

Table 3 – Swarm Control Plane Terminology Summary

Resource: Named Entity - something that has an ID in the Swarm ecosystem;

Service: A functionality that is accessible via an external API with identifiable Quality of Service (QoS) commitments;

Agent: An autonomous entity that directs its activity towards achieving goals based on observations and actions in the environment. For example:

- **Service Provider:** Agent that offers capabilities through (a) service(s). [1]
- **Service Consumer/Requester**: Agent that has service needs.
- **Owner:** A persona that owns a resource. The owner sets the policies on how that resource can be utilized. The Owner Agent will act as the service provider for that resource;
- **Persona:** A virtual representation of a player in the swarm world. Players can be people, organizations, or groups thereof. It consists of identification (represented by biomarkers, keys or others), configuration, preferences and/or state. Persona can be constructed hierarchically – for instance, a single person, may have different sub-personas (dependent upon the context).

Discovery: A mechanism for agents to either request for or advertise services.

Contract: An agreement about service provision including commitments pertaining to time and location, QoS, and load provision.

Brokerage: The process of setting up contracts.

Broker: provides assistance (as a set of services) to service providers and consumers in order to seal contracts. The broker is the intermediary to the Control Plane services providing, that includes services advertisement and registration, services provider candidates identification and selection, contract definition and registry, monitoring and even policing.

Cell: An abstraction consisting of a pool of contracts, i.e. a pool of temporarily committed resources, made available to a service consumer.

Source: Author.

The term agent in computing covers a wide range of behavior and functionality. In general, an agent is an active computational entity that has a persistent identity that perceives, reasons about, and initiates activities in its environment, as well as collaborates with other

agents. The Agent persistent identity will be called **Persona** in this work, and aggregates all information that builds its identity: historical data, preferences, and configurations. The **Agent** (procedural) executes the Persona (declarative) that represents an entity. Each agent can be applied to represent different kinds of entities, for example: a person, a building, a room, a device, a service, etc.

Figure 11 - Agents' hierarchy example

Office Management Service

- Room 1 Management Service
  - Room 1 local air conditioning equipment management
  - Room 1 lightening
  - Windows shutter control
- Room 2 Management Service
  - Room 2 local air conditioning equipment management
  - Room 2 lightening
  - Room2 display service
  - Room 2 AVB network service

Source: Author

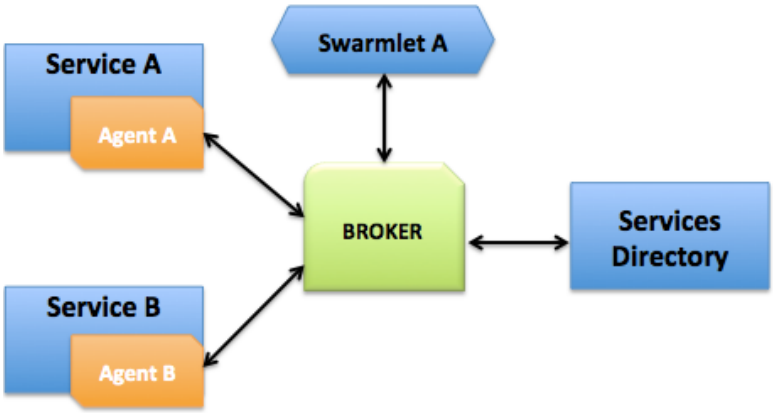
In order to leverage a system with local reliability and local policies enforcements therefore maintaining a global system overview with global optimization, we propose the use of the resources-ownership concept. The owner of a resource is an agent that will be responsible for this service, being the one that defines its usage polices and that is responsible for its negotiation. These agents can be organized as a hierarchy, allowing the higher levels to be responsible for the global optimization and for the lower-level resources-disputes solution. The lower-level agents are responsible for reliably provide the service. These agents could be related to virtual concepts, providing higher semantic level services.

As well as a huge amount of resources/services will be available in the Swarm; many potential requesters will be available to use these services as well. This way, resource sharing is expected, i.e. one service provider will be requested by multiple consumers. Therefore, it will be possible, and it will be probable, to occur conflicts between commands, as presented by Wilson, Magill and Kolberg (2005). We could use as example a window that receives the "close the window" command from the security system, but receives the command to "open

the window" from the HVAC system, in order to efficiently control the temperature. In order to solve conflicts, it could let the decision to the service provider or the service provider could inquiry to a third party, the **resource owner,** for counseling. Anyway, the conflict resolution will have to follow a policy: it could be as simple as always execute the last command, or to consider the context (if there is someone home at that time, for example), or even to consider a request originator hierarchy – in terms of user (e.g. a device owner has more priority than a guest, or relying on a credit systems) or in terms of systems priority (e.g. security is more important than resources savings).

As an example, we could consider an office with two rooms, each one with many services, including individual air conditioning systems. As each room has an intelligent control system that allows ambient automation and control, each one has an agent that owns all the resources/services in each room. There is also an agent responsible for the office, which owns both room agents, as well as the services providers that are part of each room. This way, this higher-level agent in the hierarchy, that represents the office, has the responsibility for global optimization and conflicts solving. If a command was sent to increase the temperature in the room 1 and another command was sent to decrease the temperature in room 2, this higher-level agent should identify the conflict and define what to do.

Figure 12 – Brokerage and agents cooperation example



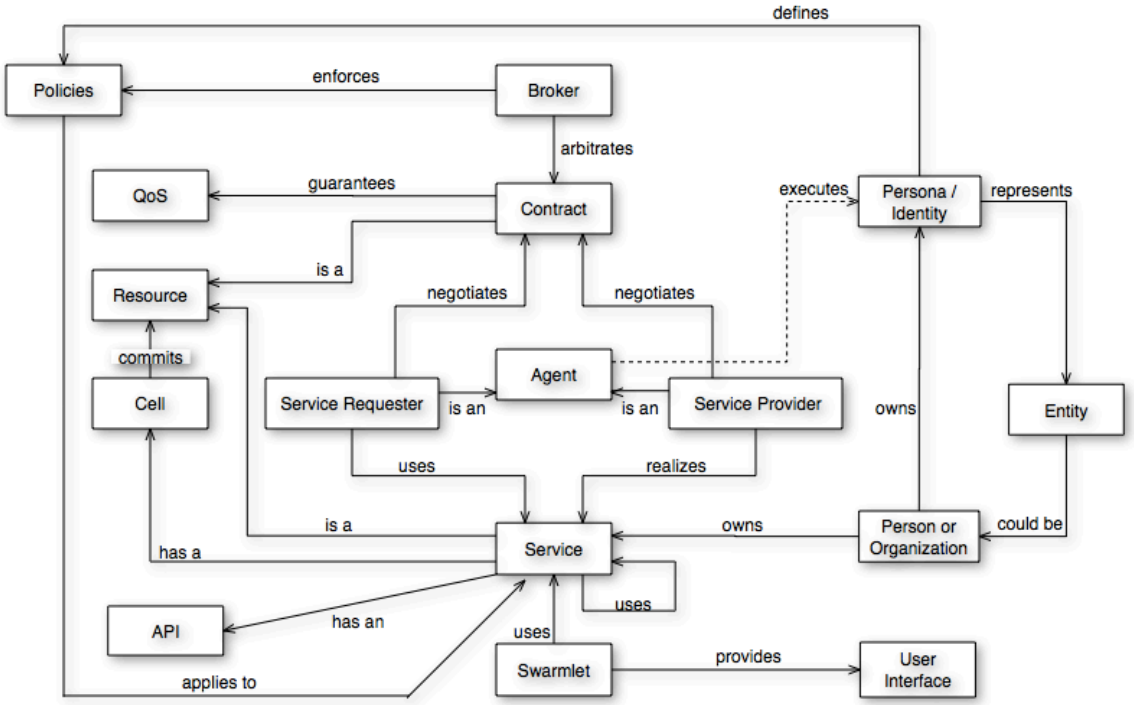
Source: Author

Personal agents are agents that represent entities. They have the important duty of retrieve entities information and preferences, learning with the entity that is represented. It may carry a represented person identification alternatives and the parameters to use it (e.g. public key certificate, face recognition parameters and fingerprint). A personal agent learns priorities from behavior, e.g. learning preferences such as comfort *versus* resources efficient consumption. Sharing this information with a room management agent, then it is possible to

configure adequate levels of heating and lightening, optimized to follow the policies of the room.

An important actor in the Swarm OS is the **Broker** that provides assistance to services providers and consumers to seal contracts. The Broker is the intermediary to access the Swarm OS framework, it includes in its responsibilities: services advertisement and registration, services provider candidates identification and selection, contract definition and registry, monitoring and even policing.

Figure 13 - The SWARM Control Plane Objects Relationship



Source: Author

Putting the Swarm vision in a nutshell, it is a system with autonomous and heterogeneous devices cooperating through their own resources sharing. Each cooperation relationship is sealed with a contract that establishes a service providing commitment and enforces an agreed quality of service. The contract represents the available resource itself; i.e. it is a share, or a cell, of the device's overall resource. The Swarm is organized in a service-oriented and multi-agents architecture. The agents compose this system as intelligent software components that interoperate providing and consuming services. The service is a resource; it provides access to a capability. A service itself can use other services. Swarmlets will run over this framework, leveraging machine-to-machine cooperation, allowing the distributed composition of a graph of services. Personal agents will have a key role in the system. Each

personal agent represents a person and carries its Persona, composed by policies, preferences and information of an individual. Agents and Personas can represent entities.

3.2 SCENARIOS FOR THE SWARM APPLICATION

When the World Wide Web was first launched, few people would have predicted the vast range of applications that it would enable. It has profoundly changed the way people interact and behave, how businesses are run and how information is exchanged. We believe that swarm-based systems will deploy the same way; it is essential to provide a collaborative platform to address its challenges and opportunities. By viewing key challenges through many different eyes, more innovative solutions can be generated (LEE et al., 2012).

The SWARM vision achievement depends on the availability of an open computing platform on which the swarm applications can be built and composed concurrently by millions of creative inventors. Open architectures with dynamically available resources can open up significant security and privacy risks, on the other hand, they can also make systems more powerful (efficient, through sharing of resources), more resilient (through dynamic reconfiguration leveraging redundant resources), and more capable, enabling applications that have not been realized yet. Some scenarios will be presented in this session, as speculation of its applicability.

3.2.1 Electronic Mirror Scenario

The Electronic Mirror Scenario consists of an electronic device that works as a mirror, reflecting what is in front of it. In addition, the electronic mirror is capable of overlaying information over the reflection.

The electronic mirror device offers the following services: Reflection Service, Video Streaming Service and Over-the-top Display Service. The Reflection Service reflects what is in front of the mirror. It uses as input the video recording service and the effect is to present the reflected image on the display. The video streaming service generates (output) a video stream capturing images in front of it, offering setup parameters as the video stream destination and streaming parameters. The Over-the-top Display Service allows image/video displaying over the reflection, as in a regular display. It may generate audio as well.

A swarmlet for personal assistance will look forward managing the display and using other services in order to add value to the mirror. It will look for personal agents to retrieve

preferences that will help the swarmlet to select the information to be presented / services to be accessed. The personal agent will provide proximity parameters (such as face recognition data), link to relevant services (such as agenda and e-mail) and preferences (configuration about the preferred look and feel).

The product may have a remote service address pre-registered, as the *augmented hair service,* that can change the hair color and haircut of a given person in a picture or in a video.

3.2.1.1 Use case description for the electronic mirror scenario

Pre-condition:

- Two or more Swarm OS enabled devices connected to a common network:
    - Electronic mirror device
        - Composed by 1 display, 4 embedded cameras, processing and communication capability;
        - Running:
            - Electronic mirror swarmlet.
            - Reflection service: reflects what is in front of the mirror with capability to display images and videos. It uses as input the video recording service and the effect is to present the reflected image on the display.
            - Video streaming service: generates a video stream capturing images in front of it. It offers as parameters to be configured: the output destination and streaming parameters. The output is the acquired video stream.
            - Over-the-top Display Service: allows image/video displaying over the reflection, as in a regular display. It may generate audio as well.
    - Personal device (cell phone)
        - Running the personal agent.
    - Services in the cloud
        - Augmented hair service: receives a picture or a video and can virtually change your hairstyle, considering color and haircut.

- Agenda service: presents the next events scheduled to the person, allows editing.
- E-mails service: presents the unread mails, allows interaction.
- News service: presents the last news in selected topics considering user preferences and habits.
- Face recognition service: identifies a person using a video stream as input, considering a pre-loaded database with face parameters.

Expected behavior:

- Simulate the person reflection, and shows it on the display as a mirror;
- Presents an overlaid interaction layer with a graphical interface that allows user to:
  - Simulate to change the user's hair color (with augmented reality);
  - See personal agenda;
  - Read urgent e-mails;
  - Read news headlines.

Actions:

- Swarmlet looks for personal agents (using the Broker);
- Swarmlet identifies the personal agents in the local network and retrieves the face recognition parameters;
- Swarmlet identifies and establishes contracts (through the Broker) to use:
  - Video Streaming Service
  - Over-the-top Display Service
  - Augmented hair service: receiving as input the output of the Video Recording Service.
  - Face recognition service: receiving as input the Video Recording Service, and the face parameters of people around, generating an event when a person is recognized for the first time.
- The swarmlet receives an event from Face recognition service.
- The swarmlet negotiates with the Reflection service (receiving as input the output of the Video Recording Service) and presents the option to apply changes to the reflection through interaction.
- The swarmlet consult the Personal agent and retrieves the services of interest and negotiates access to them:

  - Agenda service: receiving as parameter the Person ID, generating as output a screen with the service interface.
  - E-mails service: receiving as parameter the Person ID, generating as output a screen with the service interface.
  - News service: receiving as parameter the Person ID, generating as output a screen with the service interface.
- Electronic mirror swarmlet allows access to the services: agenda, News and e-mail.

3.2.2 Bell used by two scenario

In this scenario, two Swarmlets require the use of the same resource, a bell. A wellness application agent wants to use a bell in order to remind a person about the time to take a medicine and a security application agent wants to use a bell in order to advertise a person at the door. The bell resource is reserved for both, but with a partial reliability agreement.

3.2.2.1 Use case description for the bell used by two scenario

Pre-condition:

- Two independent applications running over a swarm enabled network: wellness application and security application.
- A Bell Service is offered in the area of interest.

Actions:

- Wellness application communicates with the Broker in order to identify the available bell services near to the user.
- Broker recommends the best matches to the Wellness Application Agent requirements.
- Wellness application negotiates with the Bell Device Agent (it can be done through the Broker) the usage of the bell and the resources are committed (new cell is created).
- Security agent communicates with the broker looking for a bell service.
- Broker recommends the best matches to the Security Application Agent requirements.

- Security application negotiates with the Bell Device Agent (it can be done through the Broker) the usage of the bell and the resources are committed (new cell is created).
- Bell Device agent re-negotiates the contract with Wellness application limiting the quality of service.
- This way the bell is shared and both agents can use the bell under an agreed QoS (not with a full reliability).

3.2.3 GYM Scenario

The Gym scenario considers the relationship between personal agents and a Gym Swarmlet. The Gym Swarmlet will select music, set up equipment and air conditioning temperature according to user preferences that are stored and retrieved by each user's Personal Agents.

3.2.3.1 Use case description for the GYM scenario

Pre-condition:

- Personal agent running on user's Smart Phone.
- Gym Swarmlets running at the gym: network, music, equipment and classes.

Actions:

- User enters in the gym and its personal agent identifies the available networks.
- Personal Agent identifies a network from a commercial partner and asks for connection.
- Gym network swarmlet communicates with the Personal Agent and authenticates the user as a costumer.
- Gym network swarmlet allows the user's Smart Phone to connect to the network.
- Gym network swarmlet makes available a website to configure user's preferences.
- A swarmlet running in the Gym's network, Music, identify the Agents nearby and collects the users' musical preferences
- The swarmlet Music puts together a playlist and plays it.

- Gym equipment agent detect the user's Smart Phone by proximity
- Gym equipment agent discovers the user's Personal Agent and authenticates the user and his device
- Gym equipment agent looks for the trainningInfoService and access the user's train and use it
- The gym classes swarmlet send alarms about the classes start based on user's configured preferences or on system recommendations

Obs.: The current scenario (not Swarm enabled) would be: user enters in the Gym and connects to the local net with a password informed by the front desk. User download, install and execute the gym application. In this application, the user configures its preferences regarding music and classes. When the user is authenticated, the application downloads the user training and synchronizes it with equipment in use. The music in the gym is selected according to the pool of users' preferences from the ones that are in the gym at the moment.

3.2.4 Building Scenario

A building allows any person to control its elevator and lighting (specially relevant to a paraplegics, for example) through a personal device. Allow automatic feedback from a person to the building about the person localization as well as temperature and acoustic discomfort.

3.2.4.1 Use case description for the Gym scenario

Pre-condition:

- Personal agent running.
- Building agent running.
- User has no access permission to the Emergency Alarm System.

Actions:

- The user enters in the building and its personal agent identifies the available networks;
- The user's personal agent identifies the public network form the building and asks for connection;

- The building agent allows the Personal Agent connection;
- Personal agent asks for the available services and presents an interaction screen to the user;
- User interacts controlling the elevator or the lighting system, for example (direct interaction of the smart phone to the elevator control system or the lighting control system).
- Building agent looks for localization service;
- It finds the Personal agent localization service and uses it;
- Personal agent informs its position;
- Personal agent identifies the Emergency Alarm System and requests to use it.
- As it is a protected service, the service provider tries to authenticate the Personal Agent in a specific authentication service and it fails.
- The permission is denied and the service cannot be offered to the Personal Agent.
- In emergencies, all devices can be controlled by a superior entity (PKI allowed) unless explicit forbidden by the user.

Obs.: The current scenario would be: user enters in the Building and connects to the local network with a password informed by the front desk. User download, install and execute the Building's application. This application runs on a centralized automation system from the building that does all the desired features.

3.3 SWARM OS GROUNDING / BINDING

This section will discuss the grounding (i.e. binding) elements of the Swarm OS – Encoding, Messaging, Transport and network, Addressing and Security. In this thesis, grounding/binding refers to how to access services. It includes all communication levels and the protocol to actually use a service.

Figure 14 – Binding sub-elements

Source: Author

3.3.1 Encoding (Description + Formatting Data + Structure)

It refers to the syntaxes and templates defined to describe services, devices and policies. There are two main types of syntax description: based on tables or on markups. The markup description is easier for humans to read and write; it associates labels to data contents. On the other hand, tables define containers of information specifying the meaning of each group of bits. The IEEE Audio Video Bridging and the MPEG standards are examples of the table usage for description and control of devices. On the other hand, JSON (JavaScript Object Notation, RFC4627 (CROCKFORD, 2006), and XML (eXtensible Markup Language) (BRAY et al., 2008) are examples of markup languages. The JSON format is derived from the JavaScript scripting language serving as a simpler alternative to XML. Thus, this work will make use of JSON for encoding.

Descriptions based on syntax only works well for preprogrammed systems, but dynamic systems that can automatically discover, invoke, compose and monitor services, require semantics mark-ups. By describing services using formal and declarative semantic mark-up, an engine can automatically place new service descriptions at the correct place in the concept hierarchy. Descriptions interoperability can be achieved through reasoning over semantic descriptions. Two services can be semantically equivalent and provide the same interface syntax; or one can be a sub-class of the other; or they can be semantically equivalent but use different syntax, and therefore require ontology translation by a specialized mediator service, as described by Obitko and Marik (2005) and Burnstein (2004). This work will adopt WSMO for semantics description. Our approach uses semantic description, but not as a mandatory service, the Mediation will be offered as a service. This approach facilitates the integration of devices with scarce resources.

Figure 15 - XML, JSON and table description examples

```
<Books>
    <Book ISBN="0553212419">
        <title>Sherlock Holmes: Complete Novels...
        <author>Sir Arthur Conan Doyle</author>
    </Book>
    <Book ISBN="0743273567">
        <title>The Great Gatsby</title>
        <author>F. Scott Fitzgerald</author>
    </Book>
    <Book ISBN="0684826976">
        <title>Undaunted Courage</title>
        <author>Stephen E. Ambrose</author>
    </Book>
    <Book ISBN="0743203178">
        <title>Nothing Like It In the World</title>
        <author>Stephen E. Ambrose</author>
    </Book>
</Books>
```

(a) XML

```
{
    Name: "Rick",
    Company: "West Wind",
    Entered: "2010-12-30T10:00:00Z",
    Value: 0,
  - Address: {
        Street: "32 Kaiea",
        City: "Paia",
        State: "HI",
        Zip: "96779",
        Country: "US"
    }
```

(b) JSON

Bit	Field Value	Function	Meaning
15	0001 16	IMPLEMENTED	Implements an AVDECC Listener.
7-14	—	—	Reserved for future use
6	0200 16	OTHER_SINK	The AVDECC Listener has other stream sinks not covered by the following.
5	0400 16	CONTROL_SINK	The AVDECC Listener has control stream sinks.
4	0800 16	MEDIA_CLOCK_SINK	The AVDECC Listener has media clock stream sinks.
3	1000 16	SMPTE_SINK	The AVDECC Listener has SMPTE time code stream sinks.
2	2000 16	MIDI_SINK	The AVDECC Listener has MIDI stream sinks.
1	4000 16	AUDIO_SINK	The AVDECC Listener has audio stream sinks.
0	8000 16	VIDEO_SINK	The AVDECC Listener has video stream sinks (which can include embedded audio).

(c) Table

Source: Various

3.3.2 Messaging

In open systems, messages can be used to make independent agents work together by sharing the appropriate messages and using narrow, agreed-upon, interfaces, but without any single conceptual integration. Accordingly to Singh and Huhns (2005), Integration indicates that the given components or resources are pulled together into one logical resource with a single schema, whereas the terms interoperation means that the given components work together. Integration is fragile, meaning that when one of the integrated services changes, it may affect the integrated whole. In order to obtain interoperation, the messaging system could rely on direct commands or documents exchange. This is a question of procedural versus declarative representations, respectively. Declarative representations have the advantage of enabling standardization, optimization through sophisticated tools, and improved

productivity in terms of capturing and reusing knowledge. However, declarative approaches sometimes have a greater startup cost, because they require deeper and cleaner conceptualization of the domain of interest. Further, declarative conceptualizations rely upon a reasoning engine for processing and such engines can have poorer performance. Open systems, because of the autonomy and heterogeneity, demand flexibility and open representations, being the declarative modeling well suited.

For this reason, the document-centric view, that considers services as business relations, coheres better with applying services in the Swarm open environment. This work will use the document centric allied to the remote procedure calls. The Remote Procedure Call view is a natural choice for the use case of making independently developed applications interoperate.

In a service-oriented architecture, there is the service provider and the service consumer that is typically designed as a server-client communication, where the server is a reactive and stateless process and the client is the triggering and reasoning process. In a machine to machine environment many devices acts as server as well as client, in order to collaborate with others. As detailed in the State of the Art review, the SOAP web services support remote procedure calls using XML documents in a generic way and the RESTful web services, using HTTP commands applied to resources. As HTTP is lighter and is spread over the Internet, it makes sense to take advantage of this capability adopting RESTful web services as a framework for providing messaging in the Swarm. On the other hand, in order to include the constrained devices do the system, the CoAP standard is an important asset that could extend the network through gateways in order to put these devices into the system.

3.3.3 Transport and network

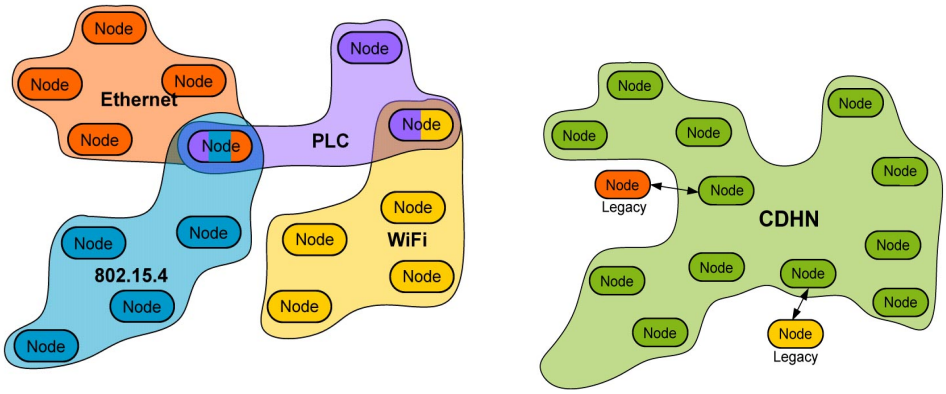
Transport and network deal with data packets generation and transmission, it refers to network and connectivity. The most common transport layers of the OSI model technologies are: TCP/IP (Transmission Control Protocol) and UDP/IP (User Datagram Protocol). Some application protocols may violate the OSI model layers, using the TCP handshaking for the application protocol. TCP provides reliable, ordered, error-checked delivery of a stream of octets between programs running on computers connected to a network (local, intranet or Internet). TCP has performance problems over lossy links, sensitivity to mobility, it doesn't support multicast and has high overhead for short-lived transactions (EGGERT, 2010). UDP

emphasizes low-overhead operation and reduced latency rather than error checking and delivery validation. UDP does not guarantee packets delivery, or the delivery order. SwarmOS framework is suitable to transmissions over the Internet as well as to machine-to-machine communication scope, using HTTP and CoAP respectively. This way, the framework will adopt a hybrid approach using TCP/IP to support HTTP and UDP to support CoAP and multicast advertisements.

To be part of the network all devices must be connected. This connection is defined by communication systems that usually specify physical, data link and network layers of the OSI (Open Systems Interconnection) model. Nowadays there are many different communication systems, such as: Wi-Fi, Bluetooth and Power Line Communication. As the communication systems are complementary, supporting different particularities of each application, instead of trying to address all the Swarm diversity with just one kind of communication system, our proposal is to use the concept of virtual network. Virtual networks connect the devices logically; taking advantage of nodes that have more than one communication interface, compatible to different communication standards, making these devices to act as bridges (also known as gateways).

Figure 16 (a) shows multiple devices with different communication standards, but some of them have more than one communication interface being able to communicate on more than one communication technology. Using nodes with multiple network technologies to act as integration points (gateways/bridges) generates a (virtual) network independent of communication technologies. Thus network technologies would be transparent to applications and end-users, such as presented on Figure 16 (b).

Figure 16 - Example of devices interoperability in a network.



(a) network with technologies diverisity (b) virtual network

Source: NOWAK et al. (2011)

It is a known problem and some projects have already addressed this issue. There are application level solutions that will generate middleware services or processes, and solutions beneath the transport layer, such as the work of Bruni et al. (2006) and Brzozowski (2011). A relevant work is Inter-MAC (NOWAK et al., 2011), part of the Omega project (EU FP7) (JAVAUDIN; BELLEC, 2011). Considering the OSI (Open Systems Interconnection) network model, Inter-MAC would be located below the network layer but above the MAC layer. This strategy allows the use of a single network address for multiple interfaces on the same device, and enables the matching parameters of quality of service settings of the MAC layer of each communication technology, thus ensuring the quality of user experience. The Inter-MAC approach could be applied to the Swarm framework, as a next step, targeting the efficient virtual network creation, but the framework should be able to support higher-level bridges as service implementations as well.

A key feature in the Swarm OS is the quality of service assurance. The quality of service at the network level relies on the bandwidth reservation and time synchronization. Bandwidth reservation can guarantee to the service consumer a specified bandwidth, keeping its streams of starvation.

Time synchronization is essential to many applications, such as control/automation and multimedia distribution. Audio and video rendering were usually confined to a single device, such as a television, but users are increasingly demanding that audio and video to be untethered from the entertainment center– to use the network to distribute the contents using any devices, possible a different device to play the video and one or more devices to play the audio channels. As human brain has high accurate systems to perceive audio and video, all audio channels must be within 5-20μs of each other (and stationary), and that video can lead audio by as much as 25ms but video may lag behind audio by only 15μs - because human brains perceive late audio as expected, but early audio as unnatural. These requirements are very strict, and the Swarm framework shall be prepared to that. Time synchronization is important also to provide a common time base for sampling/receiving data streams at a source device and presenting those streams at the destination device with the same relative timing.

Audio Video Bridging (AVB) is a common name for the set of standards developed by the IEEE Audio Video Bridging Task Group of the IEEE 802.1 standards committee. The gPTP (IEEE 802.1AS-2011) Protocol is a subset of the IEEE1588 and allows AVB devices periodically exchange timing information that allows both ends of the link to synchronize their time base reference clock very precisely. gPTP, as an 802.1 standard, is defined for many different transports increasingly found in the home, including Ethernet, Wireless

(commonly, Wi-Fi), MoCA, and G.hn. Thus any of these LAN technologies may be used in any combination, and still maintain accurate time. Each of these standards and industry specifications include a description of how they plug into the gPTP architecture. It bears special mention that the measurement utilized by gPTP for Wi-Fi links is defined as the Timing Measurement capability in IEEE 802.11v-2011. It is important that this feature to be offered as a service to the Swarm OS framework, allowing the Swarmlets to interact and control this protocol.

AVB defined also a solution for bandwidth reservation, the Stream Reservation Protocol (SRP). The SRP is specified at the IEEE 802.1Q-2011. SRP is designed to allow the sources of AVB content (Talkers) to advertise that content (Streams) across a network of AVB Bridges, and users of the AVB content (Listeners) to register to receive the streams through AVB Bridges. SRP gives AVB end stations the ability to automatically configure a bridged network and to adjust to engineered networks such as those configured with multiple virtual LAN segments. SRP also protects best effort traffic from starvation by limiting AVB traffic. Listeners can report latency through higher-layer protocols and used, in conjunction with gPTP and the transport protocol, to synchronize the playback of multiple Streams and/or multiple Listeners.

Enclosing, it is important to the system to allow the use of multiple communication systems at the network layer (for efficiency) or at the Swarm OS level, as a service. On the other hand, time synchronization and bandwidth reservation should ideally be provided and have its representation as a service as well.

3.3.4 Addressing

Essential to the services and resources oriented view is the possibility to access resources, devices and services. The addressing is going to enable messages to reach its receiver. The addressing should be able to access a device, but its services and resources, as well as grouping and broadcasting. This work inherits the concept of URI (Universal Resource Identifier) for unique identification of resources and adopts the concept of Namespaces for denoting consistent information spaces.

3.3.5 Security (Authentication / Access Control / Confidentiality / Integrity)

The main security aspects to the Swarm Framework are authentication, access control and confidentiality. The authentication and access control will manage the privileges of the Swarm ecosystem. Authentication is part of the access control; it identifies the entity that is trying to execute an action and enforces the access control based on entity identification and security policies. A suitable mechanism to the access control would be the use of the sandbox paradigm, where the service policies allied to the owner preferences will determine if the action is allowed without authentication or if it needs authentication. If it needs authentication, it could demand yet attribute certificates setting the user privileges. Confidentiality usually is assured with the transmissions of ciphered messages. The integrity assures that the message was not replaced during the communication.

Security can be applied to many layers of the networking stack. Considering a RESTful interface, it would be possible to have security at the physical/data link layer, e.g. WPA/WEP for Wi-Fi; IPSEC at the IP layer; HTTPS (HTTP + SSL/TLS) at the HTTP layer; or application level security.

Security at the IP level could protect data that higher-level security mechanisms cannot, as the IP spoofing for example, where IP packets are created with a forged IP address with the purpose of concealing the identity of the sender or impersonating another computer system. The Internet Protocol security (IPSec) uses cryptographic security services to protect communications over Internet Protocol (IP) networks. IPSec supports network-level peer authentication, data origin authentication, data integrity, data confidentiality (encryption), and replay protection.

Currently HTTPS is very popular. The security of HTTPS is the combination of HTTP with TLS, which uses long term public and secret keys to exchange a short term session key to encrypt the data flow between client and server. X.509 certificates are used, so certificate authorities and a public key infrastructure are necessary. HTTPS provides confidentiality and authentication. Just as HTTP is secured using Transport Layer Security (TLS) over TCP, CoAP is secured using Datagram TLS (DTLS) (RFC6347) over UDP and this support is mandatory. DTLS is a variation over TLS.

Regarding access control, it is possible to have the following permissions in the Swarm OS framework:

- Connection to the network
- Discovery of a device and its services

- Use of a service

We will consider a sandbox scheme in order to control the Swarmlet service accesses, but we will provide an extra access control level that requires a human authorization.

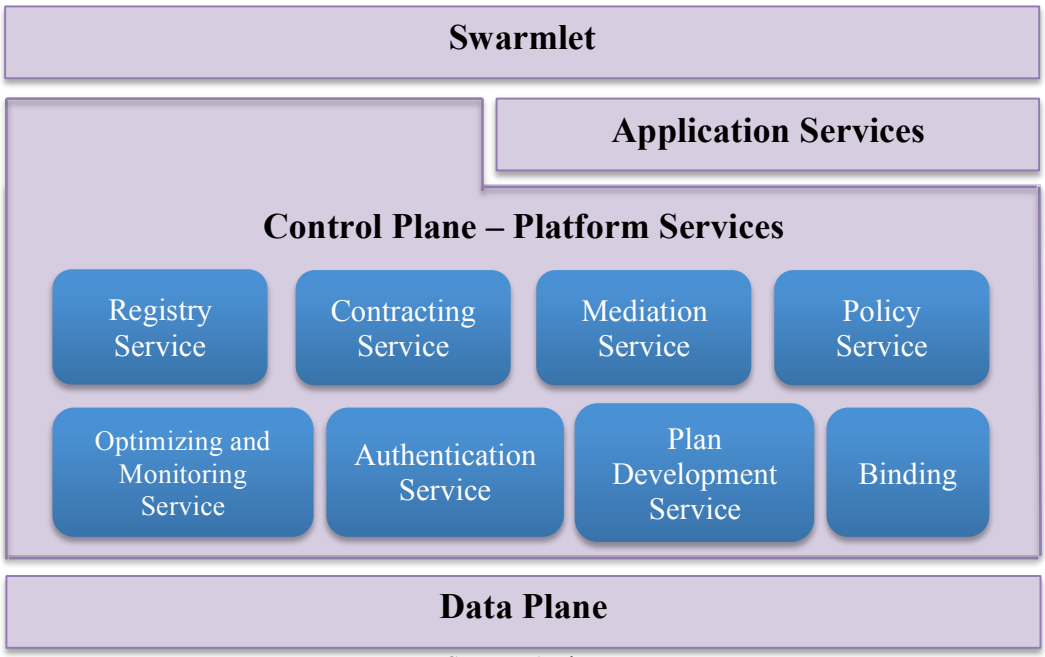
- Authorizing and authentication – it is necessary to have an authentication as well as user manual approval;
- Authentication only – if the service consumer is authenticated, it can use the service;
- Non-secure – it is not necessary to have any security authentication in order to use the service.

3.4 CONTROL PLANE PLATFORM SERVICES

As previously discussed, and presented in Figure 17, the Swarm OS is composed by: Control Plane and Data Plane. Running over them are the Swarmlets that implement the application business logic. As the application is distributed in services, it works as a manager. The Swarmlet will be responsible for the services choreography maintenance, interconnecting the services, feeding their inputs and consuming their outputs. In addition, the Swarmlet will manage the user interface, being it explicit or implicit. Explicit user interface systems require the user to directly send commands to the system, which may be done with the traditional graphical user interface (associated to touch, keyboard, mouse, voice) or with new and alternative interface (e.g. Han et al. (2010) presented a home automation system based on virtual reality). Implicit user interface is the system automation, in which the user does not control the system directly; instead a software agent controls the system on behalf of the user, based on the context and on the learning from previous behaviors. Intille (2002) discussed mixed systems, in which the automatic commands are suggested, but the user interacts directly for actions to be taken.

In order to manage the Swarm resources, the Control Plane needs to be aware of the resources available, to negotiate the usage of those resources, and to ensure that they are actually provided. Figure 17 presents the services composing the Control Plane. They are: Registry, Contracting, Mediation, Policy, Optimizing and Monitoring, Authentication, Plan Development and Binding.

Figure 17 – Swarm OS Architecture



Source: Author

The **Registry Service** maintains a searchable index of all named objects in the Swarm space (services and personas) and their parameters/properties/state. To express a notion of locality, all entries are scoped in both time and space, with the possible addition of auxiliary attributes. The supported functionalities include the introduction of new objects (either by the object itself, by a third party, or by implicit observation), their update and their removal (either explicitly or by means of time-outs or expiration), as well as information queries (extraction). The registry service builds on the distributed database model that underlay the Swarm OS Global Data Plane (GDP), as proposed by Kubiatowicz and Palmer (2014). The information introduced to the registry is automatically distributed within the scope of interest. Access to registry entries is subject to access control. Another option is to use an unstructured peer-to-peer system, as presented by Sharifkhani and Pakravan (2013).

The **Contracting Service** contains all the functions necessary to match service requestors with service providers. This includes finding the set of potential service providers (most often with the help of the registry service), negotiation of the service contract subject to the policies in place, and monitoring contract adherence by the respective partners. A service contract may include usage, traffic and load commitments, QoS, contract duration and scope, as well as payment strategies. In order to maximize its impact, it needs to be supported by a lightweight transactions system (optionally, but not necessarily, coupled to a reward mechanism such as micro-payments).

The **Mediation Service** intends to assist the brokerage process if the direct matching of the descriptors of requested and available services fails. Using a deeper understanding of the semantics of requested services based on access to service ontologies, mediation may suggest alternative services or a concatenation. Mediation uses ontologies in order to map desires and goals to services, to expand descriptions and to translate protocols. Mediation also manages ontologies, expanding and updating them. This service has the potential to make the system able to be scalable and not too dependent on standards - a potentially huge number of distributed and autonomous parties are unlikely to agree beforehand to a common terminology, or to all of them adhere to the same standard.

The **Policy Service** supports all the functions necessary to define, to store and to execute policies. A policy details access privileges to data and services, as well as operational rules regarding the brokerage of resources. One important function of the service is to ensure that deployed policies are consistent (not being contradictory), or to group them into categories for which such consistency is assured. It also may also distribute to agents interested in following a given policy the code necessary to execute that policy. The Policy Service shall organize, unify and cohere policies, taking advantage of the GDP (Global Data Plane) to store, to distribute and to access policies.

The **Authentication Service** provides certification of authenticity to agents (personae), services, contracts and messages (if such certification is required). For this purpose the Authentication service will act as third (trusted) party in selected interactions. It may also be requested to aid in establishing secure communication channels between Swarm agents.

The **Optimizing and Monitoring Service** plays a complementary role: reason about the performance of the swarm based on the behavior observed. Particularly, based on continuous monitoring of the swarm operation, quantitative models can be developed and continuously enhanced (for instance, using machine learning). Trends in popularity of individual services, and evolution of traffic and load scenarios can be identified. This can help to predict and to suggest necessary changes in the number of available resources needed (e.g. communication, storage, computing power and energy).

The **Plan Development** is a framework service that may provide Swarmlets with: Service Composition and Contract Management. The Swarmlet may retrieve a higher-level user interaction and use this framework service, the Plan Development, in order to map this interaction into a graph of services. It will define the parameters that shall be used by the discovery process – that will be executed by the Broker block. The Plan development is

responsible for managing contracts that are registered to it, verifying if the service provided is effective and if it is about to expire, making new commitments in order to maintain the whole execution.

Binding (or grounding) adapts interactions between services, creating an abstraction of the communication. The layers to shape the communications are: encoding, messaging, addressing, transport, connection and security. Binding deals with the adaptation in system layers, dealing with message transmission. Binding is accessed as a service, in which the service provider shall post its own binding information to the Binding Service. Then, when a Service consumer wants to use this service, it will do it through the Binding Service. The binding information may be implemented in many ways; one possibility is to use the Accessors (LEE et al., 2014) strategy, in which the binding information is a JavaScript code embedded in the service description file. This JavaScript contains all the process required to deal with the service inputs and to generate the outputs. It may be necessary for the Binding Service to use the Mediation Service to adapt protocol, in case it is not implemented as an accessor.

3.5 THE CONTROL PLANE BROKER

Due to the heterogeneity of the Swarm scenario, Swarm OS will be built as a distributed Operating System in four layers architecture: a micro-kernel, platform services, application services and Swarmlets. The Control Plane services discussed in the last session are mainly in the category of *platform services,* which go into a lower layer than the Swarmlets and the application services. These services can run anywhere on the Swarm OS distributed platform (in the device or even in the cloud). The only element that needs to be available at any Swarm device is a *micro kernel,* which supports a minimum part of the Control Plane services, embedding the Local Broker, which will be discussed next.

An essential actor in the Swarm OS is the Broker. It assists (as a set of services) service providers and consumers with sealing contracts. The Broker intermediates the access to the Control Plane services, as presented in Figure 19. The service consumer and provider will post their description files to the Broker, as they become active. The description files contain the description of the service and the contract constraints (the required service level agreement).

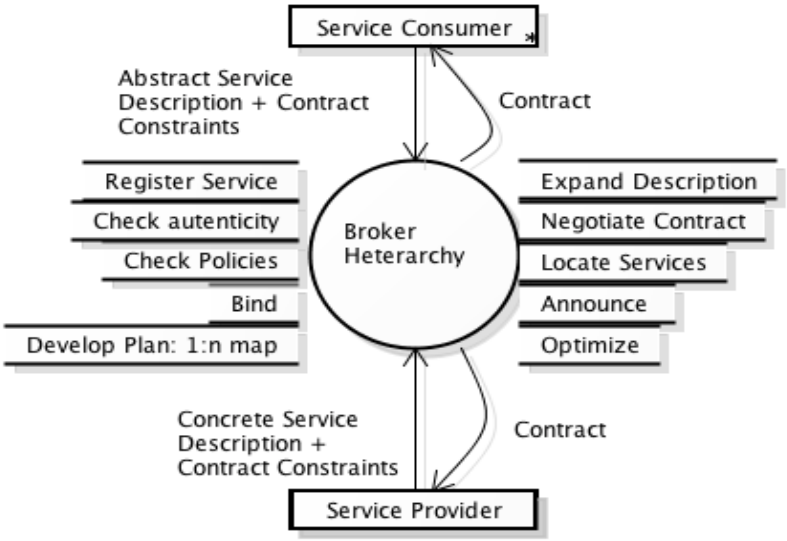
Figure 18 – Services architecture

Source: Author

The Broker will:

1- Produce an entry in the Registry
2- Generate advertisements when connecting to a network.
3- Answering locate and discovery queries when received, looking for Service Providers in their cache or in the Registry.
4- When required by Service Consumers, look for Services Providers that are functionally compatible and suitable to establish a Contract (using the registry, its cache or using locate and discovery calls). At a first level this is a constrained optimization problem, in which contracting can be mapped as an objective function that shall be optimized with respect to some variables when faced with constraints on those variables. The process of looking for suitable providers may:
    a. Expand service description using Mediation
    b. Use the Plan Development to map the required Service to actual Services Providers abstracting a 1:1 service mapping to a 1:n or even to a n:1, figuring out the required orchestrations.
    c. Consult policies to decide about a Service Provider selection.
5- Receive and forward messages for protocol adaptation using the binding description.
6- Optimize and monitor: the Broker may register the commands sent to an entity and look for contradictory commands. In this case, the broker will look for a related policy or send the contradictory commands to the owner agent.
7- Authenticate: the Broker provides APIs to verify signatures and to make key exchanges.

Figure 19 - Broker, Platform Services, Service Provider and Service Consumer Relationship



Source: Author

Considering that even a simple node provides at least one service, this service will have to be advertised and discovered in order to be part of the system. Peer-to-peer discovery mechanisms foresee inquiries about services, as well as voluntary publicity. Hence, a device should provide some kind of local directory service that would be responsible for performing these tasks or would even indicate a third party to do it. Abstracting the heterogeneities in the Swarm scenario for now, we could consider the Control Plane group of services as a brokerage organized in a hierarchy and the first brokerage level is the **Local Broker.** The Local Broker can be connected to a higher-level Broker, using this higher-level Broker as a directory. In case the Local Broker is not connected to any other Broker, it will have to respond to any service search request query by itself.

The Broker heterarchy is a peer-to-peer unstructured system, in which the node that has received a discovery query will look for the response in its own cache and in case of not finding a suitable service, it will forward the query to a registered third party broker. It is also possibe to use the other Control Plane services, so as to expand the service description using the semantic mediation service. As the system is totally unstructured without rigid processing rules, different kinds of Brokerage implementations are possible, as well as nodes with a variety of processing power. Thus, it is possible to have domains of brokers that implement structured peer-to-peer discovery mechanisms, using a solution as the distributed hashing tables, as presented in Schmidt and Parashar, (2004), Ranjan, Aaron and Buyya (2008) and Kang, Choi and Kim (2008).

Considering a Local Broker in each device, Service Providers and Service Consumers shall communicate with the Local Broker in order to establish a relationship. The Service Provider shall make available to the Broker heterarchy: its own description, its contract constraints and its binding information. The Service Provider Description defines its interfaces and its functionalities. It may be standardized and/or have semantic notations and process descriptions. Standardized services will provide a category code that has pre-defined interfaces and features. Another option would be having a semantic description of the service. The Service Provider Contract Constraints present a table or a function with the relationship between criteria. For example, different quality of service maps to specific billing values. Those commitments may be variable in time; in this case, an address is specified to push and to read updated Contract Constraints. The Binding information will abstract the communication protocol information. There are some proposals for Binding implementation, e.g.: WSDL could be used, as well as the Accessors model that uses Java Script as the binding description language. If a Service Provider presents a Binding, all the communication between Service Consumer and Provider will be conducted through the Binding Service (implemented as a man in the middle (BISHOP, 2004)). On the other hand, the Service Consumer shall inform the broker about the required services description and the contract constraints for each required service. Both service description and contract constraints are compatible with the Service Provider ones.

3.6 ARCHITECTURE WORKFLOW

In this section, we describe workflows of the connection between the control plane services, broker, service consumer and provider.

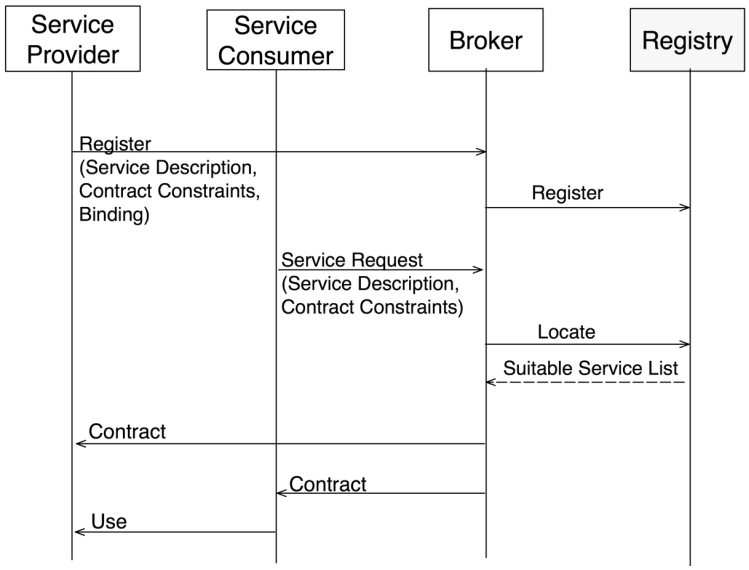
3.6.1 Registry

When a Service Consumer (in fact, its representative agent) is looking for a Service, it will pass the service description and the desired parameters to the broker. The Broker will look for suitable Services in the registry, and given the available options, it will negotiate with the suitable Service Providers to set up a contract. Once the contract is established, Broker, Service Provider and Service Consumer will have copies of the contract and Service Consumer may start to consume the service. Such scheme is pictured in Figure 20. Often, the usage of a broker can be avoided; for instance, in case service provider and consumer can

directly agree on a contract without negotiations (in case of pre-defined partners, for example).

In the proposed architecture, the process of discovery is implicit and is centered on the registry service. The first layer of discovery starts within the device itself. Even a simple device provides at least one service, and this service needs to be advertised and registered in order to be part of the system. Hence, a device should provide some kind of local registry service (or have a proxy for it) in which all its services are registered. When a Service Provider starts, it advertises itself to the local Registry. After that, it will be responsibility of the Swarm OS, when the device gets connected in a given setting, to ensure that the information is propagated within the scope of interest. Service consumers may subscribe for new service provider advertisements.

Figure 20 - Swarm Control Plane Workflow with Registry and Contracting Services



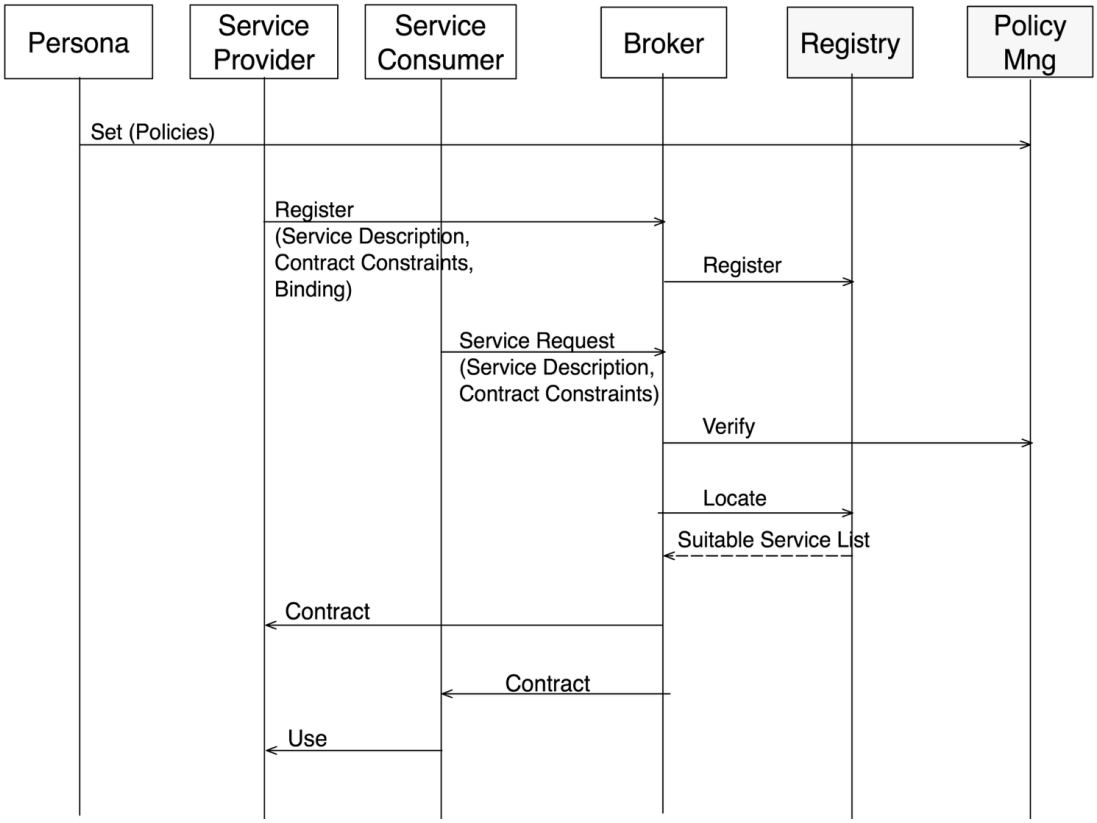
Source: Author

Security is a transversal matter that affects Discovery. The privacy is assured embedding requester's Principal in discovery queries, allowing devices to decide whether it will answer to it or be invisible to this consumer.

3.6.2 Policy Service

Figure 21 presents the workflow with the Policy Service, where the Broker is considering policies in order to establish the negotiation rules and priorities.

Figure 21 - Workflow using Policy Service.

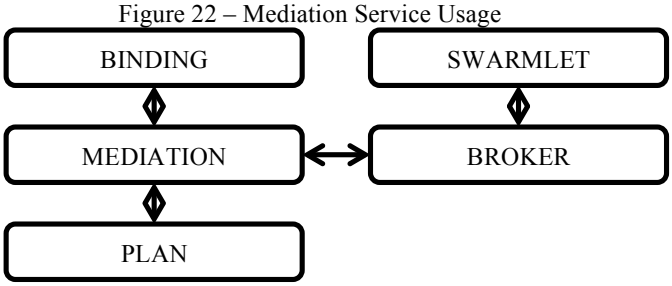


Source: Author

3.6.3 Mediation Service

The Mediation Service may receive requests form the Broker, Plan Development and Binding. The Broker will demand to expand service descriptions to support discovery processes, as well as to expand the current Ontology. The Plan Development will use Mediation and access Ontologies in order to develop plans and to map goals to services. The Binding Service may require the semantics processing in order to adapt communications protocols.

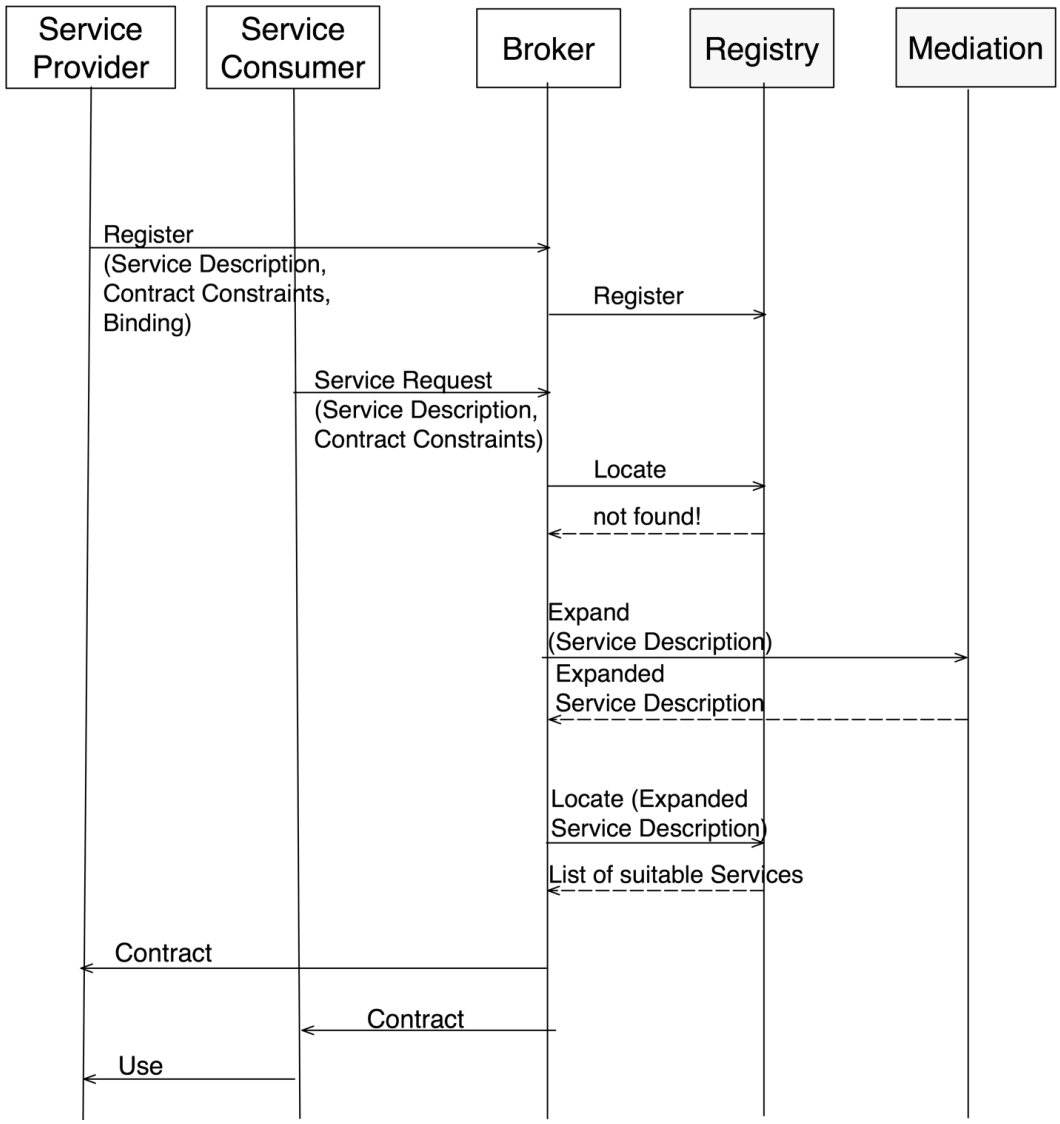
Figure 22 – Mediation Service Usage



Source: Author

Figure 23 presents the sequence diagram to the use of Mediation for service description expansion. The Service Description expansion consists on using Semantic Computing to expand the descriptions, looking for equivalences in the terms used in the service description file.

Figure 23 – Mediation workflow example



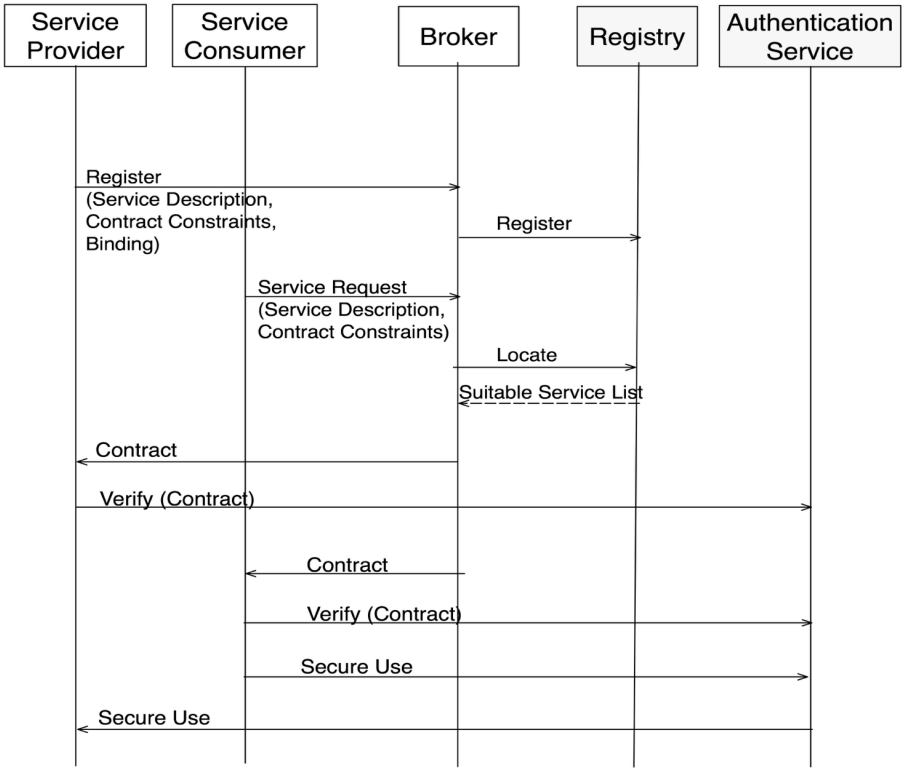
Source: Author

As discussed before, Swarm OS supports mediation with semantic reasoning, helping among other processes, the discovery. This way, the discovery may be done in a traditional way, using services identifications (name, type and category); as well as using the complete service description, contemplating semantics.

3.6.4 Authentication Service

The Authentication Service verifies signatures and permissions, providing digital and attribute certification using PKI (Public Key Infrastructure) (ITU-T, 2012). As presented in Figure 24, it may be used to verify contracts validity, to authenticate Agents or to establish a secure channel.

Figure 24 – Embedding the authentication service

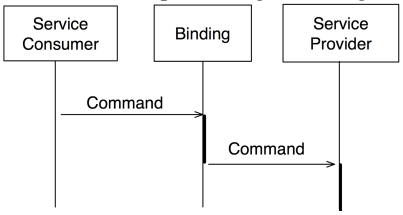


Source: Author

3.6.5 Binding Service

The Binding Service is used as a bridge to communicate, adapting communications from broker to broker, as well as between services, as presented in Figure 25.

Figure 25 - Control plane diagram using binding

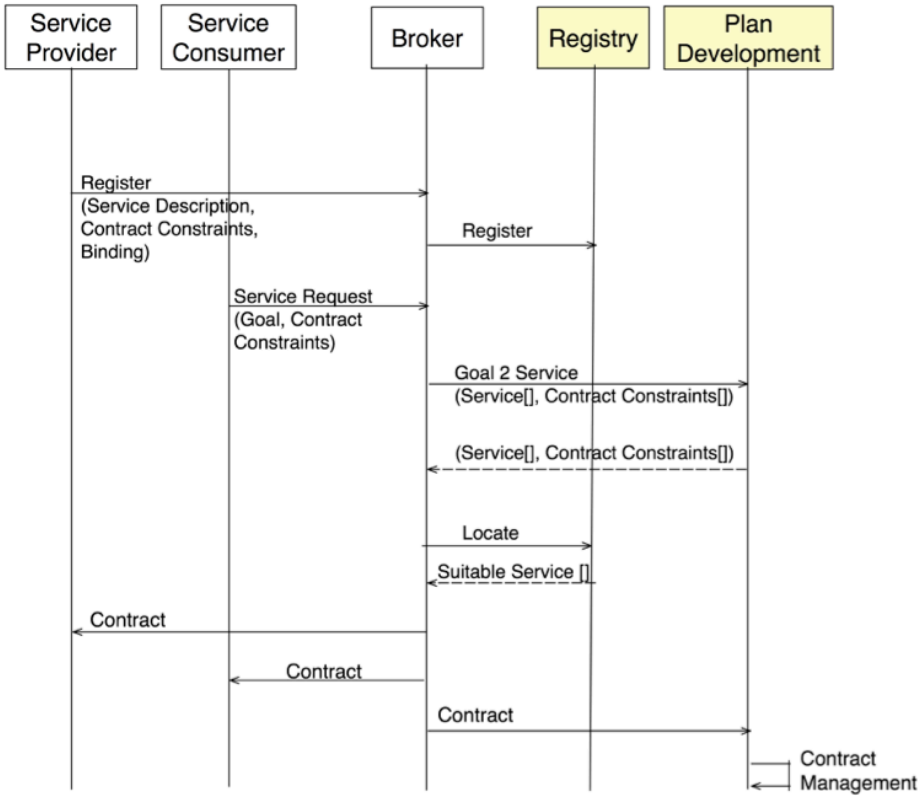


Source: Author

3.6.6 Plan Development

The plan development flow is presented in Figure 26. In this flow the Swarmlet has a higher semantic level of the user interaction - a desire or a goal – so the swarmlet decides to use the Plan Development in order to determine how to accomplish this goal putting a graph of services together. In order to do that, the Plan Development may use of the Mediation service to access the ontology and the semantic processing APIs. When the plan is developed, the Plan Development uses the Local Broker to locate services providers and to negotiate contracts. After the contract is consolidated, the Plan Development may be used to guarantee that a suitable contract will be always available, starting the service request, when it is necessary.

Figure 26 - Control plane sequence diagram – Plan Development



Source: Author

3.7 PLATFORM SERVICES INTERFACES

In this section, services interfaces are proposed in order to allow services consumers and services providers to access the control plane features. The interfaces are separated in contracting, discovery and binding interfaces.

3.7.1 Contracting interfaces

The contract is an agreement about service provision including commitments pertaining to time and location, QoS, and load provision. The Contracting activity is a constraint optimization problem that shall result in the selection of the contract terms, helping to select between suitable services.

The service providing agreement may be done using take-it-or-leave-it or negotiation mechanisms. The take-it-or-leave-it contracts are published and if there is any entity interested, it could use the contract “as is” (without modifications). The negotiation process is based on contract constraints, establishing the conditions to each quality of service and context. The contract constraint publisher can be a service provider or a service consumer. The negotiation mechanism requires a communication to offer counter proposals until agreeing on the contract terms. We can consider auction as a kind of negotiation process.

The world, as well as the Swarm ecosystem, is a place with finite resources, what could lead to situations where the resources are constrained and, even if the service is existenting, more than one consumer is disputing it. In order to solve the dispute, we could consider some strategies as: auction or priority. The simpler priority system is to use the request chronological order, where the priority is given to the requester that first posted a demand. Other interesting option would be to have an infrastructure with support to a reputation system, where the most an entity provides non-paid services, more rewarded it is; and the more rewarded, higher is the priority. Other important mechanism to prioritize requests would be to identify ownership and group credentials. This way a service could be more reliable to a person or to a pre-defined group of people.

The contract terms shall consider:

- Quality of service: it is the overall performance of the service providing. The main parameters that defines the quality of service are:

    - Resource reservation: defines how much of the resource will be reserved to the service consumer, e.g. a network service provider could allocate 20Mbps of its bandwidth.
    - Reliability: defines, as a percentage, the availability of the resource at the contracted amount. As the services usually provides a resource sharing mechanism to multiple consumers, it is possible to overbook a resource, considering that all the consumers will not use it at the same time. The reliability will give the probability that the contract will be honored.
    - Delay: defines the expected delay time to receive the service. It may include processing power and network impacts on the service providing.
- Expiration: the expiration define date and time after which the contract is not valid any more.
- Commitment: it defines the kind of commitment that is being stated. It could be one time execution or multiple time execution with the quantities defined ahead or by a period of time.
- Billing: defines the terms of the payments for the services providing. It should define the payment schedule for the service providing in conformance to what has been negotiated.

Following our REST oriented architecture; the transaction process is executed with:

- GET on the "<service address>/contract" to retrieve the negotiation policies of the service (it contains the types of transactions supported and may contain a price table listing the quality of service parameter combinations and its prices. If the service provider works just with the take it or leave it; the GET will retrieve the proposed contract.
- A POST to the negotiation interface with the proposed contract or the contract constraints will be interpreted as a proposal.

Using the Broker, the Service Provider will POST the Contract Constraints combined to the Concrete Service Description and Binding at the negotiation interface (<broker address>/negotiate). On the other hand, Service Consumer will POST its own Contract Constraints combined to the Abstract Service Descriptions that it is looking for. Contract Service Description refers to a service that is actually being provided; its description includes

grounding information. Abstract Service Description refers to the service description used to reques for a service; it does not contain grounding description.

The proponent may sign all the proposed contracts and the acceptance shall be marked by the success message for the POST query with the contract signed by the other part as well. The negotiation may be executed sending a new contract proposal instead of the signed contract.

In order to make the system more efficient, a given service could be considered Latent when a previous contract is existent. This way, all the negotiation process could be replaced by previous contracts presentation to the Service Provider. This way, the services states regarding the resources commitment are:

- Unavailable: no contract established, no resources committed, no capabilities being provided, no API available. E.g.: Device is off.
- Available: no contract established, no resources committed, no capabilities being provided, API available;
- Running: contract established, resources committed, capabilities are being provided, API available / resources contracted are being used;
- Latent 1: previous contract established, no resources committed, no capabilities being provided, API available. The service consumer can try to use the previous contracts (if the resources are available, it is faster; otherwise they have to re-start negotiations);
- Latent 2: contract established, resources committed, no capabilities being provided, API available.

An additional feature is the contract maintenance. In order to have a service of continuously update contracts when its expiration approaches, a valid contract may be registered to a specific address.

Contract Maintenance
Service Consumer -> Broker
POST @ ../plan_development/contract_maintenance: {Contract}
Return with: OK / NOK

3.7.2 Discovery interfaces

In order to have services composition and agents cooperation, the available capabilities must be advertised by services providers and discovered by service consumers. In the SWARM ecosystem, data and resources of real world objects will be available globally and in large amounts and will be hosted in a widely distributed heterogeneous system. Therefore, a discovery mechanism that allows accessing them is needed, even if its location and form are unknown to the requester.

Considering a mature and successful scenario of the Swarm, a perfect brute force search for a service by its functionalities could return billions of service provider candidates. This way, it is important to have a discovery strategy that returns the results in waves. The waves strategy refers to restrict services searches, and iteratively broaden it up to find a matching service. The relevance order could consider quantities, or aspects as location and quality of service, for example. Quality of service will consider the capability to offer the service in certain qualities levels, as the interaction delay, for example. Sometimes the localization could be used as a low service delay guarantee, but a superior connectivity system, is capable of compensate physical distance.

The services discovery can be done through direct access between services requester and providers or using a service registry (MIAN; BALDONI; BERALDI, 2009). The services registries maintain a list of available services helping search and advertisement processes; they could aggregate additional features providing advanced services recommendation, for example. The direct access allows services providers to advertise their services, as well as the service requester to send messages specifically to a device to require a service or to a group of nodes asking for information about services.

As the goal is to have a system that scales globally, counting on heterogeneous devices, it is reasonable to consider that some devices could provide registry services, allowing the organization of domains around registries, in a heterarchy way. As a device can have multiple processes executing concurrently and these processes can be independent, each of them may actually work as service provider or consumer, acting as a local registry. In order to avoid tasks replications, the discovery services will be considered as a platform service, being provided by the Swarm OS control plane.

Conceptually, the types of directories/indexes are: (1) “white pages,” where entries are found by their name, (2) “yellow pages,” where entries are found by their characteristics and capabilities, and (3) “green pages”, accessed through the others, describe how the service can

be used, it is a nested model comprising business processes, service descriptions, and binding information (OASIS, 2004). In our system this three categories shall be covered.

Usually one specific device offers the same services set and it usually connects to the same networks; this way, the service requester may keep a local cache of suitable services, optimizing the process. Advertisements, services listing and service description messages do not have to contain the entire service description, instead, a service description hash code shall be used and the whole service description is sent just if it is explicitly required. This kind of approach reduces the size of exchanged messages in the system.

Discovery is specially challenging in the swarm scenario, because it considers billions of computing devices making available its many resources without local networks boundaries. Similar challenges are faced by the DNS (Domain Name System), used to map URLs to IPs on the Internet. DNS uses a distributed and hierarchical solution, due to the size of the Internet it would not be feasible to store all pairs domain name and IP address in one server (all Internet would depend on one server, the data traffic would be enormous to read it and the writing too, and it would present delay due to the geographic distance from nodes to the server)[2].

The DNS-SD (DNS Service Discovery) (CHESHIRE, KROCHMAL, 2013) is an extension of the DNS that uses the same infrastructure to make available a mechanism to look for a service in a given domain name. Given a domain and a service type, the DNS-SD lists the available services. Our problem cannot be solved with a direct transposition of the DNS-SD strategy because in many times, the service requester does not have the domains already identified. In addition, the semantically described services have a complex description, with many fields that should be used as comparing parameters, being the service type too poor to provide an efficient searching query.

As introduced in SSDP – adopted by UPnP and DLNA – we will use the discovery with advertisements strategy. When a service first comes on-line it will send out an advertisement so that everyone knows it is there. At that point it shouldn't ever need to send out another advertisement unless it is going off-line, has changed state or its cache entry is about to expire. Any clients who come on-line after the service came on-line will discover the desired service by sending out a discovery request. The client should never need to repeat the discovery request because any services that subsequently come on-line will advertise themselves. The end result is that no one needs to send out steady streams of messages. The

[2] http://tools.ietf.org/html/rfc1034

entire system is event driven: only when things change will messages need to be sent out. The cost, however, is that the protocol is more complex. For the networks with a broader scope than local area network an unstructured peer-to-peer mechanism will be used, but this definition is out of this thesis scope.

In order to protect privacy it should be possible to a device be invisible to discovery queries, so the responses to discovery queries are not mandatory. A device could opt to be invisible to some discovery queries, taking advantage of the identification of the requester, to select to which ones it will respond. This way is important to have the option to send a Principle within the discovery query parameters. The Principle is the information that authenticates an entity. It could be its digital certificate (ITU-T, 2012). If the requester does not include its Principle, the provider can just ignore the query.

As we are adopting a resources oriented approach, as well as in CoAP, a server is discovered by a client by the client learning a URI that references a resource in the namespace of the server. This is performed using a GET to a well-known relative address of the service provider, which returns a payload with the hyperlinks that refer to the available resources.

Multicast messages shall be used to find service providers under a scope that supports IP multicast. When a server's address is already known, unicast discovery is performed in order to locate the entry point to the resource of interest. As in XMPP, the discovery queries can be used hierarchically, in which some nodes are branches (i.e., contain further nodes) and some nodes are leaves (i.e., do not contain further nodes). Where if the discovery query returns additional items, it is possible to send a discovery query to this item in order to retrieve more information – using a specific URI or a shared URI plus a label.

As used by JXTA discovery, Swarm OS will allow discovery parameters definition, such as: searching scope, a threshold that defines the maximum number of results that each peer should provide. The responses count with an Expiration tag, informing for how long this tag should be considered. The service description may also be included in the query message in order to filter the number of responses or the number of links returned in a response. This service description will be a semantic-described document.

A resource directory can be implemented, by the registry requesting the resource descriptions of other end-points (GET) or allowing servers to POST requests to a “well-known relative address on the resource directory. This, in turn, adds links to the resource directory under an appropriate resource. Any client can then discover these links by making a request to a resource directory lookup interface.

In the future, Swarm OS may adopt mechanisms specifications about how multiple registries may form a group, or an affiliation, and how to permit policy-based copying of core data structures among them.

Putting it in a nutshell, the services discovery shall allow:

- Direct access between services requester and providers as well as through registries;
- Discovery using semantic descriptions and searches based on ontologies expansions and on reasoning processes;
- Discovery queries responses with filters by service type/category or service description;
- Discovery queries with parameters of scope (location, network) and threshold (number of items);
- Discovery queries with Principal parameter (in order to authenticate the requester);
- Two layer responses with description-hash and URI; and a second level with the semantic descriptions. The URI could be from a third party;
- Response with the advertisement expiration: the expiration will vary accordingly to the device cathegory, being later to fixed devices and sooner to mobile, battery-powered devices.

3.7.2.1 Discovery / advertise message

The Local Broker is responsible by sending the Discovery Messages when it connects to a network. These messages are multicast (in UDP) and the devices that receive the Discovery Messages answer with a unicast containing the list of the services that are being provided. The Local Broker of each device or Third Party Registry Services listens and answers Discovery Messages.

Discovery
Broker -> Broker (Multicast)

Discovery answer
Broker -> Broker (Unicast)
Post @ …/discovery : {serviceList}

3.7.2.2 Locate message

Service consumer looks for a service using its descriptions: this query will be directed to the Local Broker, that will first look for the service in its own local cache. If there is no service match and it is connected to a third-party Directory, the query will be redirected to this Directory. If there is no relationship with a third party Directory, the local Broker shall multicast the message to its neighbors. If the neighbors have a service in its local cache or a self-provided service that matches the query, it returns a message with the service description. Otherwise, it will look for other Broker to expand the searching or to send it to a Mediation Service to expand the service description.

When receiving this message, Local Broker will look into its cache for the service:

- If it finds a service that fits to the required configurations, it will negotiate with the service provider broker
- If it cannot find a service, the local Broker contacts for a third party Directory / Broker and passes the locate query.
- If it cannot find a service, the local Broker broadcasts the query to multicast addresses to look for the service.
- If it cannot find the service, it can ask for a service description expansion to the *Mediator* service (if there is any mapped one).
    - It asks for the potential service providers for their Ontologies
    - It sends the service descriptions and ontologies to the registered *Mediation Service*
    - The Mediation Service returns the Mediated Service Description to the Broker that passes it to the Swarmlet.

Locate

Broker -> Broker or Broker -> Registry

POST @ {../locate} : serviceDescription, responseTo, requestID

- "responseTo" : address to receive the response to the Locate query.
- "requestID": identification number that helps the local broker to make the reference to the consumer requesting. It is used in order to identify the service consumer; specially when the broker forwards the requesting away (optional)
- serviceDescription: can be the category type or the whole description. In the case of the whole description, it contains:
    - Scope: identifies the region where the service shall be provided (optional)
    - Principal: information that identifies the requesting entity securely (optional)
    - Threshold: (optional)
        - unitsThreshold (information about the minimum number of services that you want to compare),
        - hopsThreshold (maximum number of hops),
    - Contract Proposal (billing conditions, quality of service) and negotiation policies. (Optional – if this field is not available, it will retrieve the first free service found with any quality of service)

Answer: POST @ {Address}/discovery: {serviceDescription}

- Address: refers to the "responseTo" defined in the Locate request. It may be composed when the "requestID" is informed (<responseTo>/<requestID>).

3.7.2.3 Service details request message

The service consumer or the service consumer Broker requires information about a specific service using the Service address. If this service provider cannot provide a self-description, it will return an error with the address of an alternative address to consult its description. If it can provide its self-description, it will be provided in the message. It will retrieve the service description, its Owner address and its Broker address (when available). The response contains the document with the Service Description

Retrieve Service Details
Broker -> Informed address
GET @ {<informed address>}

3.7.2.4 Share ontology

Broker requests an ontology, in case the Broker has access to a semantic reasoning service and it doesn't find a matching service, it shall conciliate different ontologies in order to expand descriptions, increasing the range of searches queries. In order to do that the not conciliated ontologies will be required. After receiving the ontologies files, if they are different, the Broker may use the ontologies files to look for suitable services using the Mediation Service.

Share Ontology

- Broker -> (ServiceProvider or ServiceConsumer)
- GET @ …/ontology
- Return with: ontology file

3.7.2.5 Plan development

The *Plan Development Service* is responsible to breakdown a higher-level service description to actual services, using the Mediation Service, if it is necessary. When looking

for a suitable service, the Broker may extend the searching considering the mapping from 1:n or n:1. It will not provide external interfaces.

3.7.3 Binding / grounding

Service provider posts its own binding information about the service: this is a unicast message requiring for a specific service binding description (that is the practical usage information – green pages). After that Service provider sends a message containing its services binding: this is a unicast message sent under request. It contains the service address, the service binding description, its Owner address and its Broker address (when available). It is a GET at the binding address.

3.8 DISCUSSION

This chapter presented the Swarm OS Control-Plane proposal. It defined its main elements, taxonomy, architecture, platform services, workflows and interfaces. We could achieve a flexible and scalable computational architecture proposal that potentially will dynamically and opportunistically integrate heterogeneous networked devices so they can synergistically and organically autoconfigure themselves to perform complex tasks.

Considering the heterogeneity of the Swarm scenario, the Swarm OS is proposed as a distributed Operating System with four layers architecture: a micro-kernel, platform services, application services and Swarmlets. Control-Plane services are mainly in the category of *platform services,* which go into a lower layer than the Swarmlets and the application services. These services can run anywhere on the Swarm OS distributed platform (in the device or even in the cloud). The only element that needs to be available at any Swarm device is a *micro kernel,* embedding a Local Broker that locally implements the minimum part of the Control-Plane Services. There is a Local Broker embedded in each device; the Brokers organize themselves as a heterarchy that leverages the access to the services distributed between the devices in the Swarm.

The use of semantic computing, in the Swarm OS Control Plane, leverages the consolidation of a scalable and organic system. The proposed mediation service is capable of giving flexibility to the system regarding standards, i.e. it is possible to expand vocabularies for service descriptions and to adapt protocols.

Devices synergistically cooperate with each other offering its resources as web services composing simpler services to perform complex tasks. This cooperation is based on quality-of-services assurance and micro-payment systems using contracts. The dynamicity is guaranteed with the definition of contract states (unavailable, available, running and latent), a simple negotiation mechanism that pushes new contract proposals and a Broker to monitor and support contract changes.

Available resources are advertised in the Swarm and Brokers provide assistance to service providers and consumers in order to seal contracts. Service providers and consumers are able to opportunistically seal new contracts, without pre-definitions at the programming time.

Swarm autonomous characteristics with implicit user interfaces (pro-active behavior) are supported by a multi-agent system, with multiple personas collecting preferences, context and defining policies. Persona is a virtual representation of a player in the swarm world. Players can be people, organizations, or groups thereof. It consists of identification (represented by biomarkers, keys or others), configuration, preferences and/or state.

Despite the Swarm OS Control Plane proposal presented in this chapter constitutes a contribution to consolidate a Swarm, there is a long way down in order to consolidate a framework to the Swarm, potentially connecting billions of devices worldwide into a single platform abstraction. A next step to the Swarm consolidation is the development of a formal Swarm OS Control Plane Ontology, including service list, description, persona and contracts. As this thesis has as focus the architecture of the Swarm OS Control Plane, the proposed ontologies and templates were not complete. The definition of interfaces and APIs shall be completed and documented. Other research efforts on the Swarm shall be integrated to this work, as they are developed, such as the Global Data Plane (KUBIATOWICZ, PALMER (2014)), micro-payments (CATALANO, RUFFO, 2004) and Tessellation OS (COLMENARES et al., 2013). This way, it is possible to realize the contributions of this thesis are valuable but this is an open research topic with many contributions to be made.

4 THE SWARM OS CONTROL PLANE PROOF OF CONCEPT

This chapter presents the description of the proof of concept and a use case around the Swarm OS proposal. It is divided in four parts: the first part presents the Control Plane that was developed based in its description in the previous chapter; the second part presents the proof of concept implementation of the Smart Jukebox Swarmlet; the third part describes the results porting the framework and the Smart Jukebox to embedded platforms; finally, the forth part presents the proof of concept of a semantic Broker using the WSMX.

4.1 FRAMEWORK

A proof of concept was developed, in order to have a partial implementation of the Swarm OS Control Plane. As the Broker is the access point to the control plane services, an important part of the framework proof of concept is a Broker implementation. The features implemented in this proof-of-concept Broker were: Register, Announce, Locate and Negotiate.

In this implementation, the Register feature is implemented with the Broker offering a web service interface to receive the Service Description Files from each local service that are provided by the platform. Hence, when each Service Provider starts running, it shall register with the Local Broker. This registration consists of an HTTP Post message at a previously established location (local host and pre-defined port); in the proof of concept, the location can be modified using a configuration file.

The Announce feature is implemented with the Local Broker periodically sending an UDP multicast message containing the Service List file. All Local Brokers in this network will listen to these messages and will identify services that are being required by Service Consumers.

The Locate feature allow that Service Consumers to register service dependences and contract constraints at the Local Broker. Service dependences are the description of services that are required by the Service Consumer and contract constraints are the conditions that shall be complied with to establish the service providing. The Locate can be conducted at the same interface as the Register, and the file content can be merged.

After the Broker identifies the Service Provider candidates to a given request, it will rank them accordingly to the contract constraints, including QoS. The Broker will propose a

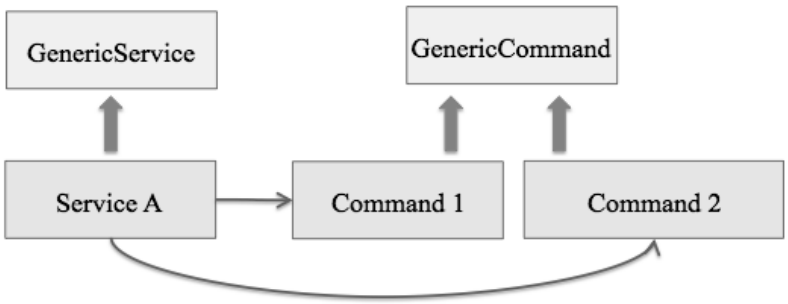
contract to the Service Provider and to the Service Consumer that may send counter proposals.

In addition to the Broker, support classes – that can be offered as APIs, to facilitate services, agents, and Swarmlets implementation - were deployed. The support classes consider that each element has configuration files. The configuration files are the Service Description, Contract Constraints and Other Configurations. The implementation offers classes to interpret these files. It is possible to load commands to classes using the configuration files by Java Reflection – the commands are added to the classes at runtime.

To facilitate the development in the Swarm OS, the Generic Service and Generic Command classes were created. Generic Service implements common functionalities, such as loading and parsing configuration files, Local Broker communication (HTTP client), log messages generation, commands generation and HTTP client infrastructure provision. On the other hand, Generic Command implements the common functionalities to Commands, that includes the infrastructure to receive HTTP commands (server-side) and the Java reflection processing; Services inherit Generic Service and contain commands. Commands inherit Generic Command.

The Broker, as well as the other Control Plane entities (Broker, Services and Personas/Agents), implements an HTTP server and an HTTP client. Each Broker receives a file from the local services (services running in the same device), containing its respective *service description*, *contract constraints* and *service dependences*. The information is registered in a local directory.

Figure 27 – Generic Command and Generic Service classes relationship.



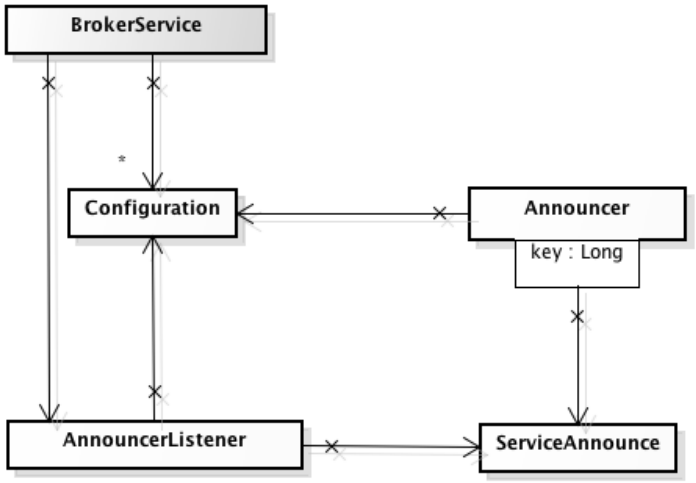
Source: Author

When each device connects to a network, the device local Broker sends an *advertisement* in UDP multicast with its *service description list* containing the hash of the services registered with this Broker. The Broker makes available each Service Description for a HTTP GET from any other entity. The Broker may record the Advertisements received,

caching them and/or the other Brokers. In the current implementation, the Broker continuously multicast the advertisements and there is no caching of the received advertisements.

As shown in Figure 28, the Brokers generate and listen to *advertisements*. The current implementation makes available an advertiser (Announcer) that sends service advertisements (ServiceAnnounce) and an advertiser listener (AnnouncerListener) receives them. The Broker Service uses all of them.

Figure 28 – Broker advertisement class structure.

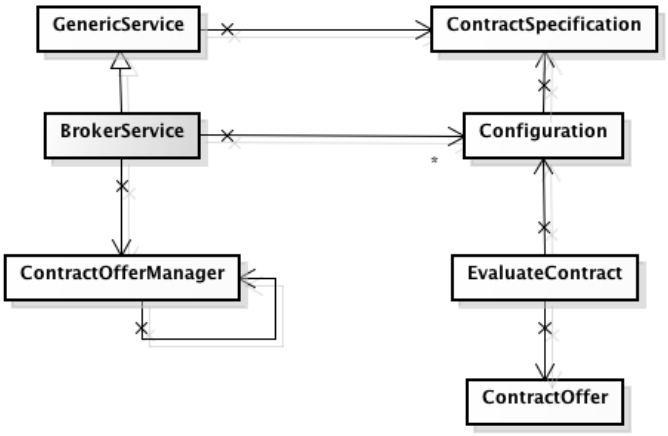


Source: Author

Once the Broker reads the service dependences, it will look for the registered services. In our case, it will filter the advertisements looking for the required service. It will look for services that match the descriptions sent. Afterwards, the contract constraints of the suitable services will be processed seeking the best option for both Service Providers and Requesters. Once the best match is identified, the Broker sends a Contract Proposal to both Service Provider and Requester that, in the current implementation, will answer with an acceptance message (OK) or with a rejection message (NOK). The implementation model is shown in Figure 29.

The Broker inherits the Generic Service as well, being implemented as a special case of the Generic Service. This occurs because the Generic Service offers many features that will be used by the Broker, such as the HTTP client and server and the configuration file parsing.

Figure 29 – Broker negotiation classes structure



Source: Author

Figure 30 – Service Description File Example

facerecognition.json

```
{
        "name":"facerecognition",
        "port":13000,
        "address":"127.0.0.1",

        "contractSpecification": {
                "features": [ ]
        },

        "dependences": [
                {

                "filter": {
                        "identification":0,
                        "address":"",
                        "name":"pir",
                        "processor": "br.org.lsi.swarmos.genericservice.events.EventDependenceProcessor"
                        },

                "offer": null

                }

        ],

        "commands" : [ ],

        "properties" : [
                {"name": "addressOfFR", "value": "localhost" },
                {"name": "portOfFR", "value": "9998" },
                {"name": "directoryOfDictionary", "value": "/home/max/Src/lsi/facerecognition/test/build/tmp_dic" },
                {"name": "temporaryDictionary", "value": "/tmp" },
                {"name": "timeInSecondsToSendNameAgain", "value": "5" },
                {"name" : "brokerAddress", "value": "localhost" },
                {"name" : "brokerPort", "value": 12000 }
        ],

        "logPort" : 5003

}
```

Source: Author

The Service Description file is in JSON format and can be divided into four sections (Figure 30). The first section presents information about the service: name, port, and IP address. This information is important to allow that an entity that receives this description in an announcement to be able to access this service. The second section contains the contract constraints (contractSpecification), which in this case is not declared. The next section, dependences, presents the name of the service this service is requesting; in this case, the PIR is being requested. The following session allows for add additional commands to the service

using reflection; no commands are added to the example. Finally, the last section, properties, contains internal configurations relevant to the service.

The implementation is in pure Java, using some additional libraries for parsing configuration files and to implement the HTTP protocol. We are using JSON file format as the configuration files. Maven was used for building automation.

4.2 APPLYING THE SWARM OS TO A USE CASE: SMART JUKEBOX

As previously described in section 1.5, the architecture proof of concept will be developed considering the Smart Jukebox swarmlet scenario. The Smart Jukebox is an automatic music composer that identifies a user that approaches an area of interest, and based on this user's music preferences, it composes a new song dedicated to the user.

The Smart Jukebox use case has a total of 3 application services, the swarmlet and personal agents implemented: *Composer and Player*, *Proximity and Identification (Face Recognition)* and *Persona Registration*.

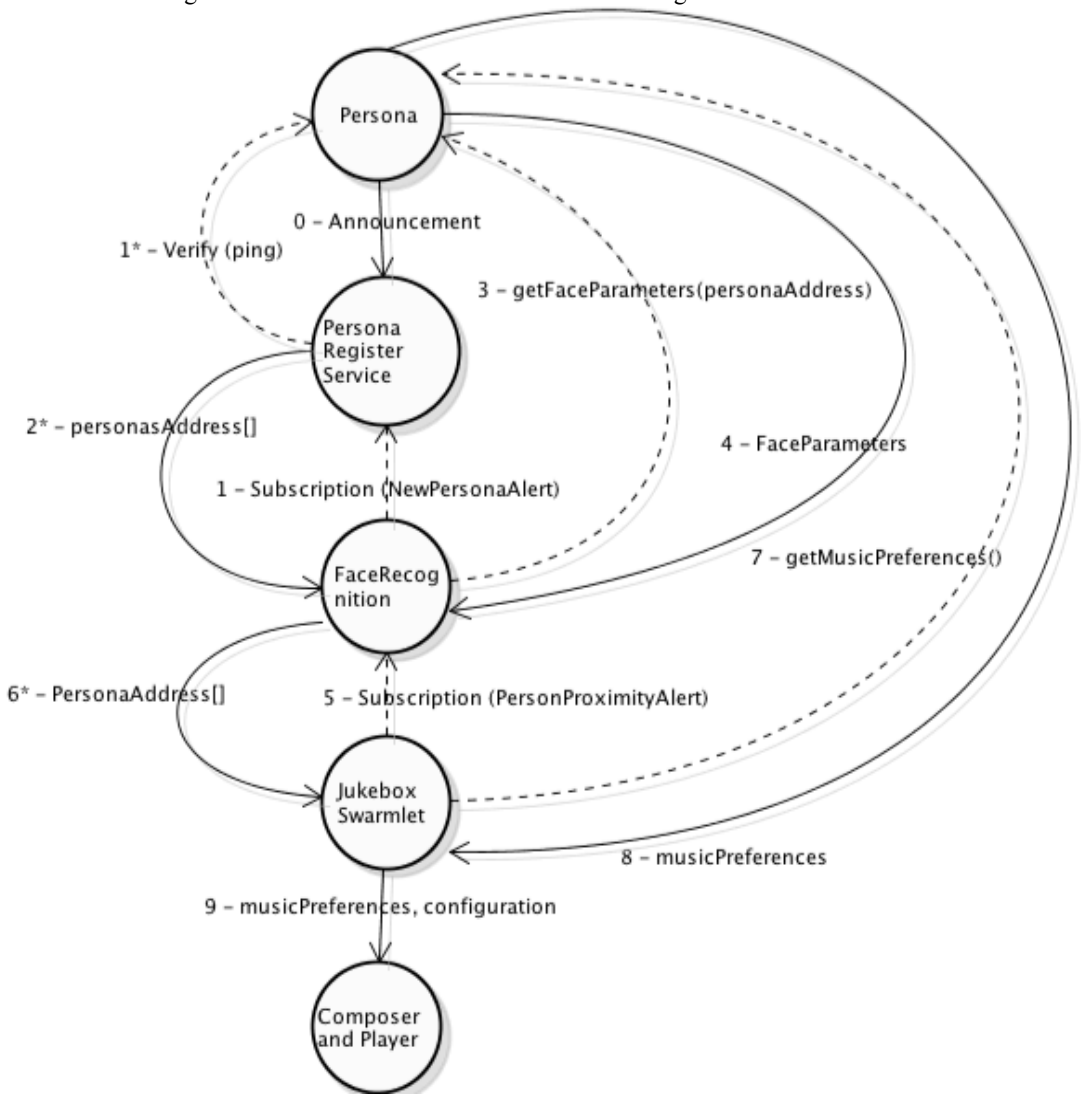
Figure 31 presents the Smart Jukebox dataflow diagram. The *Composer and Player Service* is responsible to compose a new music inspired with a music style given as input. The *Proximity and Identification Service* account for generating events when a person enters in an area of interest, retrieving the person's identification – using a camera and a Face Recognition process. The *Persona Registration Service* will register the agents available into a Local Area Network, whose coverage contains the area of interest. The Persona Registration Service is necessary to narrow down the library that will be used by the Face Recognition algorithm. The Smart Jukebox dataflow operates as follows:

- New devices connect to the local network and advertise themselves and their services. Thus, a Personal Agent that connects to the local network broadcasts its presence.
    - Specifically, a person installs a Personal Agent in his/her smartphone. The personal agent advertises itself to the local broker each time it starts its execution (when the smartphone restarts – it is always running). When the smartphone connects to a local area network, the Broker advertises its device services, including the Personal Agent advertisement.
- The *Persona Registration Service* identifies *Personal Agents* advertisements (in the local network) by a service category tag and registers the Personal Agents

identified. The *Persona Registration Service* continuously verifies if the Personal Agent is still in the network maintaining the Personas list updated.

- The *Proximity and Identification Service* (i.e. *Face Recognition*) establishes a contract with the *Persona Registration Service* in order to receive notifications with the address of new *Personal Agents* in the network.
- The *Proximity and Identification Service* (i.e. *Face Recognition*) requests information with a GET at the advertised Personal Agent's service address - sending its own Principal embedded in the query.
- The *Personal Agent* verifies its configurations (principal) and following its polices, will share the Public Part of the Persona back to the *Proximity and Identification Service* (i.e. *Face Recognition*) (the police may require direct authorization from the Agent).
- The *Proximity and Identification Service* (i.e. *Face Recognition*) will consult in the Persona (Personal Agent data) where to retrieve the face recognition parameters, and will retrieve it.
- The *Proximity and Identification Service* (i.e. *Face Recognition*) scans the area of interest and when it identifies a registered person, it sends a notification to the agents subscribed (the Swarmlet, in this case).
- The Swarmlet looks for suitable Proximity and Identification, Composer and Player services.
- The Swarmlet negotiates with the Proximity and Identification, Composer and Player services for long-term provision,
- The *Swarmlet* subscribes with the *Proximity and Identification Service* to receive a person's *Personal Agent* address when he or she is in an area of interest.
- The *Swarmlet* retrieves the person's music preferences from their *Personal Agents* – this will depend on the *Personal Agent* policies to allow this access as well.
- The *Swarmlet* sends new seeds to the *Composer and Player Service*, considering the music preferences of people in a specific area.

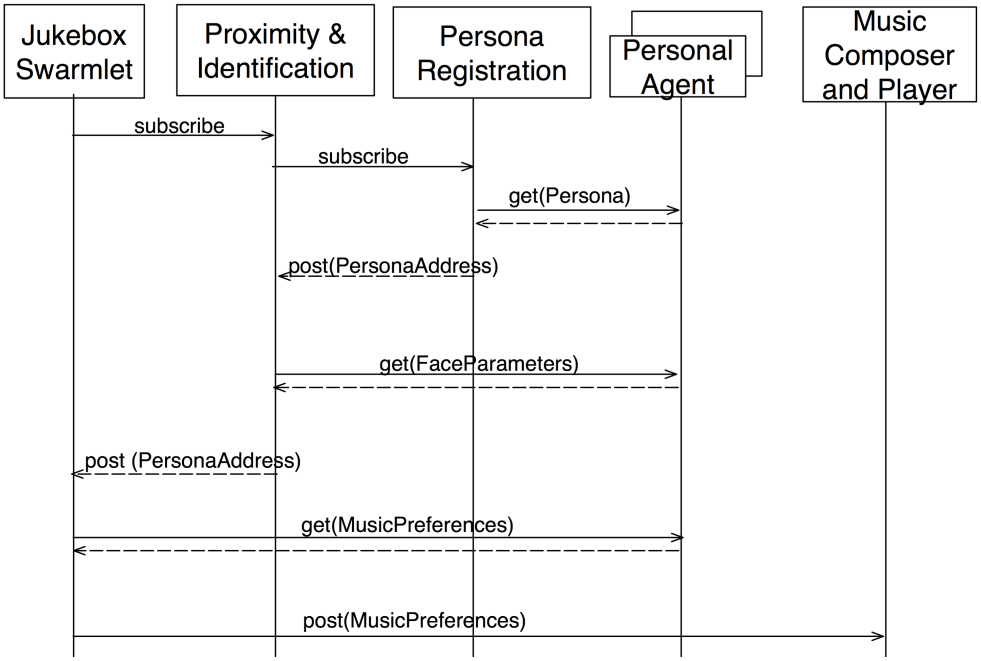
Figure 31 - Smart Jukebox Services Dataflow Diagram



Source: Author

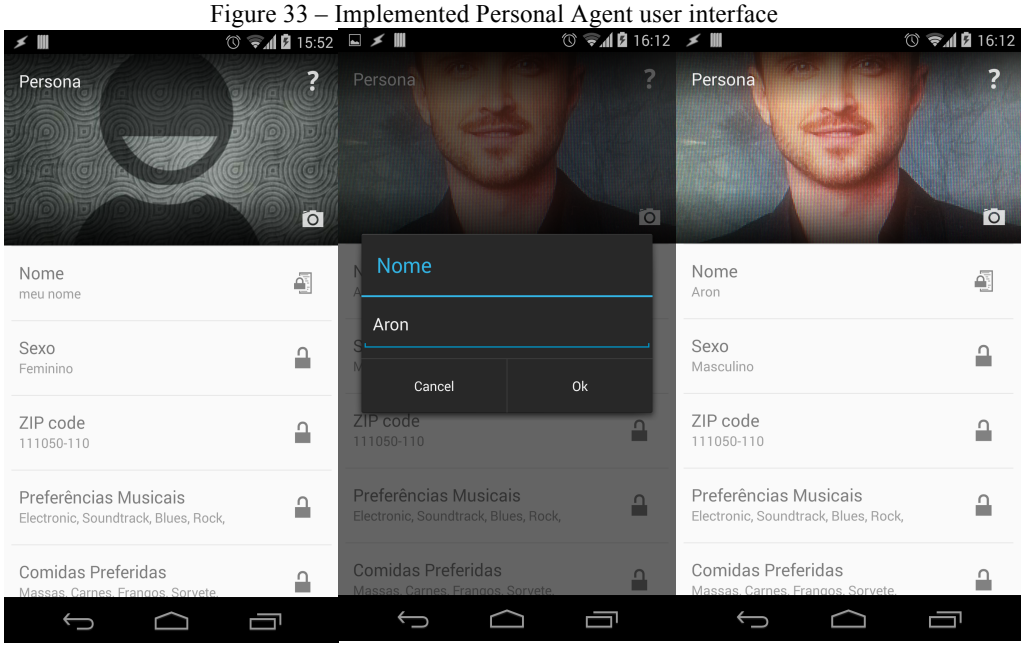
Figure 32 shows the Smart Jukebox Sequence Diagram, the *Jukebox Swarmlet* subscribes to the *Proximity and Identification Service* to receive the *Personal Agent* identifications of the people that are in the area of interest. The *Proximity and Identification Service* subscribes to the *Persona Registration Service*, receiving notifications when a new *Personal Agent* connects to the network. When notified with new Personal Agent connections, the *Proximity and Identification Service* will retrieve the face recognition parameters from the Personal Agent and update its database for face recognition. When the *Jukebox Swarmlet* receives a notification of a person in the area of interest, it will access the *Personal Agent* to retrieve his/her music preferences and then will set-up the composer with this information.

Figure 32 – Smart Jukebox Sequence Diagram



Source: Author

Figure 33 – Implemented Personal Agent user interface



Source: Author

The Personal Agent was developed as an Android application. This application has a user interface that allows user to input personal information and preferences. Figure 33 presents some of the user interface screen shoots. The application allows the user to insert personal information (name, sex and zip code), to take pictures of his/her own face in order to

extract face recognition parameters and other preferences (music style, environment temperature and food preferences and interests). The mature implementation should extract user preferences from the user's day-to-day activities monitoring. Another desirable feature would be to have a distributed persona, the Personal Agent would be able to run in multiple devices (and in the cloud), synchronize status and information from multiple instances. Current implementation is limited to this Android application with direct user configuration.

The Swarmlets, services and personal agents are distributed and may be running all in the same device as well as in different ones. One important thing is to have a local Broker running in each device. As the development was carried out in Java, it is easy to run the Smart Jukebox example in any device. The test scenario considered the Personal Agents running on Android cellphones and the other elements running in personal computers. Figure 34 presents the distribution of application services on devices in the system to the test scenario, however it can be arbitrarily changed without impacting in the system functionality. It is important to have a Broker running in each device, in order to register and advertise services, establish contracts and locate services.

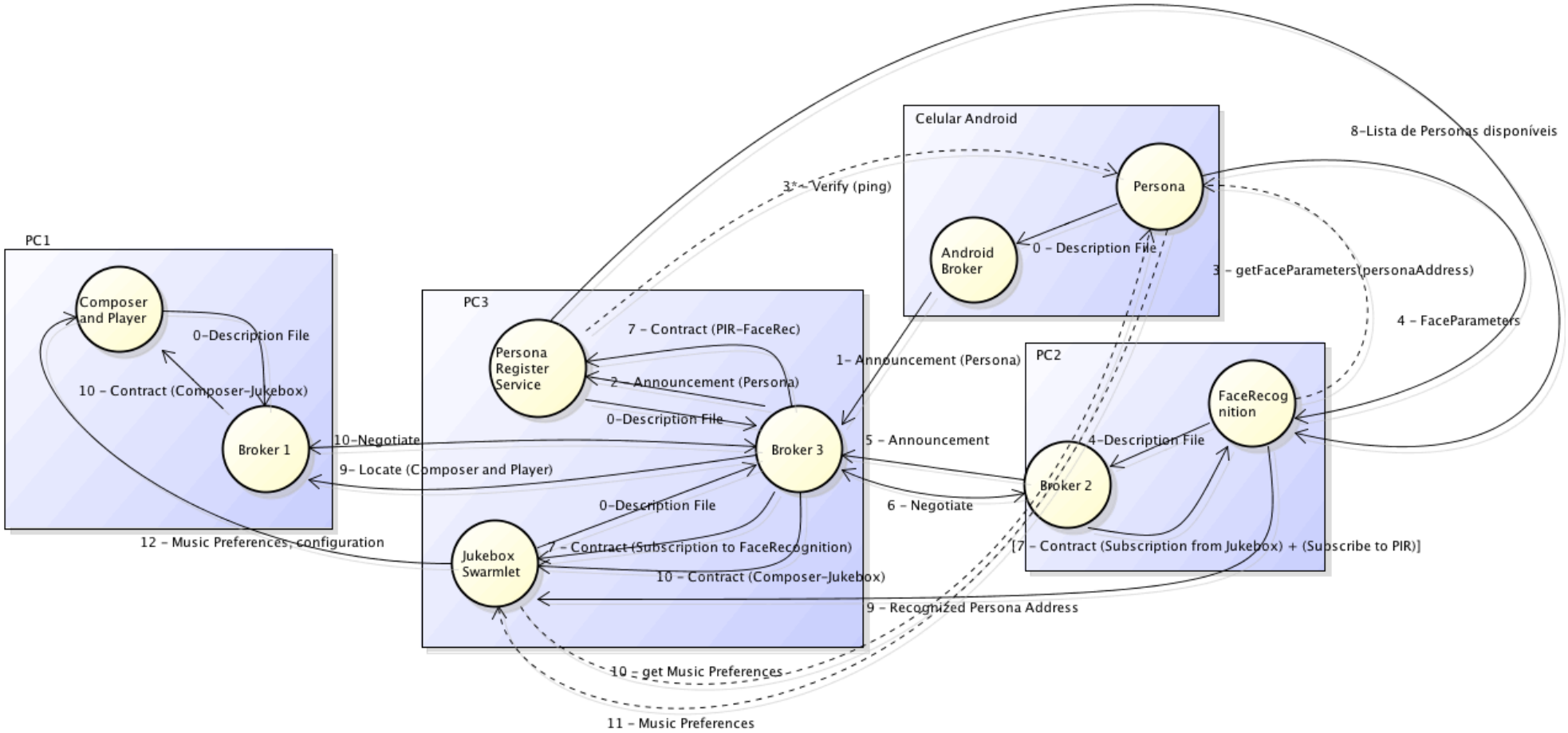
Figure 35 presents a simplified class diagram representing the Services and Commands implemented for the Smart Jukebox example. Taking advantage of the features implemented for the framework, all the Services, Persona and Swarmlet inherit the Generic Service and the Generic Command. A subscribing service was created extending the generic service and was offered by the Persona Registration and the Face Recognition services.

As verified in the diagram, the entities that inherits from the Generic Service are: Swarmlet, Personal Agent (PersonaService), Broker (BrokerService), Composer and Player Service (ComposerService), Proximity and Identification Service (FaceRecognitionService) and the Persona Registration Service (PIRService). The Generic Service provides a Service Server that will put the HTTP server available. As any service provision is related to a contract, a Contract Specification class is related to the Generic Service.

A Generic Service may have Generic Commands. In the class diagram presented, there are commands from the Broker (Announce, Register, Offer Contract and Negotiate Contract), from the Proximity and Identification Service (Receive Persona Event), from the Composer and Player Service (Play), from the Personal Agent (Get Profile) and from the Generic Subscription Service (Receive events and notifications).

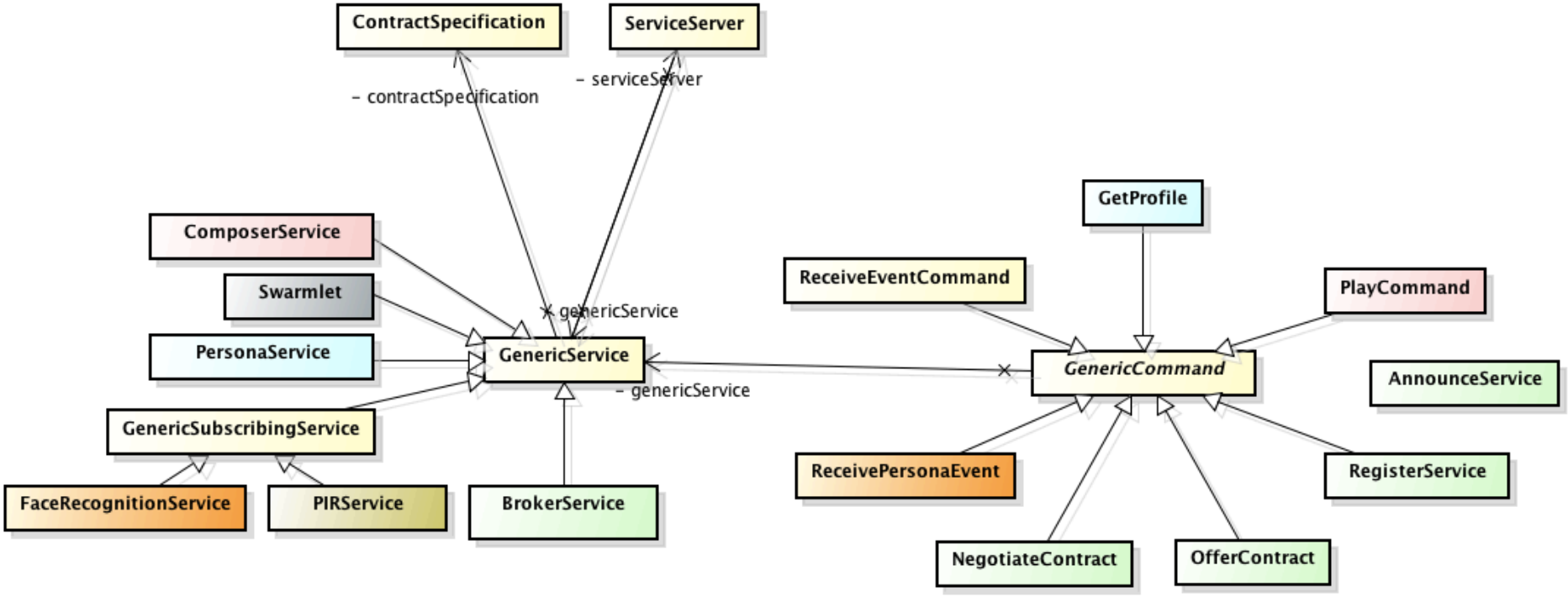
Tests were run and have shown the platform allows for distributed computation. Local Brokers can find each other and communicate. Multiple Services and Personal Agents can be added distributed in arbitrary devices maintaining the Swarmlet working.

Figure 34 - Smart Jukebox Detailed Data Flow Diagram



Source: Author

Figure 35 – Smart Jukebox Class Diagram – Services and Commands



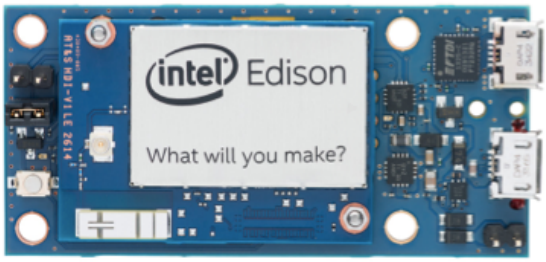
Source: Author

4.3 CONTROL PLANE RUNNING ON EMBEDDED DEVICES

In order to run initial tests of the Control Plane portability to embedded platforms, the Smart Jukebox example was tested in different devices. The platforms selection for the test was done considering its availability in the laboratory. The tests were conducted with an Intel® Edison development platform, an Intel Galileo Arduino board and with a Raspberry Pi. All the platforms were able to run the Local Broker and the PIR Service as is, without any optimizations or adaptations.

The Intel® Edison development platform (INTEL, 2014) was designed to rapidly prototype and produce IoT and wearable products. It runs a 22 nm Intel® SoC that includes a dual-core, dual-threaded Intel® AtomTM CPU at 500 MHz and a 32-bit Intel® QuarkTM microcontroller at 100 MHz, 1 GB RAM LPDDR3 POP memory (2 channel 32bits @ 800MT/sec) and 4 GB FLASH eMMC (v4.51 spec), with support to Wi-Fi 802.11 a/b/g/n onboard antenna or external antenna and Bluetooth 4.0. The available operating system is a Yocto Linux v1.6 (a complete embedded Linux development environment with tools, metadata, and documentation), supporting the Arduino IDE, the Eclipse (C, C++, and Python) and Intel XDK (Node.JS and HTML5).

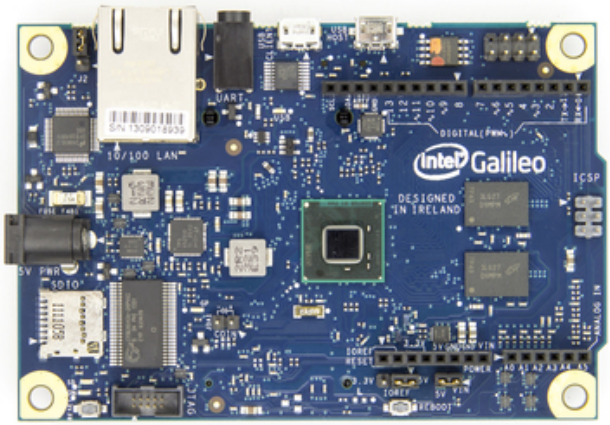
Figure 36 – Intel Edison development platform



Source: Intel (2014)

Galileo is a microcontroller board based on the Intel® Quark SoC X1000 Application Processor, a 32-bit Intel Pentium-class system on a chip (ARDUINO, 2014). It's the first board based on Intel® architecture designed to be hardware and software pin-compatible with Arduino shields designed for the Uno R3. The Galileo board is also software compatible with the Arduino Software Development Environment (IDE). A Linux Operating System, specifically the Debian Wheezy, was installed using a Secure Drive Card (SD Card). The SD card is necessary to expand the available flash memory.

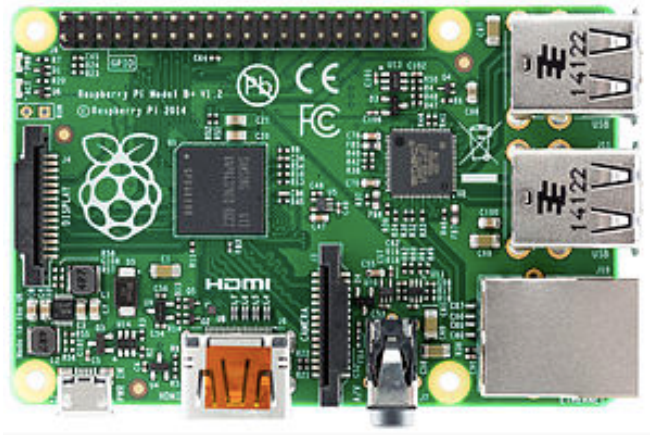
Figure 37 – Intel Galileo Arduino development platform



Source: Arduino (2014)

As presented on Raspberry Pi Foundation (2014), the Raspberry Pi is a low cost, credit card sized computer that plugs into a computer monitor or TV, and uses a standard keyboard and mouse. It was developed with the intention of teaching basic computer science in schools. The Raspberry Pi is based on the Broadcom BCM2835 System on a Chip, which includes an ARM1176JZF-S 700MHz processor, VideoCore IV GPU, and 512 MB RAM (in the Model B, that is the one used for testing). The system has SD Card sockets for boot media and persistent storage. The Operating System used was the Raspbian, the Debian Wheezy Linux release. Figure 38 presents a picture of the platform.

Figure 38 – Raspberry Pi development platform



Source: Raspberry Pi Foundation (2014)

In all platforms the tests were conducted using the Linux distribution available with the platform, and then installing an official Java Virtual Machine (JVM), from Oracle. As all platforms support TCP/IP connection, no gateways were necessary. The network connection

with Edison was done with Wi-Fi and with the Galileo board and with the Raspberry Pi, the network connection was done with the Ethernet cable. In all platforms the JVM installed was the Oracle Java SE 8u25.

The Broker, PIR service and Swarmlet implementations were compiled building Java byte codes for each one. The Java byte codes run on the platform using its JVM without any problems. Naturally, the JSON configuration files had to be edited to each platform, as explained in section 4.1.

The following scenarios were successfully tested:

- **Scenario 1**: 1 Edison, 1 personal computer and 1 Android tablet. The Jukebox example was distributed among these devices running the following softwares in each one:
    - **Edison**: Broker, Persona registration service and Swarmlet;
    - **Personal Computer:** Broker, Face recognition service and Composer service;
    - **Android tablet**: Broker and Persona.

- **Scenario 2:** 1 Galileo, 1 personal computer and 1 Android tablet. The Jukebox example was distributed among these devices running the following softwares in each one:
    - **Galileo**: Broker, Persona registration service and Swarmlet;
    - **Personal Computer:** Broker, Face recognition service and Composer service;
    - **Android tablet**: Broker and Persona.

- **Scenario 3:** 1 Edison, 1 Galileo, 1 personal computer and 1 Android tablet. The Jukebox example was distributed among these devices running the following softwares in each one:
    - **Edison**: Broker and Swarmlet;
    - **Galileo**: Broker and Persona registration service;
    - **Personal Computer:** Broker, Face recognition service and Composer service;
    - **Android tablet**: Broker and Persona.

- **Scenario 4**: 1 Raspberry Pi, 1 personal computer and 1 Android tablet. The Jukebox example was distributed among these devices running the following softwares in each one:
    - **Raspberry Pi**: Broker, Persona registration service and Swarmlet;
    - **Personal Computer:** Broker, Face recognition service and Composer service;
    - **Android tablet**: Broker and Persona.

As the focus of the work is to develop a framework for heterogeneous devices communication, it was considered relevant to run the Smart Jukebox example with other computing platforms than Personal Computers. Currently, many embedded development platforms are commercially available. These platforms aim to fulfill the Internet of Things demands. Figure 39 and Table 4 present some of the currently most popular platforms - including the ones used in the test: Raspberry Pi B, Intel Galileo and Intel Edison - comparing them regarding embedded RAM and the CPU available. This comparison shows that the selected platforms are significant to the comparison and that the Control Plane could potentially be port to the other platforms as well.

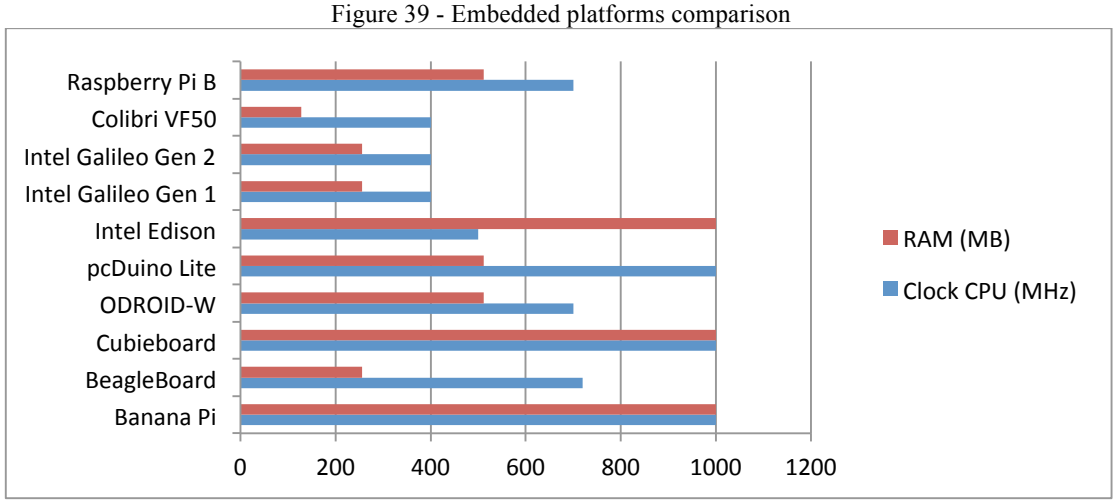
Table 4 – Embedded platforms comparation

PLATAFORM	*CPU*	*Clock CPU (MHz)*	*RAM (MB)*
Banana Pi	ARM Cortex-A7	1000	1000
BeagleBoard	ARM Cortex-A8	720	256
Cubieboard	ARM Cortex-A8	1000	1000
ODROID-W	ARM11	700	512
pcDuino Lite	ARM Cortex-A8	1000	512
Intel Edison	Intel® AtomTM	500	1000
Intel Galileo Gen 1	32bit Intel® Pentium	400	256
Intel Galileo Gen 2	32bit Intel® Pentium	400	256
Colibri VF50	ARM Cortex™-A5	400	128
Raspberry Pi B	ARM1176JZF-S core	700	512

Source: Author

Despite these tests has shown the viability to run the Control Plane in embedded platforms, there are more constrained devices that will not be able to execute the Control Plane, requiring the use of CoAP protocol and an optimized implementation. On the other hand, the embedded platforms are continuously more capable. Products that previously were considered to communicate using Zigbee or Bluetooth are currently using Wi-Fi (such as: lamps, music systems and refrigerators).

Figure 39 - Embedded platforms comparison



Source: Author

4.4 MEDIATION: SEMANTIC BROKER PROOF OF CONCEPT

Section 2.3 presented the main ontology standards for semantic web services. In this section we will discuss the practical application of these frameworks to develop a Broker with mediation service support. The Smart Jukebox example was used as a practical use case, but with a narrower scope: the Composer service providing and consuming.

The evaluated related projects offered great insights in order to achieve a real world implementation of a semantic-aided Internet of Things, however, many of the project homepages are not available in current days. This way, the project selected to develop a proof of concept of a Broker supporting semantic was the WSMF, due to:

(1) WSML language, it is based on F-Logic and Description Logic, which are simpler than XML, so it is possible to easily write and modify ontologies;

(2) The design tool, WSMT, is mature and sufficient for web services annotation in WSML language. Since it is based on Eclipse, it presents useful editing features.

(3) The Web Service Modeling eXecution environment, WSMX, integrates the main aspects of the Semantic Web Services approach, such as discovering, reasoning, invocation and composition. WSMX is an execution environment based on principles of Service Oriented Architecture (SOA), developed aiming business application integration where enhanced web services are integrated for various business applications. WSMX features are offered as services, accessed by SOAP, in a centralized fashion.

This way, the WSMX provides a framework suitable to develop a proof of concept of a Broker supporting Mediation service.

In the proposed Control Plane architecture, service consumers send to the broker an abstract description of the service that it is requesting (including functional and non functional features, as well as billing conditions). Services requesters will share its own description with the local broker, a concrete description (including functional, non functional, billing conditions and binding information). The local broker will process service requests, communicating between other Brokers if necessary, helping to find the best match and return a contract, i.e. a service level agreement, to the service consumer that can start accessing the service provider directly. Considering the semantic swarm, the descriptions would be a set of files with the semantic descriptions of services and the domain ontology description (that can be requested when necessary).

Figure 40 - Ontology describing the domain of the simplified Composer Service.

```
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-rule"
namespace { _"http://citi.usp.br/ontologies#"}

ontology ComposerOntology

concept Genre
    hasName ofType _string

concept Bitrate
    hasKbps ofType _integer

concept Song
    hasTitle    ofType _string
    hasGenre    ofType _string
    hasDuration ofType _integer
    hasBitrate  ofType Bitrate
    hasNumberOfChannels ofType _integer

concept Symphony subConceptOf Song

axiom SymphonyDefinition
    definedBy
      ?x memberOf Symphony equivalent ?x memberOf Song.

concept SoundTrack subConceptOf Song

concept Lullaby subConceptOf Song
```

Source: Author.

The proof of concept will vary a little bit, since we are adopting WSMX. We are focusing in the Mediation process and we will use a centralized model. There is just one "Semantic Broker" that will receive all concrete and abstract services – i.e. service descriptions that are being provided, as well as the requests for services. This example does not have contracts, but support quality-of-service description and ranking among services due

to their QoS. Service consumer does not access the service provider directly; instead the communication is through the Broker.

The proof of concept was designed considering the Smart Jukebox example. There are two Composer Services being provided and registered at the WSMX (representing our Broker). Each of the services uses a different vocabulary for the Service Description and an ontology will map the terms used, making both services equivalent. A Service Consumer requests a Composer service. The WSMX expands the service description searching and finds both Composers. WSMX uses the most suitable service considering Quality of Services parameters.

The Domain Ontology specifies the vocabulary that can be used in the web services description. In Figure 40 a very simple domain for the Composer example is presented. It presents the Song and Symphony concepts that are described as equivalent. Thus, when a Service Consumer looks for a service that returns a Song, the Broker will consider any service that returns a Song or a Symphony as valid.

The Service Description contains the description of a concrete web service, i.e. a web service that is being offered. As the WSMF supports SOAP services only, the proof-of-concept implementation used Apache CXF (http://cxf.apache.org/) to implement two different SOAP web service interfaces for the Composer. Then, a semantic annotation was created for each service; an example is presented in Figure 41. It contains its post-conditions, input, output and binding information. The quality-of-service conditions are defined in another file, as presented in Figure 42.

Figure 41 - Part of the Composer service description

```
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-rule"
namespace { _"http://citi.usp.br/services#"}

webService SongComposer
importsOntology {composer#ComposerOntology}

capability SongComposerCapability
sharedVariables ?song
postcondition composedSong
     definedBy
          ?song memberOf composer#Song.

interface WSSongComposerInterface
choreography WSSongComposerInterface
    importsOntology {composer#ComposerOntology}
    in composer#Genre withGrounding {
        _"http://localhost:8093?wsdl#wsdl.interfaceMessageReference(
            Composer/composeHymn/genre)"}
    out composer#Song
```

Source: Author

Figure 42 - Part of the quality-of-service definitions file related to a Composer Service

```
ontology ComposerBaseQoS
importsOntology { qos#QoSUpperOntology }

concept BitrateUnit subConceptOf qos#MeasurementUnit
concept Bitrate subConceptOf {qos#Quality, qos#HigherBetter}
    qos#unit impliesType BitrateUnit
instance BitrateRange memberOf qos#QualityRange
    qos#item hasValue Bitrate
    qos#min hasValue 0
instance providedBitrate memberOf {compQoS#Bitrate, qos#ServiceSpec}
    qos#value hasValue 128
    qos#unit hasValue compQoS#BitrateUnit

concept NumberOfChannelsUnit subConceptOf qos#MeasurementUnit
concept NumberOfChannels subConceptOf {qos#Quality, qos#HigherBetter}
    qos#unit impliesType NumberOfChannelsUnit
instance NumberOfChannelsRange memberOf qos#QualityRange
    qos#item hasValue NumberOfChannels
    qos#min hasValue 0
instance providedNumberOfChannels memberOf {compQoS#NumberOfChannels,
                                            qos#ServiceSpec}
    qos#value hasValue 5
    qos#unit hasValue compQoS#NumberOfChannelsUnit
```

Source: Author

The Service Consumer will present an abstract service description, i.e. the description of a service that is being requested (without binding definitions), that is named Goal in WSMO. The Goal describes the attributes of the required web service, as presented in Figure 43. Important aspects of a goal description are:

(1) In **postcondition** parameter, a service that returns a Song element as output is required.

(2) Quality of service parameters are defined, this case the music streaming bitrate that is generated by the composer (**requiredBitrate**) and the number of channels of the music (**requiredNumberOfChannels**). The Broker shall find and order a list of suitable services using the parameters defined. Suitable services are selected using the minimum value as filter; on the other hand, the ordered list is built using the weight (**hasWeight**) parameter, associated with the mark-up that informs that the bitrate and number of channels should be as high as possible (**HigherBetter** markup).

(3) An initial instance of the concept **Genre** is given as input which is used to find a compatible service that takes it as a parameter and returns a **Song**.

WSMX offers its functionality in the form of web services, thus, in order to achieve a goal, we invoke the service **achieveGoal** passing the goal in WSML as a parameter. In this example, WSMX executes the Locate process looking into its own registry, in three steps:

- **Service Discovery**: in the first step, WSMX will use the Goal to find services that receive a Genre and return a Song. It will discover services that return

Symphony too since it as stated previously that both concepts are semantically equivalent.

- **Quality-of-service evaluation**: once the candidate services are discovered, the QoS restrictions given in the goal are applied: a multiple constraint optimization problem is solved. It discards the candidate services with bitrate and number of channels lower than the minimum defined, and will prioritize the others accordingly to the QoS parameters that are offered, looking for the maximum values with the higher weight.
- **Web service invocation**: After the candidates are chosen and ranked we have the description of the best-suited web service. Then a SOAP envelope is generated in order to invoke the actual web service, using the grounding information: URL, port, message and parameter. Finally, the web service composes a Jazz song and returns a SOAP message with a Symphony object ready to play.

Figure 43 – Goal description file for a Composer web service

```
ontology ComposerBaseQoS
importsOntology { qos#QoSUpperOntology }

concept BitrateUnit subConceptOf qos#MeasurementUnit
concept Bitrate subConceptOf {qos#Quality, qos#HigherBetter}
    qos#unit impliesType BitrateUnit
instance BitrateRange memberOf qos#QualityRange
    qos#item hasValue Bitrate
    qos#min hasValue 0
instance providedBitrate memberOf {compQoS#Bitrate, qos#ServiceSpec}
    qos#value hasValue 128
    qos#unit hasValue compQoS#BitrateUnit

concept NumberOfChannelsUnit subConceptOf qos#MeasurementUnit
concept NumberOfChannels subConceptOf {qos#Quality, qos#HigherBetter}
    qos#unit impliesType NumberOfChannelsUnit
instance NumberOfChannelsRange memberOf qos#QualityRange
    qos#item hasValue NumberOfChannels
    qos#min hasValue 0
instance providedNumberOfChannels memberOf {compQoS#NumberOfChannels,
                                            qos#ServiceSpec}
    qos#value hasValue 5
    qos#unit hasValue compQoS#NumberOfChannelsUnit
```

Source: Author

Using WSMX as a Semantic Broker proof of concept, and applying it to the Composer Service registration, locates and selection from the Smart Jukebox example previously presented; we could proof the concept of:

1) Registry: WSMX maintains a centralized registry of the available services.
2) Locate Service: WSMX allow the search of suitable services registered inside.

3) Service description expansion: the use of different terms in two different services description files and yet being capable to locate both services as suitable to a service consumer request, processing an ontology description.
4) Best service match to the service consumer request considering quality-of-service: rank the suitable services accordingly to the optimization of the QoS parameters (weight of each parameter and looking for the higher values available).

Using WSMX as a Semantic Broker proof of concept, and applying it to the Composer Service registration, locate and selection from the Smart Jukebox example previously presented; it was possible to realize that WSMX, which is a good reference and partially meets the needs of the semantic-enabled Swarm Broker. Functionally, it presents features required for the semantic swarm: registration, locate services with optimization parameters, ontologies expansion and pre-defined composition of services. On the other hand it does not support others, such as: dynamic composition of services, contracts and micropayments, negotiation, authentication, announcements and optimization. In addition, WSMX is developed in a centralized fashion, but the Swarm requires a distributed implementation, guaranteeing a minimum broker, implemented in every node, working jointly with more complex device to provide more complex functionalities. Other important point to the system optimization guaranteeing the compatibility with low processing power devices is the use of RESTful binding associated with JSON descriptions, allowing lightweight communications. WSMX does not fully implement the WSMO specification and has some manual steps that should be automated to practical use, as the translators between SOAP and WSML for example.

4.5 DISCUSSION

This chapter presented the proof of concept of the Swarm OS Control Plane Proposal, applying it to the Smart Jukebox use case. In addition, a Semantic Broker Proof of Concept applied to the Smart Jukebox use case was presented as well. It was possible to achieve a flexible and scalable computational framework that opportunistically integrates heterogeneous networked devices so they can synergistically and organically autoconfigure themselves to perform complex tasks.

The proof of concept described in this chapter implemented a Local Broker, as a Java application, capable of supporting minimum Control-Plane services: Register, Announce, Locate and Negotiate. Our proof of concept considered the Smart Jukebox use case, where

services were offered and arbitrarily distributed to run over personal computers, smart phones and embedded platforms. These devices synergistically cooperate with each other offering its resources as web services composing simpler services (in our case: Persona Registration Service, Face Recognition Service and Composer and Player Service) to perform a more complex task (identify a person in an area of interest, identify this person's music preferences, compose and play a music to this person). This cooperation is based on quality-of-services assurance using contracts. In our proof of concept the number of audio channels and audio streaming bitrate capability were used for quality of service negotiation.

Multiple instances of each service may be offered in the proof of concept and the Smart Jukebox swarmlet is able to compose the application selecting services from the set of services available. Each device has a Broker locally running that helps to advertise and seal contracts between services providers and consumers.

The proof of concept exercised the Personal Agent aspect, using a Persona implementation in order to collect face-recognition parameters and music preferences. The Persona implementation was simplified; it was implemented as an Android application with a user interface to configure the person preferences. It is expected to have, in the future, a distributed implementation of the Persona, with a computer learning system extracting users' preferences from their behavior.

The use of semantic computing, in the Swarm OS Control Plane, leverages the consolidation of a scalable and organic system. The mediation service is not part of the micro kernel, it is a non-minimum feature, but as it is relevant to the system, a specific proof of concept was developed. The WSMX was used to test the mediation service viability. In this test, it was possible to test the Swarm OS Control Plane flexibility regarding standards. An example of vocabulary expansion for service description was conducted and successfully implemented to the Composer Service.

Despite the Swarm OS Control Plane was not fully developed, the implementations described in this chapter could present many of the aspects of the proposed architecture running, proving its viability. As next steps, the architecture shall be fully implemented. The more relevant points are: integration of the Mediation Service to the Control-Plane framework, explore the Broker heterarchy arrangements and develop the micro kernel and platform service layers separation in the Control-Plane architecture.

5 CONCLUSION

This work investigated, proposed and developed a computational architecture for organic and heterogeneous networks: the SWARM Operating System Control Plane. The thesis presented an overview of the technologies related to the topic and discussed some existenting middleware solutions for heterogeneous networks. It presented the control plane architecture proposal for dynamic and opportunistic operation. Proofs of concepts were implemented and tests were conducted, showing the viability of the architecture. Future scenarios of the deployed Swarm were explored showing the potential impact of the technology. This work does not exhaust the research topic; instead it opens new research lines. It establishes a framework that shall be expanded and consolidated with other research works.

5.1 SWARM CHARACTERIZATION

This thesis proposes the Swarm OS, a middleware for heterogeneous devices interoperability in computer networks. Terms that are related to this scope are ubiquitous computing, pervasive computing, cyber-physical systems, the Internet of Things and the Swarm. Swarm is the term adopted and the work brought contributions to this term. The Swarm, as proposed, is characterized as: heterogeneous, distributed, autonomous, cooperative, dematerialized, virtualized, dynamics, wide communication scope and granular. The main challenges to implement the Swarm ecosystem are the interoperability and scalability. Another obstacles to the Swarm are the technology complexity due to the multiple fields of knowledge involved, the semantic localization and the unpredicted Swarm requirements.

5.2 SWARM STATE OF THE ART

Middlewares for distributed computation has been discussed and many works have been developed segmented, for specific fields of applications. The transformation power of swarm computing lies in defining a common platform that can blur the field applications and maximize its transformation power, taking advantage of the synergies among these fields.

Considering the Swarm principles, related technologies were analyzed and discussed: service-oriented architecture, RESTful web services, agent-based computing, semantic computing and middlewares for distributed networks. In order to achieve collaboration

between different devices in the Swarm, the Service Oriented Architecture and Web Services are pertinent; they allow connected devices to use features from each other, independently of the implementation, standardizing the interfaces. The entities using and offering services, as well as the entities that will manage Swarm Computing resources may be faced as agents, putting together a multi-agent system. Semantic computing is a powerful technology that we propose in this thesis to be used in order to achieve interoperability and flexibility when using evolving, live standards.

Regarding middleware solutions, we have presented some of the current systems related to the work theme. We could identify distributed processing middlewares that have been used to ubiquitous computing, semantic middlewares proposals and we have explored middlewares developed to local network, as domestic environment. These systems have strong and weak points; and there is still space for a proposition for the Swarm Computing Control Plane, with a broader scope of communication, flexibility to standards and capable of working in heterogeneous nodes.

5.3 SWARM OS CONTROL PLANE ARCHITECTURE

The middleware for the cooperation between devices in the Swarm, we are embedding to the operating system and calling it Swarm OS. The scope of the thesis is in the control plane, responsible for managing Swarm resources – defining the Swarm ecosystem actors, how to describe and use services and resources, how advertise and discover them, transaction processes, content adaptation and cooperation.

This thesis presents an overview of the Swarm operation and defining a terminology. The main terms are: resource, service, agent, service provider, service consumer, service requester, owner, persona, discovery, contract, brokerage, broker and cell. The Swarm is a system with autonomous and heterogeneous devices cooperating through their own resources sharing. Each cooperation relationship is sealed with a contract that establishes a service providing commitment and enforces an agreed quality of service. The contract represents the available resource itself; i.e. it is a share, or a cell, of the device's overall resource. The Swarm is organized in a service-oriented and multi-agents architecture. The agents compose this system as intelligent software components that interoperate providing and consuming services. The service is a resource; it provides access to a capability. A service itself can use other services. Swarmlets will run over this framework, leveraging machine-to-machine cooperation, allowing the distributed composition of a graph of services. Personal agents will

have a key role in the system. Each personal agent represents a person and carries its Persona, composed by policies, preferences and information of an individual. Agents and Personas can represent entities.

The thesis presents a proposal for the Swarm OS Control Plane standard grounding, proposing and discussing the following layers: Encoding, Messaging, Transport and network, Addressing and Security. The work proposes the use of JSON files and RESTful web services, using CoAP and HTTP, semantic annotations, gateways for protocols abstraction, public key infrastructure with digital and attribute certificates.

As the Swarm Control Plane is a service-oriented architecture, Services Providers and Consumers shall find each other and establish a service-level agreement. An essential actor in the Swarm OS is the Broker, which assists (as a set of services) service providers and consumers to seal contracts. The Broker intermediates the access to the Control Plane services. The service consumer and provider will post their description files to the Broker, as they became actives. The Service Provider shall make available to the Broker heterarchy: its own description, its contract constraints and its binding information. On the other hand, the Service Consumer shall inform to the broker abstract service descriptions and the contract constraints for each required service. The Control Plane services provided by the Broker are: registry, contracting, mediation, policy, authentication, optimizing and monitoring, plan development and binding.

Due to the heterogeneity of the Swarm scenario, this work proposal is to build the Swarm OS as a distributed Operating System in four layers architecture: a micro-kernel, platform services, application services and Swarmlets. The Control Plane services discussed in the last session are mainly in the category of *platform services,* which go into a lower layer than the Swarmlets and the application services. These services can run anywhere on the Swarm OS distributed platform (in the device or even in the cloud). The only element that needs to be available at any Swarm device is a *micro kernel,* which supports a minimum part of the Control Plane services, embedding a Local Broker that implements the minimum, mandatory, part of the Control Plane Services.

5.4 SWARM OS CONTROL PLANE PROOF OF CONCEPT

In order to show the architecture feasibility, proof of concepts have been developed. In order to apply the proof of concepts, a demonstration scenario was developed. The proof of concept was divided in four phases: the first one consisted in the partial implementation of the

proposed Control Plane Architecture; secondly, a demonstration scenario was developed running over the Control Plane Architecture proof of concept; next, tests with embedded platforms were conducted, porting the framework and running the demonstration scenario; finally, tests with a semantic platform were executed.

The developed framework is a partial implementation of the architecture; it implemented a Broker capable of Register, Announce, Locate and Negotiate; as well as support classes that are used as APIs to facilitate Services and Agents development.

The demonstration scenario implemented and demonstrated a Smart Jukebox use case. The Smart Jukebox is an automatic music composer that identifies when a person approaches an area of interest (using face recognition), and based on this person's music preferences, it composes a new song dedicated to the user. The Smart Jukebox demonstration exercised the framework and has shown that the implementation generate a scenario with automatic discovery of services and the establishment of service level agreements. Future work includes the implementation of the complete architecture.

As this work aims to be applicable to heterogeneous devices, tests were conducted to show the viability of the Control Plane in embedded platforms. The framework could be executed in three different embedded platforms and the demonstration scenario was executed running on these platforms mixed with personal computers in many configurations.

This work shows that applying semantic computing to the Swarm leverages flexibility and scalability to the system regarding the standardization of services, devices and protocols. On the other hand, semantic web services technologies are available for some years, but it has been evolved slowly. Among other reasons, it happens due to the lack of familiarity and complexity with their languages (SONG, CARDENAS and MASUOKA, 2010). Our model considers the devices would carry its own semantic description, as well as its own ontology and ontology mapping. In order to facilitate the adoption of the technology, an option would be to establish an international forum responsible for ontologies mapping.

An example of a Semantic Broker was developed and applied to a scenario, using WSMX, which is a good reference and partially meets the needs of the semantic-enabled Swarm Broker. Functionally, it partially presents features required for the semantic swarm, but it lacks of: dynamic composition of services, contracts and micropayments, negotiation, authentication, announcements and optimization. In addition, WSMX is developed in a centralized fashion, but the Swarm requires a distributed implementation, guaranteeing a minimum broker, implemented in every node, working jointly with more complex device to provide more complex functionalities. Other important point to the system optimization

guaranteeing the compatibility with low processing power devices is the use of RESTful binding associated with JSON descriptions, allowing lightweight communications. WSMX does not fully implement the WSMO specification and has some manual steps that should be automated to practical use, as the translators between SOAP and WSML for example.

Many works using semantic computing to address pervasive computing deploy frameworks with specific, restricted ontologies. On the other hand, other works developed complete ontologies without connecting them to frameworks and applications. An exception is the Masuoka et al. (2003) work, but the code and ontologies are not available. As next steps, this work propose to integrate these more complete ontologies to the Swarm framework, taking advantage of this structured knowledge leveraging a cross-niche solution. The semantic Swarm may consolidate really large ontologies. As the ontologies size impacts on the Semantic Broker performance and footprint, one important future contribution is strategies development to efficiently organize, compose and process ontologies.

5.5 FINAL REMARKS

In order to have the Swarm OS as a framework for large scale application, it would be necessary to standardize the file formats for service description, list of services, contract constraints document, contracts and Persona; as well as to define the Broker interfaces. A first proposal is presented in this work, but a standard would have to be consolidated. The supporting API should be documented, optimized and have other languages options.

This work contributes to the Swarm development. The proposed Swarm envisions that future computation will be performed on a vastly distributed platform of collaborating components, connecting trillions of storing, processing, sensory and actuating devices worldwide into a single platform abstraction. This enables the true emergence of the cyber-physical systems and has potential to generate a new economy around devices cooperation with micro payments, as well as to completely change the context-sensitive computing that will have access to a complete set of information. That will have a tremendous impact in potentially all economic domains, such as: advanced healthcare, improved energy efficiency, environment-friendly living, mobility management, enhanced security, productivity tools and many others.

REFERENCES

ALLSEEN ALLIANCE. Presents information about the AllJoyn Project. Available at: <https://allseenalliance.org >. Access: Dec. 2014.

ANGELIDES, M. C.; AGIUS, H. **The handbook of MPEG:** applications standards in practice**.** England: John Wiley & Sons Ltd., 2011. 551 p.

ARDAKANI, B. et al. **VSCP specification 1.7.12**. 2012. Available at: <http://ufpr.dl.sourceforge.net/project/m2m/VSCP%20Specification/vscp_spec_1_7_12.pdf>. Access: Jul. 2012.

ARDUINO. Presents an overview about the Intel Galileo. Available at: <http://arduino.cc/en/ArduinoCertified/IntelGalileo>. Access on: Dec. 2014

BAADER, F. **The description logic handbook:** theory, implementation, and applications. Cambridge: Cambridge University Press, 2003. v.1.

BINUGROHO, E. H.; SEO, Y. B.; CHOI, J. W. Home Network Infrastructure Based On Corba Event Channel. In: WORLD CONGRESS THE INTERNATIONAL FEDERATION OF AUTOMATIC CONTROL, 17., 2008, Seoul, Korea. **Proceedings**… p. 2194-2199. Available at: <http://www.nt.ntnu.no/users/skoge/prost/proceedings/ifac2008/data/papers/4028.pdf>. Access: Nov. 2014.

BITAR, E. I.; BELOUADHA, F. Z.; ROUDIES, O. Review of Web services description approaches. In: International Conference on Intelligent Systems: Theories and Applications (SITA), 8., 2013, Rabat. **Proceedings**… p. 1-5. Available at: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6560813&tag=1>. Access: Nov. 2014

BISHOP, M. **Introduction to computer security.** United States: Addison-Wesley Professional, 2004. 784 p.

BOOTH, D. et al. **Web services architecture.** W3C Working Group Note 11. Feb. 2004. Available at: <http://www.w3.org/TR/ws-arch/>. Access: May 2014.

BOOTH, D.; LIU, C. K. **Web Services Description Language (WSDL) version 2.0 Part 0:** Primer**.** W3C Recommendation, Jun. 2007. Available at: <http://www.w3.org/TR/wsdl20-primer>. Access: May 2014.

BRAY, T.; PAOLI, J.; SPERBERG-MCQUEEN, C. M.; MALER, E. **Extensible markup language (XML) 1.0.** 5th ed**.** W3C Recommendation, Nov. 2008. Available at: <http://www.w3.org/TR/xml/>. Access: Mar. 2014.

BRUNI, C. et al. **An innovative approach to the formulation of connection admission control problem**. World Academy of Science, Engineering and Technology, v. 18, 2006. ISSN 1307-6884.

BRZOZOWSKI, M. et al. Inter-MAC, from vision to demonstration: enabling heterogeneous meshed home area networks. In: ITG CONFERENCE ON ELECTRONIC MEDIA TECHNOLOGY (CEMT), 14., 2011, Dortmund, Germany. **Proceedings**… p.1-6.

BURNETT, I. S.; PEREIRA, F.; WALLE, R. V.; KOENEN, R. **The MPEG-21 handbook**. England: John Wiley & Sons Ltd, 2006. 464 p.

BURNSTEIN, M. H. Dynamic invocation of semantic web services that use unfamiliar ontologies. **IEEE Intelligent Systems**, v.19, n.4, 2004, p.67-73.

COLMENARES, J. A.; EADS, G.; HOFMEYR, S.; BIRD, S.; MORETO, M.; CHOU, D.; GLUZMAN, B.; ROMAN, E.; BARTOLINI, D. B.; MOR, N.; ASANOVIC, K. and KUBIATOWICZ, J. D. Tessellation: refactoring the OS around explicit resource containers with continuous adaptation. **In: Design Automation Conference, Special Session on the Future of Operating Systems for Embedded Systems and Software (ESS).** 2013.

CANNATA, A.; GEROSA, M.; TAISCH, M. SOCRADES: a framework for developing intelligent systems in manufacturing. **In: IEEE International Conference on Industrial Engineering and Engineering Management, 2008 (IEEM 2008**). p. 1904–1908. 2008.

CATALANO, D.; RUFFO, G. A fair micro-payment scheme for profit sharing in P2P networks. In: International Workshop on Hot Topics in Peer-to-Peer Systems, 2004. pp. 32 – 39. 2004

CHENG, J.; KUNZ, T. **A survey on smart home networking**. Technical Report SCE-09-10, Sep. 2009.

CHESHIRE, S.; KROCHMAL, M. **RFC 6763:** DNS-based service discovery. 2013

CISCO IBSG. **The Internet of things**. 2011. Available at: <http://www.cisco.com/web/about/ac79/ docs/innov/IoT_IBSG_0411FINAL.pdf>. Access: Apr. 2014.

COALITION, D.-S et al. DAML-S: Web service description for the semantic web. **In: International Semantic Web Conference-ISWC**. Springer, 2002. p. 348–363.

COSTA, L. C. P.; ALMEIDA, N. S.; ZUFFO, M. K. Accessible display design to control home area networks. In: IEEE INTERNATIONAL CONFERENCE ON CONSUMER ELECTRONICS (ICCE), 2013, Las Vegas, United States. **Proceedings…**

COSTA, P. et al. The RUNES middleware: a reconfigurable component-based approach to networked embedded systems. In: IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC), 16, 2005. **Proceedings…** 2005. v. 2. p. 806-810.

CROCKFORD, D. **RFC4627**: The application/json media type for JavaScript Object Notation (JSON). 2006. Available at: <http://www.ietf.org/rfc/rfc4627.txt>. Access: Dec. 2014.

CULLER, D.; ESTRIN, D.; SRIVASTAVA, M. Overview of sensor networks**. IEEE Computer Society Magazine**, Aug. 2004.

DE, S. et al. Service modelling for the Internet of things. In: Federated Conference on Computer Science and Information Systems (FedCSIS), 2011. **Proceedings…** 2011. p. 949–955.

SILVA, G. S. H.; SILVA, L; ISHARA, P. W. K.; KUMARA, T. SmartBee; Multichannel access ZigBee gateway with plug and play device interface for Smart Home/Office Automation. **Information and Automation**, 2008.

DLNA. **DLNA Overview and Vision Whitepaper**. 2007. Available at: < http://www.homexpert.com.br/2012/wp-content/uploads/2012/03/dna_whitepaper.pdf > Access: Nov. 2014.

DLNA. **DLNA for HD video streaming in home networking environments.** Available at: <http://www.dlna.org/docs/white-papers/dlna-for-hd-video-streaming-in-home-networking-environments.pdf>. Access: May 2012.

DORIGO, M. Swarm-Bots and Swarmanoid: two experiments in embodied swarm intelligence. Web Intelligence and Intelligent Agent Technologies, 2009. WI-IAT '09. **IEEE/WIC/ACM International Joint Conferences on** , v.2, p.2-3, 2009.

EGGERT, L. Congestion control for the constrained application protocol (CoAP). IETF Network Working Group Internet Draft. Jun. 2010. Available at: <http://tools.ietf.org/html/draft-eggert-core-congestion-control-01>. Access: Mar. 2014.

FARREL, J.; LAUSEN, H. **Semantic annotations for WSDL and XML schema.** W3C Recommendation. n.28. 2007. Available at: <http://www.w3.org/TR/sawsdl/> Access: May 2014.

FERREIRA, P.; VEIGA, L.; RIBEIRO, C. OBIWAN: Design and Implementation of a Middleware Platform. **IEEE Transactions On Parallel And Distributed Systems**, v. 14, n. 11, Nov. 2003.

FIELDING, R. T.; TAYLOR, R. N. Principled design of the modern web architecture. **ACM Transactions on Internet Technologies**, v. 2, p.115-150. May 2002. Available at: <http://www.ics.uci.edu/~taylor/documents/2002-REST-TOIT.pdf>. Access: Dec. 2013.

FRISCH, K. von. **Bees; their vision, chemical senses, and language.** Ithaca: Cornell University Press, 1956.

GELERNTER, D. Generative communication in Linda**. ACM Transactions on Programming Languages and Systems (TOPLAS)**, v.7, n.1, p.80-112. New York, NY, USA: ACM, Jan. 1985.

GUINARD, D. et al. Interacting with the SOA-based internet of things: discovery, query, selection, and on-demand provisioning of web services. **IEEE Transactions on Services Computing**, v. 3, p. 223–235, 2010.

GUINARD, D.; TRIFA, V.; MATTERN, F.; WILDE, E. From the Internet of things to the web of things: Resource-oriented architecture and best practices. **Architecting the Internet of Things**. Springer, 2011, p. 97–129.

HAN, J.; YUN, J.; JANG, J.; PARK, K.-R. User friendly home automation based on 3D virtual world. **IEEE Transactions on Consumer Electronics**, v.56, n.3, p.1843-1847, Aug. 2010

HAVI. **HAVI Home Pages**. Available at: <http://www.havi.org>. Access: May 2012.

HELLER, M. **REST and CRUD: the impedance mismatch**. Infoworld. Jan. 2007. Available at: <http://www.infoworld.com/d/developer-world/rest-and-crud-impedance-mismatch-927>. Access: Mar. 2014.

HORNSBY, A.; BELIMPASAKIS, P.; DEFEE, I. XMPP-based wireless sensor network and its integration into the extended home environment. In: IEEE International Symposium on Consumer Electronics, 13, 2009, **Proceedings…** p.794-797

HUO, H. et al. An elderly health care system using wireless sensor networks at home. In: International Conference on Sensor Technologies and Applications, 3, 2009. **Proceedings…**

HUSTON, S. D.; JOHNSON, J. C. E.; SYYID, U. **The ACE programmer's guide:** practical design patterns for network and systems programming. United States: Addison-Wesley/Pearson Education. 2003, 544 p.

INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS. **IEEE Std 1588-2008**: IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems (Revision of IEEE Std 1588-2002). 2008a. 269 p.

INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS. **IEEE Std 802.1BA-2011**: IEEE Standard for Local and metropolitan area networks – Audio Video Bridging (AVB) Systems. 2011a. 45 p.

INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS. **IEEE Std 802.1AS-2011**: IEEE Standard for Local and Metropolitan Area Networks – Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks. 2011b. 292p.

INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS. **IEEE Std 802.1Qat-2010:** IEEE Standard for Local and Metropolitan Area Networks---Virtual Bridged Local Area Networks Amendment 14: Stream Reservation Protocol (SRP) (Revision of IEEE Std 802.1Q-2005). 2010b. 119 p.

INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS. **IEEE Std 802.1Qav-2009**: IEEE Standard for Local and Metropolitan Area Networks - Virtual Bridged Local Area Networks Amendment 12: Forwarding and Queuing Enhancements for Time-Sensitive Streams (Amendment to IEEE Std 802.1Q-2005). 2010a. 72 p.

INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS. **IEEE Std 1394-2008**: IEEE Standard for a High-Performance Serial Bus. 2008b. 954p.

INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS. **IEEE Std 1722-2011**: IEEE Standard for Layer 2 Transport Protocol for Time Sensitive Applications in a Bridged Local Area Network. 2011d. 57p.

INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS. **IEEE P1722.1/D23:** IEEE Standard for Device Discovery, Connection Management, and Control Protocol for IEEE 1722(TM) Based Devices. 2013. 380 p.

INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS. **IEEE Std 1733-2011:** IEEE Standard for Layer 3 Transport Protocol for Time-Sensitive Applications in Local Area Networks. 2011c. 21p.

INTEL. **Intel® Edison Development Platform**. Set. 2014. Available at: <https://communities.intel.com/docs/DOC-23139>. Access: Dec 2014.

INTERNATIONAL ELETROTECHNICAL COMISSION. **IEC 618883-1**: Consumer audio/video equipment – Digital interface – Part 1: General. 1998. 46p.

INTILLE S. S. Designing a home of the future. **IEEE Pervasive Computing.** v. 1, n. 2, p. 76-82, Apr. 2002.

ITU-T. X.509 : Information technology - open systems interconnection - the directory: public-key and attribute certificate frameworks. 2012. Available at: <http://www.ietf.org/mail-archive/web/wpkops/current/pdf9ygpIVtdwl.pdf>. Access: Dec. 2014.

JAVAUDIN, J. P.; BELLEC, M. OMEGA project: on convergent digital home networks. In: International Workshop on Cross Layer Design (IWCLD), 3, 2011, **Proceedings…** Nov. 2011. p. 1-5.

JENNINGS, N. R. Agent-based computing: promise and perils. In: International Joint Conference on Artificial Intelligence (IJCAI-99), 16, 1999. Stockholm, Sweden. **Proceedings…** p.1429-1436.

JERONIMO, M.; WEAST, J. **UPnP Design by Example**: a software developer's guide to Universal Plug and Play. United States: Intel Press, 2003.

JINI. **Jini™ Architecture Specification**. Available at: <http://river.apache.org/doc/specs/html/jini-spec.html >. Access: Mar. 2012.

KANG, E.; CHOI, W.; KIM, U. Distributed file discovery protocol in mobile peer-to-peer networks. In: International Conference on Networked Computing and Advanced Information Management, 4, 2008. **Proceedings…**

KATASONOV A. et al. Smart semantic middleware for the Internet of things. **ICINCO-ICSO**, v. 8, p.169-178, 2008.

KIFER, M.; LAUSEN, G. F-logic: a higher-order language for reasoning about objects, inheritance, and scheme. **In: ACM SIGMOD Record**. ACM, 1989, v. 18, p. 134-146.

KIM, D. S.; LEE, J. M.; KWON, W. H. Design and implementation of home network systems using UPnP middleware for networked appliances. **IEEE Transactions on Consumer Electronics**, v. 48, n. 4, p. 963-972, 2002.

KLYNE, G.; CARROLL, J. J.; MCBRIDE, B. **RDF 1.1 concepts and abstract syntax**. W3C Recommendation, n. 25, 2014. Available at: <http://www.w3.org/TR/rdf11-concepts/>. Access: May, 2014.

KOPECK, J. **Relationship of WSMO to other relevant technologies.** W3C Member Submission. Jun. 2005. Available at: <http://www.w3.org/Submission/WSMO-related/>. Access: Mar. 2014.

KOPECKY, J. et al. SAWSDL: semantic annotations for WSDL and XML schema. **In: IEEE Internet Computing**. v.11, n. 6, p.60–67, 2007.

KUBIATOWICZ, J.; PALMER, D. **A Case for the Universal DataPlane.** Available at: <http://www.terraswarm.org/pubs/116/kubitowicz_udplane_edge.pdf>. Access: Dec. 2014.

LASSILA, O. Using the semantic web in mobile and ubiquitous computing. In: BRAMER, M.; TERZIYAN, V. **Industrial Applications of Semantic Web.** US: Springer, n. 188, p. 19-25. 2005.

LEA, R. et al. Networking home entertainment devices with HAVi. **IEEE Computer**, v. 33, n. 9, p. 35-43, Sep. 2000.

LEE, E. A. et al. **The TerraSwarm research center (TSRC) (a white paper)**. Technical Report No. UCB/EECS-2012-207. Nov. 2012. Available at: <http://www.eecs.berkeley.edu/Pubs/TechRpts/2012/EECS-2012-207.html>. Access: Apr. 2014.

LEE, S. W.; PRENZEL, O.; BIEN, Z**.** Applying human learning principles to user-centered IoT systems. **Computer**, v.46, n.2, p.46-52, Feb. 2013.

LICKLIDER, J. C. R. Man-computer symbiosis. **IRE Transactions on Human Factors in Electronics**, v. HFE-1, p.4-11, Mar. 1960.

MARTIN, D. et al. **OWL-S: Semantic Markup for Web Services**. W3C Member Submission. 2004. Available at: <http://www.w3.org/Submission/OWL-S>. Access: May 2014.

MASUOKA, R et al. Ontology-enabled pervasive computing applications. **In: IEEE Intelligent Systems**. 2003. v. 18(5). p. 68–72.

MASUOKA, R.; PARSIA, B.; LABROU, Y. Task computing – the semantic web meets pervasive computing. In: FESNEL, D.; SYCARA, K.; MYLOPOULOS, J. **The Semantic Web - ISWC 2003**, in Lecture Notes in Computer Science. Berlin Heidelberg: Springer, n. 2870, p. 866–881. 2003.

MCILRAITH, S. A.; SON, T. C.; ZENG, H. Semantic web services. **IEEE Intelligent Systems**, v. 16, n. 2, p. 46–53. 2001.

MIAN, A. N.; BALDONI, R.; BERALDI, R. A survey of service discovery protocols in multihop mobile ad hoc networks. **Pervasive Computing Magazine,** p. 66-74, 2009

MICROSOFT CORPORATION. **An overview of the Simple Control Protocol (SCP).** 2005.

MITRA, N.; A. LAFON, Y. **SOAP version 1.2 Part 0**: Primer. 2nd ed. 2007. Available at: <http://www.w3.org/TR/soap12-part0/>. Access: May 2014

MOON, K. D.; LEE, Y. H.; LEE, C. E.; SON, Y. S. Design of a universal middleware bridge for device interoperability in heterogeneous home network middleware. **IEEE Transactions on Consumer Electronics**, v.51, n.1, Feb. 2005.

NARASIMHAN, P. Fault-tolerant CORBA: from specification to reality. **Computer**. 2007.

NIEWONLY, D. **How the Internet of things is revolutionizing healthcare**. 2013. Available at: <http://cache.freescale.com/files/corporate/doc/white_paper/IOTREVHEALCARWP.pdf>. Access: Jan 2015.

NOWAK, S. et al. Towards a convergent digital home network infrastructure. **IEEE Transactions on Consumer Electronics**, v.57, n.4, p.1695-1703, Nov. 2011.

ORGANIZATION FOR THE ADVANCEMENT OF STRUCTURED INFORMATION STANDARDS (OASIS). **Introduction to UDDI: important features and functional concepts**. v. 3. 2004.

OBITKO, M.; MARIK, V**.** Integrating transportation ontologies. In: International Conference on Applications of Holonic and Multi-Agent Systems – HoloMAS, 2, 2005, Copenhagen, Denmark. **Proceedings…**

OH, J. Y.; PARK, J. H.; JUNG, G. H.; KANG, S. J. CORBA based core middleware architecture supporting seamless interoperability between standard home network middlewares. **IEEE Transactions on Consumer Electronics**, v.49, n.3, Aug. 2003.

PARK, J. H.; LEE, M. J.; KANG, S. J. CORBA-based distributed and replicated resource repository architecture for hierarchically configurable home network. **Journal of Systems Architecture: the EUROMICRO Journal**. v.51, n.2, p.125-142, Feb. 2005.

PARSA, A.; ERCAN, A.O.; MALAGON, P.; BURGHARDT, F.; RABAEY, J.M.; WOLISZ, A. Connectivity Brokerage: From coexistence to collaboration. In: IEEE Radio and Wireless Symposium, Jan. 2010. **Proceedings…** p. 488-491.

RABAEY, J.M. The swarm at the edge of the cloud - A new perspective on wireless. In: Symposium on VLSI Circuits. Jun. 2011. **Proceedings…** p.6-8.

RANJAN, R.; HARWOOD, A.; BUYYA, R. Peer-to-peer-based resource discovery in global grids: a tutorial. **IEEE Communications**, v. 10, n. 2. 2008.

RASPBERRY PI FOUNDATION. United Kingdom. Presents information about platform and softwares for download. Available at: <http://www.raspberrypi.org>. Access on: Dec., 2014.

ROMAN D., et al. Web Service Modeling Ontology. **Applied Ontology**. v. 1, n. 1, p.77-106, 2005.

ROWE, A.; BERGES, M.E.; BHATIA, G.; GOLDMAN, E.; RAJKUMAR, R.; GARRET, J.H.; MOURA, J.M.F.; SOIBELMAN, L., Sensor Andrew: large-scale campus-wide sensing and actuation. **IBM Journal of Research and Development**, v. 55, n. 1.2, p. 6:1 - 6:14. 2011.

SAINT-ANDRE, P. **Extensible Messaging and Presence Protocol (XMPP):** Core (RFC6120). 2011. Available at: < http://tools.ietf.org/html/rfc6120>. Access: Dec. 2014.

SCHMIDT, C.; PARASHAR, M. A peer-to-peer approach to web service discovery. **World Wide Web: Internet and Web Information Systems**, v.7, p.211–229. Netherlands: Kluwer Academic Publishers, 2004.

SHARIFKHANI, F.; PAKRAVAN, M.R., A review of new advances in resource discovery approaches in unstructured P2P networks. In: **Advances in Computing, Communications and Informatics (ICACCI), 2013 International Conference on**, p.828-833, Aug. 2013.

SHELBY, Z. Embedded web services. **IEEE Wireless Communications**, v. 17, n. 6, p. 52-57. Dec. 2010. Avaiable at: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5675778&isnumber=5675766>. Access: Dec. 2013.
SHEU, P. C. –Y. et al. **Semantic Computing.** Hoboken, New Jersey, USA: John Wiley & Sons Inc, 2010.

SINGH, M. P.; HUHNS, M. N. **Service-oriented computing:** semantics, processes, agents. England: John Wiley & Sons, 2005.

SINGH, D.; TRIPATHI, G.; JARA, A.J., A survey of Internet-of-things: future vision, architecture, challenges and services, **Internet of Things (WF-IoT), 2014 IEEE World Forum on**, p.287-292, Mar. 2014

SMITH, M.; WELTY, C.; MCGUINESS, D. L. **OWL Web Ontology Language Guide**. W3C Recommendation. 2004. Available at: <http://www.w3.org/TR/owl-guide/>. Access: May, 2014.

SONG, Z.; CARDENAS, A. and MASUOKA, R. Semantic middleware for the Internet of things. **In: Internet of Things (IOT)**. 2010. p. 1–8. 2010.

SUH C.; KO Y. Design and implementation of intelligent home control systems based on active sensor networks. **IEEE Transactions on Consumer Electronics**, v.54, n.3, p.1177-1184, Aug. 2008.

SUN MICROSYSTEMS - THE INTERNET SOCIETY. **JXTA v2.0 Protocols Specification**. 2007. Available at: <http://jxta.kenai.com/Specifications/JXTAProtocols2_0.pdf>. Access: Jul. 2012.

SURIE D., LAGUIONIE O., PEDERSON T. Wireless sensor networking of everyday objects in a smart home environment. In: International Conference on Intelligent Sensors, Sensor

Networks and Information Processing, 2008 (ISSNIP 2008). Dec. 2008. **Procedings**… p.189-194.

TALEB, T. et al. ANGELAH: a framework for assisting elders at home. **IEEE Journal on Selected Areas in Communications**, v. 27, n.4, p.480-494, 2009

TAO WEBSITE. **Overview of the ACE+TAO Project.** Available at: <http://www.cs.wustl.edu/~schmidt/TAO.html>. Access: May 2012.

TEENER, M.D. et al. Heterogeneous networks for audio and video: using IEEE 802.1 audio video bridging. **Proceedings of the IEEE**, v.101, n.11, p.2339-2354, Nov. 2013.

THE OSGI ALLIANCE. **OSGi Service Platform Residential Specification Version 4.3**. Jan. 2012. Available at: <http://www.osgi.org/Download/Release4V43>. Access: May 2012.

TOKUNAGA, E. et al. A framework for connecting home computing middleware. 22nd International Conference on Distributed Computing Systems Workshops, 2002. **Proceedings…** p.765-770.

UPNP. Standards: device control protocols. Available at: <http://upnp.org/sdcps-and-certification/standards/sdcps/>. Access: 2012.

UPTON, D. M. The Management of Manufacturing Flexibility. **California Management Review**, v. 36, n. 2, p. 72-89, 1994.

WEISS, G. **Multiagent Systems**: a modern approach to distributed artificial intelligence. Cambridge, Massachusetts: MIT Press, 1998.

WILDE, E. **Putting Things to REST.** UCB iSchool Report, n. 2007- 015. 2007. Available at: < http://dret.net/netdret/docs/wilde-irep07-015-restful-things.pdf>. Access: July, 2015.

WILSON, M.; MAGILL, E.H.; KOLBERG, M., An online approach for the service interaction problem in home automation. In: IEEE Consumer Communications and Networking Conference, 2, 2005. **Proceedings…** p. 251-256.

WOOLDRIDGE, M. **An introduction to multiagent systems**. 2nd ed. England: John Wiley & Sons, 2009.