Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingenieros de Telecomunicación

Application of Agent Technology for Fault Diagnosis of Telecommunication Networks

Tesis Doctoral

Álvaro Carrera Barroso
Ingeniero de Telecomunicación

2016

Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingenieros de Telecomunicación

Application of Agent Technology for Fault Diagnosis of Telecommunication Networks

Tesis Doctoral

Álvaro Carrera Barroso
Ingeniero de Telecomunicación

2016

Departamento de Ingeniería de Sistemas Telemáticos

Escuela Técnica Superior de Ingenieros de Telecomunicación

Universidad Politécnica de Madrid

Application of Agent Technology for Fault Diagnosis of Telecommunication Networks

Autor:

Álvaro Carrera Barroso

Ingeniero de Telecomunicación

Tutor:

Carlos Ángel Iglesias Fernández

Doctor Ingeniero de Telecomunicación

2016

Tribunal nombrado por el Magfco. y Excmo. Sr. Rector de la Universidad Politécnica de Madrid, el día _____ de _____ de _____.

Presidente: _____

Vocal: _____

Vocal: _____

Vocal: _____

Secretario: _____

Suplente: _____

Suplente: _____

Realizado el acto de defensa y lectura de la Tesis el día _____ de _____ de _____. en la E.T.S.I. Telecomunicación habiendo obtenido la calificación de _____.

EL PRESIDENTE LOS VOCALES

EL SECRETARIO

A mi madre, a mi padre y a Tamara, gracias por todo.

Agradecimientos

Me gustaría incluir en estos agradecimientos a todas aquellas personas que me han dado ánimos para realizar este trabajo y que han estado día a día motivándome para poder concluirlo.

En especial, me gustaría mencionar a mi madre y a mi padre por haberme brindado la oportunidad de desarrollar mi vocación; a Tamara por haber estado siempre ahí en los momentos de necesidad; a mis suegros y cuñados; a mis compañeros y amigos: Miguel, Juan Fernando, Vicente, Laura (Díaz), Paco, José Ignacio, Alberto, José Javier, Jorge Juan, Jesús, David, Laura (Galán), Gonzalo, Bea, Gema, Carlos, Daniel, Óscar, Emilio, Geovanny, Shenjing, Ganggao y tantos otros.

No puedo olvidar a toda la gente que me acogió en Telefónica I+D y me ofrecieron la posibilidad de comenzar a trabajar en los temas que se han desarrollado en esta tesis. Gracias a todos y en especial a Javier (Algarra), a Javier (González), a Pablo, a Raquel, a Andrés y a todos los demás que también me dieron ánimos y motivación para realizar este trabajo.

Tampoco me querría olvidar de los londinenses que me acogieron entre ellos durante unos meses, especialmente de Jose, Sara y Bea; a Ashley por el asesoramiento con el título de esta tesis; y finalmente, a Eduardo, por acogerme bajo su tutela durante mi estancia en Londres.

También agradecer, por supuesto, a Carlos Ángel por la dirección de esta tesis y su infinita paciencia. También, al resto profesores del Grupo de Sistemas Inteligentes: Mercedes, Gregorio, Marifeli, José Carlos y tantos otros.

Y por último, también querría agradecer al resto de familiares, amigos y compañeros que no hayan sido ya mencionados; pero por desgracia, la lista no puede ser infinita.

Gracias, sin vosotros no lo habría conseguido.

Resumen

La presente tesis doctoral contribuye al problema del diagnóstico autonómico de fallos en redes de telecomunicación. En las redes de telecomunicación actuales, las operadoras realizan tareas de diagnóstico de forma manual. Dichas operaciones deben ser llevadas a cabo por ingenieros altamente cualificados que cada vez tienen más dificultades a la hora de gestionar debidamente el crecimiento exponencial de la red tanto en tamaño, complejidad y heterogeneidad. Además, el advenimiento del Internet del Futuro hace que la demanda de sistemas que simplifiquen y automaticen la gestión de las redes de telecomunicación se haya incrementado en los últimos años.

Para extraer el conocimiento necesario para desarrollar las soluciones propuestas y facilitar su adopción por los operadores de red, se propone una metodología de pruebas de aceptación para sistemas multi-agente enfocada en simplificar la comunicación entre los diferentes grupos de trabajo involucrados en todo proyecto de desarrollo software: clientes y desarrolladores.

Para contribuir a la solución del problema del diagnóstico autonómico de fallos, se propone una arquitectura de agente capaz de diagnosticar fallos en redes de telecomunicación de manera autónoma. Dicha arquitectura extiende el modelo de agente Belief-Desire-Intention (BDI) con diferentes modelos de diagnóstico que gestionan las diferentes sub-tareas del proceso. La arquitectura propuesta combina diferentes técnicas de razonamiento para alcanzar su propósito gracias a un modelo estructural de la red, que usa razonamiento basado en ontologías, y un modelo causal de fallos, que usa razonamiento Bayesiano para gestionar debidamente la incertidumbre del proceso de diagnóstico.

Para asegurar la adecuación de la arquitectura propuesta en situaciones de gran complejidad y heterogeneidad, se propone un marco de argumentación que permite diagnosticar a agentes que estén ejecutando en dominios federados. Para la aplicación de este marco en un sistema multi-agente, se propone un protocolo de coordinación en el que los agentes dialogan hasta alcanzar una conclusión para un caso de diagnóstico concreto.

Como trabajos futuros, se consideran la extensión de la arquitectura para abordar otros problemas de gestión como el auto-descubrimiento o la auto-optimización, el uso de técnicas

de reputación dentro del marco de argumentación para mejorar la extensibilidad del sistema de diagnóstico en entornos federados y la aplicación de las arquitecturas propuestas en las arquitecturas de red emergentes, como SDN, que ofrecen mayor capacidad de interacción con la red.

Abstract

This PhD thesis contributes to the problem of autonomic fault diagnosis of telecommunication networks. Nowadays, in telecommunication networks, operators perform manual diagnosis tasks. Those operations must be carried out by high skilled network engineers which have increasing difficulties to properly manage the growing of those networks, both in size, complexity and heterogeneity. Moreover, the advent of the Future Internet makes the demand of solutions which simplifies and automates the telecommunication network management has been increased in recent years.

To collect the domain knowledge required to developed the proposed solutions and to simplify its adoption by the operators, an agile testing methodology is defined for multi-agent systems. This methodology is focused on the communication gap between the different work groups involved in any software development project, stakeholders and developers.

To contribute to overcoming the problem of autonomic fault diagnosis, an agent architecture for fault diagnosis of telecommunication networks is defined. That architecture extends the Belief-Desire-Intention (BDI) agent model with different diagnostic models which handle the different subtasks of the process. The proposed architecture combines different reasoning techniques to achieve its objective using a structural model of the network, which uses ontology-based reasoning, and a causal model, which uses Bayesian reasoning to properly handle the uncertainty of the diagnosis process.

To ensure the suitability of the proposed architecture in complex and heterogeneous environments, an argumentation framework is defined. This framework allows agents to perform fault diagnosis in federated domains. To apply this framework in a multi-agent system, a coordination protocol is defined. This protocol is used by agents to dialogue until a reliable conclusion for a specific diagnosis case is reached.

Future work comprises the further extension of the agent architecture to approach other managements problems, such as self-discovery or self-optimisation; the application of reputation techniques in the argumentation framework to improve the extensibility of the diagnostic system in federated domains; and the application of the proposed agent architecture in emergent networking architectures, such as SDN, which offers new capabilities of control

for the network.

Contents

CHAPTER 1

Introduction

In this chapter, the motivation and the objectives of this thesis are introduced to the reader. We summarise the problems of the network management for the Future Internet and highlight a set of challenges where the complexity and heterogeneity of the network are considered as one of the most important problems to face in the next generation of network management systems. Finally, we outline the solution proposed to achieve the objectives of this thesis.

1.1 Motivation

Telecommunication companies have seen increased their activity exponentially in the last decades (Cetinkaya et al., 2013). As a consequence, telecommunication networks have been growing constantly, both in size, heterogeneity and complexity. The current Internet is based on the premise of a simple network service used to interconnect end systems where relatively intelligent services were running. That simplicity has allowed a huge growth of the network since the beginnings of the primitive Internet (Evans, 2011). But the management approach followed by network operators in the current Internet is obstructing the evolution of the network. So, network management is really challenging for next generation networks (Plevyak and Sahin, 2011). The Future Internet will need to optimise the use of its resources continuously and recover from problems, faults or attacks transparently for the network operator and without impact on the services running over it (Charalambides et al., 2011). To achieve those goals, networks will need to be more intelligent and adaptive than the current ones and their management systems will need to be embedded in the network itself instead of being external systems (Jennings et al., 2007).

In those next generation networks, many different actors will interact dynamically to offer reliable end-to-end services. That diversity of actors (users, sensors, devices, content providers, etc.) will make network operation and management very hard for the traditional network management approach (Galis et al., 2009). Services deployed on the top of the networks will be considered as one of those actors that will have to cooperate autonomously with other actors to get the expected result (Tselentis and Galis, 2010). That dynamic and autonomous cooperation among actors is a key requirement to get flexible and efficient networks (Müller, 2012). Moreover, that complexity of the Future Internet will bring a high level of uncertainty to management tasks (Guckenheimer and Ottino, 2008). But, that uncertainty is not only an issue for the Future Internet, current Internet deals with it as exposed by Clark et al. (2007). They estimate that the current Internet is over-dimensioned by a factor of 400% to ensure its performance under almost any conditions. It means that the strategy of current Internet is to over-size the network to ensure its availability. But, maintaining this strategy in the Future Internet would be very inefficient and costly. Indeed, the cost of network management and support has increased drastically in recent years due to the complexity of the network technologies requiring ever more skilled engineers and administrators and rounding 200 billions dollars (Agoulmine, 2011). Therefore, management of the uncertainty coming from complex networks will be an essential requirement for any management system of the Future Internet (Pras et al., 2007).

To deal with that complexity, autonomic approaches have been proposed both for com-

puting (Kephart et al., 2007) and networking (Strassner et al., 2007; Tschudin and Jelger, 2007). This trend tries to achieve self-management capabilities with a Monitor-Analyze-Plan-Execute (MAPE) control loop (Kephart et al., 2007) implemented by autonomic managers. Those autonomic managers must perform different management tasks, such as self-configuring, self-healing, self-optimising and self-protecting. However, a single isolated autonomic manager can achieve autonomic behaviour only for the resources it manages, which can lead to scalability problems. To avoid those problems, the managers must be coordinated to obtain a global autonomic management of the network.

Autonomic approaches require innovative aspects and mechanisms to enable the desired self-* capabilities to govern an integrated behaviour of the Future Internet (Bouabene et al., 2010). Those mechanisms are based on the usage of specific domain knowledge of network engineering taking into consideration the dynamicity and complexity of the supervised systems. The European Telecommunications Standards Institute (ETSI) supports this autonomic approach with a generic reference model for autonomic networking named Generic Autonomic Network Architecture (GANA) (Laurent et al., 2013) which defines a set of desired properties for those autonomic systems. Those desired properties are: automation, awareness, adaptiveness, stability, scalability, robustness, security, switchable and federation. Getting them in an autonomic management system is really challenging for network operators following the traditional management approach. Thus, Laurent et al. (2013) define some enabling concepts and mechanisms for further research to achieve those desired properties in the management systems of the autonomic Future Internet. This autonomic approach is supported by the Internet Research Task Force (IRTF). Behringer et al. (2015) describe the design goals of the autonomic networking in the Request For Comments (RFC) 7575, and Jiang et al. (2015) analyse the general gap for autonomic networking reviewing the current status of autonomic aspects of current networks. Among others, they identify troubleshooting and recovery as one of the non-autonomic behaviour of the current Internet in the RFC 7576, which motivated us to develop this thesis in the field of autonomic fault diagnosis of telecommunication networks.

In conclusion, due to the increasing complexity, heterogeneity and the consequent high level of uncertainty in telecommunication networks, autonomic fault management is an interesting research field for operator companies. Accordingly, our motivation when preparing this thesis was to improve the current situation with relation to aforementioned challenges for the autonomic Future Internet. In particular, we are motivated by the fact that there is still a lack of solutions for autonomic fault diagnosis mechanisms.

1.2 Objectives

The primary objective of the thesis is to define a scalable and reliable solution for Fault Diagnosis tasks in Telecommunication Network Management. By referring to the previous section, the solution must meet some requirements that focus on the main challenges of the Future Internet management, such as to handle uncertainty and cooperate in complex federated scenarios. Therefore, we have decomposed the thesis global objective into a number of more specific ones in order to build the final solution step by step:

- **(1) Facilitate the adoption of fault diagnosis solutions by network operators.** Our objective is to define a methodology that facilitate the adoption of the autonomic solutions focusing on requirement gathering. It should be focus on gather the knowledge required to design a fault diagnosis solution facilitating the communication between stakeholders and designers.

- **(2) Define an autonomic fault diagnosis solution which can handle properly the uncertainty of complex systems.** Our objective is to define a solution suitable for the network management of the Future Internet. The formalisation of the solution should put impact on handling the uncertainty in complex systems and should be independent of the network technology.

- **(3) Define an autonomic fault diagnosis solution which works in federated environments.** Our objective is to define a solution to allow fault diagnosis systems to work in the heterogeneous environment of the Future Internet, where multiple actors in different federated domains have to cooperate to achieve agreements about diagnosis conclusions.

1.3 Solution Outline

In order to fulfil the stated objectives, we propose a number of solutions put together in a single framework to facilitates their adoption. This framework aims to provide operators an autonomic fault diagnosis solution for telecommunication networks based on agent technology (see Figure 1.1).

On the top of the contemporary state of the art on requirement engineering for software development projects, we propose a methodology to bridge the communication gap between network operators and system designers (Network Level). The aims of this methodology is

to gather the required *domain knowledge* to design the appropriate solution for the fault diagnosis problem.

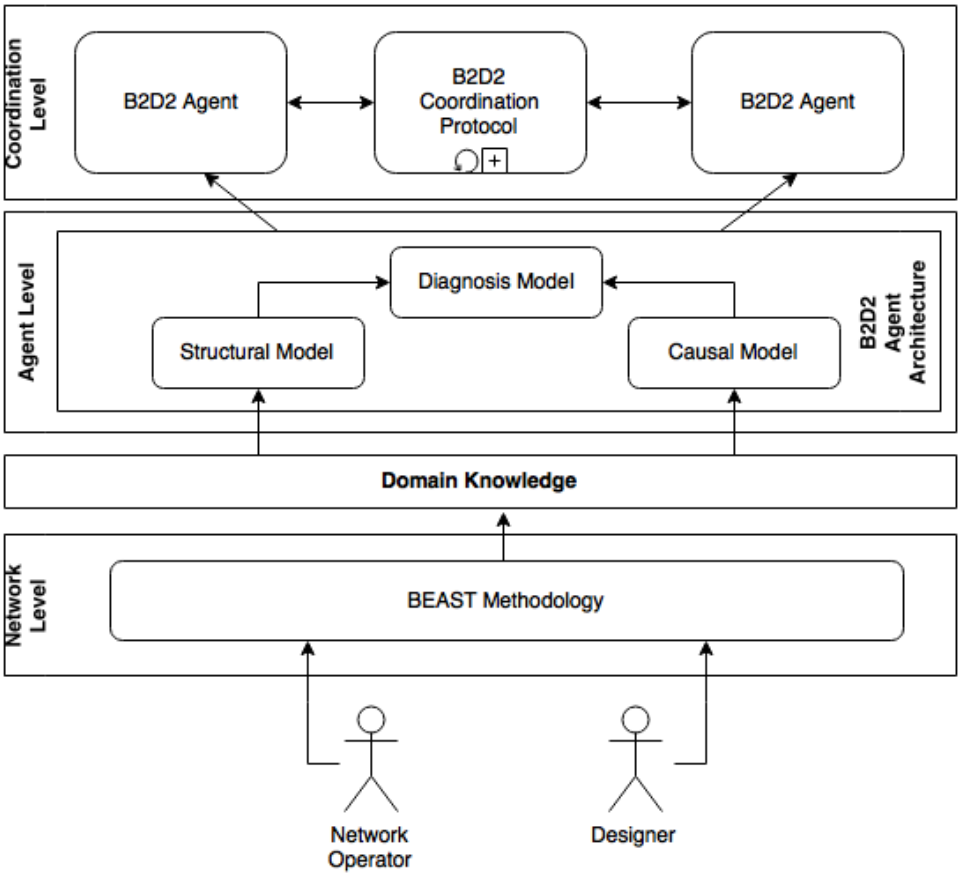

Figure 1.1: Overview of the solution proposed in this thesis.

The designed solution constitutes an agent architecture (Agent Level). This agent architecture is based on the BDI model which is extended with a set of diagnostic models to carry out properly the diagnosis process under uncertainty. A *Structural Model* of the network and a *Causal Model* of fault-symptom relations are combined with a *Diagnosis Model* in the knowledge plane of the agent to handle all diagnosis information. Due to the complexity of the considered scenario, a coordination mechanism is required to ensure the suitability of the proposed agent model in heterogeneous network scenarios with federated domains. Accordingly, we propose an argumentation-based technique to allow agents in different domains to cooperate (Coordination Level). The proposed framework consists of three main contributions:

- **(1) BEAST Methodology.** We propose an Acceptance Testing Methodology for Multi-Agent Systems development (**Objective 1**) for domain knowledge gathering.

The methodology is composed by 4 different phases: *System Behaviour Specification* to define the expected outcome of the system and gather the domain knowledge from network operators, *MAS Behaviour Specification* to translate the previous system specification to agents behaviours, *Agent Level Testing* to test agents individually while they are under development, and finally, *MAS Level Testing* to test and evaluate the behaviour of the global system in a testing environment. This contribution is described in Chapter 2.

- **(2) B2D2 Agent Architecture.** We propose an Agent Architecture to carry out the Autonomic Fault Diagnosis tasks handling the uncertainty of a complex telecommunication network (**Objective 2**). The proposed architecture follows a task model which defines the phases of the diagnosis process. We propose a Diagnosis Model which covers the inference knowledge required to carry out autonomic fault diagnosis tasks and combines different domain knowledge models to allow agents to deal with network elements and their faults under uncertainty. This contribution is described in Chapter 3.

- **(3) B2D2 Coordination Framework.** We propose an extension of the B2D2 Agent Architecture to add argumentative capabilities. Those argumentative capabilities are used in a Coordination Protocol which allows agents to argue about diagnosis cases in a federated domains environment (**Objective 3**). The coordination protocol is based on an argumentation framework which has been defined focused on the restrictions of a federated scenario where agents have to deal with uncertain knowledge. This contribution is described in Chapter 4.

1.4 Thesis Organization

This thesis is organised as follows. Chapter 2 describes the proposed methodology for knowledge gathering. Chapter 3 describes the agent architecture for autonomic fault diagnosis of telecommunication networks. Chapter 4 describes the coordination protocol to allow agents to diagnose faults in federated domains. All these chapters include different sections for related works of each topic and the corresponding evaluations of the proposal exposed in them. Finally, Chapter 5 concludes the thesis summarising the contributions and proposing possible future research to continue this work.

CHAPTER 2

Knowledge Gathering for Autonomic Fault Diagnosis

Following the objective of facilitating the adoption of the solutions presented in this thesis, this chapter presents an acceptance testing methodology that bridges the communication gap between stakeholders and developers, named BEhavioural Agent Simple Testing (BEAST) Methodology. The methodology is focused on gathering the domain knowledge from expert network operators for building autonomic fault diagnosis agents. The BEAST Methodology presents traceability from user requirements to test cases, enabling stakeholders can be aware of the project status providing executable requirements. This methodology fits with agile software development methodologies, such as SCRUM, allowing an incremental development process and involving different actors in the project. Moreover, an open source tool that supports the proposed methodology is presented. This tool supports the automatic generation of test cases from requirements which enhances the traceability between them. The methodology and the associated tool have been validated in the development of a Multi-Agent System (MAS) for fault diagnosis of a FTTH network.

2.1 Introduction

Nowadays, the application of multidisciplinary techniques from different engineering disciplines are becoming a necessity to solve complex problems, which makes interdisciplinary methodologies more and more important (Borutzky, 2010). This thesis aims to contribute to the application of agent technology for fault diagnosis of telecommunication networks, which merges Agent Oriented Software Engineering (AOSE) with network engineering. The domain knowledge required to develop systems which interact with the network must be acquired from the expert network engineers who deal with it everyday. But the gap between both engineering disciplines makes that acquisition process is a difficult task. Therefore, this chapter presents an agile acceptance testing methodology focused on facilitating the communication between stakeholders and developers.

The context of this work was a research project contracted by the company Telefónica R&D. They requested to develop a multi-agent system for fault diagnosis in their network. From a software engineering point of view, the main challenges were: (i) they required managing the project using the SCRUM Agile Methodology (Schwaber and Sutherland, 2009), (ii) the project involved integration with a wide range of external systems and the emulation of faulty behaviour of network transmission and (iii) the development team was composed of students with different timetables, so they were not working together most of the time. After the first release, the main problems we encountered were communication problems between the development team and the customer (expert network engineers), communication problems within the development team, where agents were being developed in parallel, and lack of automation in the unit testing process, which involved to test physical connections with a manual and very time consuming process.

After analysing several AOSE proposals based on agile principles (Clynch and Collier, 2007; García-Magariño et al., 2009), we have not found any proposal which covers *acceptance tests* and provides a good starting point for its application in an agile context. Thus, this research aims at bridging the gap between acceptance testing and AOSE. The key motivation of this methodology is to explore to what extent acceptance testing can benefit MAS development, in order to provide support in the development of MAS in agile environments. This brought us to identify the need for an agile acceptance testing methodology for MAS.

The main contribution of this chapter is that the Behaviour Driven Development (BDD) approach has been suitable for its application in MAS development. Furthermore, the use of BDD facilitates the communication between stakeholders and designers or developers, which is usually a gap between both of them. In the framework of this thesis, this methodology

provides a communication channel with network operators to extract the domain knowledge required to design the autonomic fault diagnosis agents.

The rest of the chapter is structured as follows. Firstly, Section 2.2 discusses related work in the research field of agile acceptance testing. Section 2.3 describes the agile acceptance testing methodology for MAS based on BDD techniques and its application to the project mentioned above. Section 2.4 provides an overview of the open source tool that supports the proposed methodology. Finally, Section 2.5 presents some concluding remarks of this chapter.

2.2 Related Work

Understanding stakeholders requirements and fulfilling their desired functionality is considered as the most important aspect for a software project to be considered successful (Agarwal and Rathod, 2006). Thus, requirements engineering plays a key role in the development process. The main challenges of requirements engineering are (Marnewick et al., 2011): (i) improving the communication between the stakeholders and the development team and (ii) understanding the problem. These challenges are even more difficult to solve in complex environments (Jarke and Lyytinen, 2015), but they can be faced using more holistic thinking and decomposing appropriately the global problem (Katina et al., 2014).

Nevertheless, the process of eliciting requirements and communicating them is still an issue and some authors consider it the *next bottleneck to be removed from the software development process* (Adzic, 2009). The main reasons for this communication gap between stakeholders and the development team are that (Adzic, 2009) (i) imperative requirements are very easy to misunderstand; (ii) even the obvious aspects are not so obvious and can be misinterpreted and (iii) requirements are over-specified, since they are expressed as a solution, and focus on what to do and not why, not allowing the development team whether discuss if those requirements are the best way to achieve stakeholders' expectations.

In order to bridge the communication gap between developers and stakeholders, the agile movement has proposed to shift the focus of requirements gathering. Instead of following a contractual approach where the requirements documents is the most important goal, they put emphasis on improving the communication among all the stakeholders and developers to have a common understanding of these requirements. Many approaches have been explored for requirements gathering, such as the use of ontologies for modelling the user requirements (Sun et al., 2010) or the definition of UML models for capturing quality requirements (Guerra-García et al., 2013). Moreover, given that requirements will have

inconsistencies and gaps (Adzic, 2011), it has been proposed to anticipate the detection of these problems by checking the requirements as soon as possible, even before the system is developed. In this line, Martin and Melnik (2008) formulated the equivalence hypothesis: "As formality increases, tests and requirements become indistinguishable. At the limit, tests and requirements are equivalent", which is the basis idea of executable requirements, i.e. a test (or set of tests) used to check if a requirement is met.

As a result, they have proposed a practice so called *agile acceptance testing*, whose purpose is improving communication by using real-world examples for discussion and specifications of the expected behaviour at the same time, which is called Specification by Example (SBE). Different authors have proposed to express the examples in a tabular form (Acceptance Test Driven Development (ATDD)[1] with Fit test framework (Mugridge and Cunningham, 2005)) or as scenarios (BDD (North, 2007) with tools such as JBehave (North, 2011) or Cucumber (Wynne and Hellesy, 2008)). In this way, requirements are expressed as acceptance tests, and these tests are automated. When an agile methodology is followed, acceptance tests can be checked in an automated way during each iteration, and thus, requirements can be progressively improved. Most frameworks provide a straight forward transition from acceptance tests to functional tests based on tools such as the xUnit family (Hamill, 2004) or even automating the test case generation (Kamalrudin and Sidek, 2014). Agile acceptance testing complements Test Driven Developoment (TDD) practices, and it can be seen as a natural extension of TDD practices, which have become mainstream in among software developers. In this way, software project management can be based not only on estimations but on the results of acceptance and functional tests. In addition, these practices facilitate to maintain requirements (i.e. acceptance tests) updated along the project lifespan.

In the multi-agent field, there have been several efforts in the testing of final systems. MAS testing present several challenges (Nguyen, 2009), given that agents are distributed, autonomous and it is interesting not its individual behaviour but the emergent behaviour of the multi-agent system that arises from the interaction among individual behaviours. A good literature review of MAS testing can be found in (Nguyen, 2009; Nguyen et al., 2011; Houhamdi, 2011). Thangarajah et al. (2011) propose to extend the scenarios of the Prometheus Methodology (Padgham and Winikoff, 2003) in order to be able to do testing of scenarios as part of requirements or acceptance testing. The work describes also a novel technique for integrating agent simulation in the testing process. Nevertheless, their proposal of acceptance tests seems targeted at technical users familiarised with AOSE vocabulary, given than the scenarios are described in terms such as percepts, goals and actions.

[1] A literate review of ATDD can be found in (Haugset and Hanssen, 2008).

Nguyen et al. (Nguyen et al., 2010) propose an extension of the Tropos Methodology (Bresciani et al., 2004) by defining a testing framework that takes into account the strong link between requirements and test cases. They distinguish external and internal testing, but they focused on the internal one. External testing produces acceptance tests for being validated by project stakeholders, while internal testing produces system and agent tests for being verified by developers.

As conclusion, to the best of our knowledge, there is no previous work dealing explicitly with acceptance testing in AOSE for agile environments. This motivated us to define an agile acceptance testing methodology for MAS.

2.3 BEAST Methodology

To cover the problems identified above, we should identify which requirements should have the testing methodology. First, our primary concern is that the methodology should help in improving the communication between the stakeholders and the development team, as well as the communication among the development team. Another requirement comes from the overall methodology: it should be compatible and suitable for its application in combination with agile techniques. Finally, it should not be tied to a specific MAS tool or framework, such as JADE or JADEX, and it should be feasible to integrate with other MAS environments with low effort.

The BEAST Methodology is intended to be used in agile environments, with special focus on providing traceability from stakeholder requirements to test cases. With this end, requirements are automated as *acceptance tests*, which are linked with MAS testing. The main benefit of this approach is that it improves the understanding of the real progress of the project from the stakeholders perspective, and, moreover, it provides a good basis for reviewing the objectives of every iteration. As a result, requirements negotiation and specification can be done in an iterative way, and can be adapted to the improved understanding of the desired system by both stakeholders and development team.

The methodology consists of four phases: *System Behaviour Specification*, *MAS Behaviour Specification*, *Agent Level Testing* and *MAS Level Testing*, which can be applied in every iteration of any agile development methodology. Figure 2.1 depicts these steps and the actors that appear in each one of them.

During the first step (*System Behaviour Specification* phase), the expected behaviour of the system is specified by the customer, the product owner and, at least, one member of the

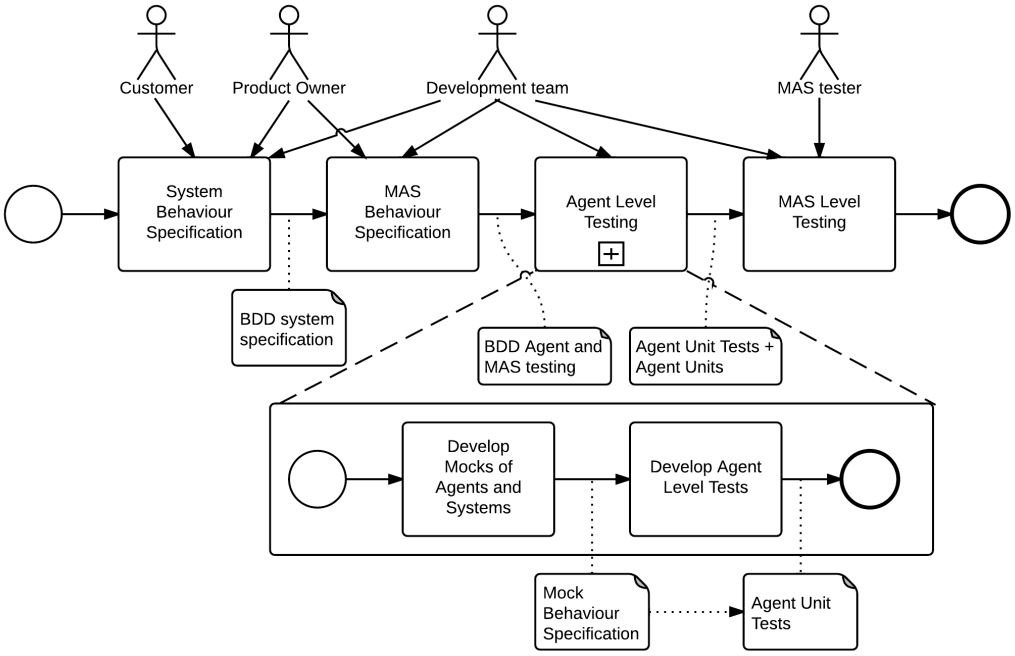

Figure 2.1: Overview of the Beast Methodology.

development team, as it is described in the SCRUM Methodology (Schwaber and Sutherland, 2009). This specification is done following the Behaviour Driven Development (BDD) technique, as shown in Section 2.3.1. Once the BDD system specification is available, the product owner and the designer of the development team must translate the system specification into agent behaviours specification during the *MAS Behaviour Specification* phase, as shown in Section 2.3.2. The output of this step is a set of BDD requirements for the MAS that are implemented and tested in the following step of the methodology. During this *Agent Level Testing* phase, shown in Section 2.3.3, the methodology proposes the use of mocking techniques to replace other agents which are not developed yet or external systems that are not available during the development phase. After the behaviour of all agents have been tested, the *MAS Level Testing* phase has two sub-phases, as shown Section 2.3.4.. First of all, once agents have been developed, integration testing can be done replacing mocks by the real agents. Second, *emergent features* should be validated in a testing environment. Both simulation techniques or testbed scenarios can complement this phase to simulate different system configurations. Thus, acceptance testing is straight forward, and the expectations of the stakeholders can be checked without discussing about ambiguities or omissions in the requirements document thanks to the traceability from project requirements to test cases.

To properly frame the BEAST Methodology, the motivational network management project contracted by the company Telefónica R&D has been chosen as case study. In this

project, the stakeholder is a network operator company which wants a tool to reduce the management cost of Fiber To The Home (FTTH) networks. The first task of the project was to write a high level project proposal and to explore different possible approaches to solve the problem. The result of this phase was that the solution that best fits the problem is a MAS architecture. SCRUM Agile Methodology (Schwaber and Sutherland, 2009) and BEAST Methodology, supported by the developed BEAST Tool, was used to manage the progress of the project. To facilitate the reading and understanding of the methodology, the following sections include the application of every phase of the methodology to the case study.

2.3.1 System Behaviour Specification

The *System Behaviour Specification* phase aims at providing a communication bridge between the project stakeholders and the development team during requirements gathering. This phase follows the BDD technique (North, 2007). System behaviours are derived from the business outcomes that the system intends to produce. These business outcomes should be prioritized by the stakeholders. Then, business outcomes are drilled down to feature sets. A feature set decompose a business outcome into a set of abstract features, which show what should be done to achieve a business outcome. These feature sets are the result of discussions between stakeholders and developers. The expected features are described using *User Stories*. Later, *User Stories* are exposed in scenarios for each particular instantiation of a *User Story*. In other words, scenarios exemplify a *User Story* to cover all possible variations of the presented feature. Thus, those scenarios are the basis of acceptance tests.

```
[Story title] - description
As a [Role]
I want a [Feature]
So that [Benefit]
```

Table 2.1: User Story template (North, 2007).

```
Scenario [Scenario name]
Given [Context]
And [Some more contexts] ...
When [Event]
Then [Outcome]
And [Some more outcomes] ...
```

Table 2.2: Scenario template (North, 2007).

Instead of using plain natural language, BDD proposes the usage of textual templates. Table 2.1 presents the template for a *User Story*. This template presents a feature, i.e. a requirement, of the system and the benefit that this feature has from the point of view of a specific role, such as a final client or a system administrator. Table 2.2 presents the template

for a scenario. A set of scenarios must exemplify a *User Story* giving specific situations to well understand the feature and to test the system meets the requirement. These templates should be instantiated by the pertinent concepts of the domain knowledge. Those concepts are part of the *ubiquitous language* (Evans, 2004) which establishes the common terminology used by stakeholders and developers. Thus, these terms will be used in the instantiation of the domain knowledge models of agents, helping to reduce the gap between technical and business terminology.

This phase in the case study started with a meeting with the stakeholders to specify a set of initial requirements. Those requirements were written in BDD format as *User Stories*. Table 2.3 shows an example of one gathered requirement (User Story, lines 1-4) with two associated scenarios (lines 6-9 and 11-18).

```
1  Story: Time-to-repair cut down
2  As an operator network,
3  I want to have a system to diagnose root cause of faults
4  So that time-to-repair is below the SLA with the customer.
5  ******
6  Scenario: System monitors streaming sessions
7  Given a user that has a Video On Demand (VoD) service,
8  When the customer is consuming any streaming content,
9  Then the system must monitor the quality of the streaming session.
10 ------
11 Scenario: System diagnoses a QoS decreasing failure
12 Given a user that has a Video On Demand (VoD) service connected through
13   an FTTH access network
14 And the customer requests a film from the streaming server,
15 When loss rate is higher to 1%, latency is higher to 150ms or jitter
16   is higher to 30ms,
17 Then the system must diagnose the root cause of fault is
18   'Damaged Fibre', 'Inadequate Bandwidth' or 'Damaged Splitter'.
```

Table 2.3: Example of User Story.

Notice that stakeholders do not know anything about the solution, in this case, a Multi-Agent System (MAS). So, the written requirements, or *User Stories*, do not refer at all to agents, only to desired features. The translation from these requirements to agents is done by the designer following a MAS design methodology, as shown in Section 2.3.2.

In the case study, the designed solution had to work in a Fiber To The Home (FTTH)

network that is composed by a set of specific devices. In a FTTH network, the optical fiber reaches the boundary of the living space, such as a box on the outside wall of a home. In these networks, there are some passive elements, such as splitters or fibers, and active elements, such as Optical Network Terminal (ONT), Optical Line Termination (OLT) or ethernet routers. Figure 2.2 depicts a standard structure of an FTTH network. This network architecture usually delivers triple-play services directly from the central office of the operators. Furthermore, the final system should deal with devices from different vendors and different access protocols, which was an issue in the project.

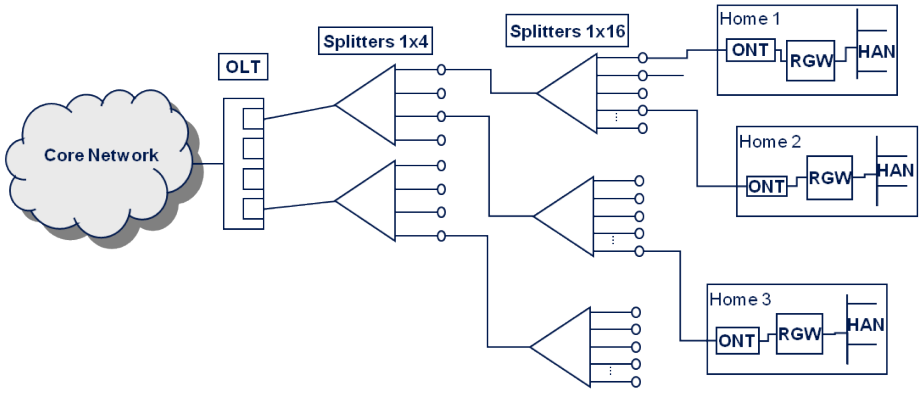

Figure 2.2: Architecture of an FTTH network.

2.3.2 MAS Behaviour Specification

This phase has the goal of architecturing the multi-agent system specifying all agent roles and the interaction among them. Based on the features identified in the previous phase, the agent features are realised with the MAS system. In order to maintain traceability and improve communication within the development team, we have found useful to use the same approach than in the previous phase for specifying the MAS behaviour. Thus, *business benefits* are described by *features* which are assigned to *agent roles*. The textual templates presented in Tables 2.1 and 2.2 are used by the MAS designer in this phase to create *Agent Stories*. These *Agent Stories* describe the expected behaviour of an agent given a context and a trigger event to achieve a specific goal. The described scenarios are translated into test cases in the following phase of the methodology. These scenarios must represent all agent behaviours.

As the proposed methodology is focused on acceptability testing, no restriction is imposed for designing the MAS using any design methodology. Therefore, methodologies, such as MAS-CommonKADS (Iglesias et al., 1998), Ingenias (Pavon et al., 2005), Prometheus (Padgham

and Winikoff, 2003), or Gaia (Wooldridge et al., 2000), can be used to design and/or develop the MAS. In other words, the proposed methodology is an agile acceptance testing methodology to ensure the communication among stakeholders and developers and no design or implementation restrictions are imposed. So, the MAS designer has the responsibility to translate *User Stories* to *Agent Stories* describing all agent roles in the system and all their behaviours using any MAS design methodology. Those *Agent Stories* are used to test all agents of the system.

As previously, *features* can be obtained in different contexts which are described as scenarios, which can involve one or more agent roles in the case of *cooperative scenarios*. In the case of *emergent features* coming from global behaviour, they will be only verified when the full system has been developed. This kind of emergent behaviour will be specified at MAS level in the agent stories, instead of for a particular agent role.

The final MAS of the case study was too complex to be shown in this section, so a simplified scenario is presented to exemplify the use of the methodology. We are going to focus only on the *Agent Stories* exposed in Table 2.4. These *Agent Stories* define the behaviour of a *Diagnosis Agent* that must be able to diagnose the root cause of fault and it is directly related with the *User Story* shown in the previous section in Table 2.3.

In Table 2.4, two *Agent Stories* are shown. Every story has two scenarios to exemplify the requirement defined in the story. The first story (lines 1-4) defines the goal that the agent has to perform a diagnostic process for the devices under its supervision. The associated scenarios (lines 6-11 and 13-16) exemplify that requirement for specific diagnosis symptoms. The second story (lines 18-21) defines the goal of fulfilling the conditions contracted with the customer in the Service Level Agreement (SLA). Its associated scenarios (lines 23-29 and 31-37) specify the goal of satisfying a concrete time restriction contracted with the customer and trying to minimise the diagnosis time.

2.3.3 Agent Level Testing

This phase has the aim of testing all agents individually providing traceability to the *User Story* they are trying to implement. Based on the requirements obtained during the previous phases, agents are designed and developed. As mentioned previously, any of the existing MAS design methodologies can be used for modelling and implementing agents. This methodology is focused on testing aspects while the whole system is being developed, considering that many different agents are being developed in parallel. Then, any test case where several agents should interact among them could not be performed until all of them

```
Story: Diagnosis process triggered by a symptom
As a Diagnosis Agent,
I want to process a FIPA-INFORM message with a detected symptom,
So that the system under my supervision is diagnosed as soon as possible.
******
Scenario: Diagnosis Agent diagnoses Damaged Splitter
Given a VoD streaming session,
When a 'high loss rate' symptom is received from a Probe Agent
And two or more geographically close users have loss rate higher to 1%,
Then the Diagnosis Agent must infer that the root cause of the
  problem is 'Damaged Splitter'.
------
Scenario: Diagnosis Agent performs bandwidth tests
Given a VoD streaming session,
When a 'jittering' symptom is received from a Probe Agent,
Then the Diagnosis Agent must perform bandwidth test to know its usage rate.

Story: SLA fulfilment
As a Diagnosis Agent,
I want to report issue status before a given deadline,
So that I achieve my goal of fulfilling SLA restrictions.
******
Scenario: Diagnosis Agent meets the SLA
Given a SLA is contracted with a customer
And the Diagnosis Agent is aware of SLA commitments,
When that customer is current on payments
And any diagnosis is in progress,
Then the Diagnosis Agent must give a response in time that fulfils the SLA
  time restrictions.
------
Scenario: Diagnosis Agent performs fast diagnosis
Given a SLA is contracted with a customer
And the Diagnosis Agent is aware of SLA commitments,
When that customer is current on payments
And any diagnosis is in progress,
Then the Diagnosis Agent must finish the network tests as soon as possible
  to minimise the diagnosis time.
```

Table 2.4: Examples of Agent Stories.

would be implemented. For that reason, we propose the use of Mock Agents to emulate non-available agents in this testing phase, as explained below. So, this phase has two main steps (see Figure 2.3): (i) developing mocks of the agents (for those not developed yet) and external systems that an agent interacts with and (ii) developing the unit tests of every agent[2].

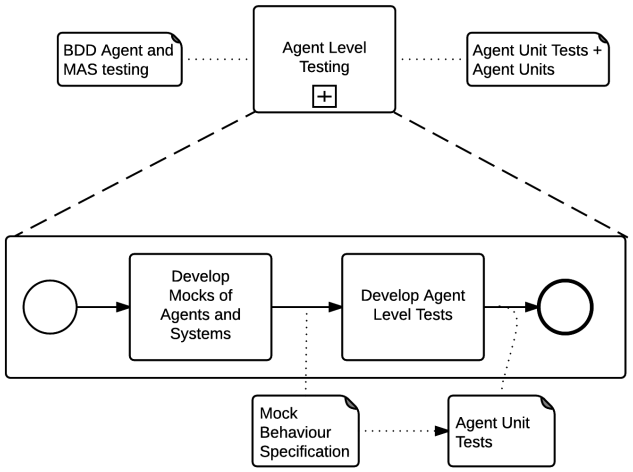

Figure 2.3: Steps of the Agent Level Testing Phase.

The first step requires to emulate or simulate the expected behaviour of the agents or external systems according to the scenarios from the *Agent Stories* developing mock agents. The second phase implements the tests configuring those developed mocks. There have been several research works developing the concept of using mock testing for agent unit testing. Coelho et al. (Coelho et al., 2006) proposed a framework for unit testing of MAS based on the use of Mock Agents on top of the multiagent framework JADE (Bellifemine et al., 2007). They proposed to develop one Mock Agent per interacting agent role. Mock Agents were programmed using script-based plans which collect the messages that should be interchanged in the testing scenarios. Tiryaki et al. (Tiryaki et al., 2007) proposed the framework *SUnit* on top of the multiagent framework *Seagent* (Dikenelli et al., 2005). They extended *JUnit* testing in order to cope with agent plan structures testing. Zhang (Zhang et al., 2011) generated automatically Mock Agents from design diagrams developed within the Prometheus Methodology (Padgham and Winikoff, 2003).

As we are interested in simulating the behaviour of agents, we have defined several types of Mock Agents which provide a simple FIPA interface. Three basic mock patterns have been defined: mock that simulates answering messages (*ResponderMockAgent*), mock that

[2]Notice that an agent that is being tested is denoted as Agent Under Test (AUT)

simulates receiving messages without providing an answer (*ListenerMockAgent*) and mock that receives a message from one agent and sends a new message to a different agent (*MediatorMockAgent*). Figure 2.4 depicts the interaction among the proposed Mock Agents, the Test Case and the Agent Under Test (AUT). The *ResponderMockAgent* has been designed to reply incoming messages with predetermined ones. This can be used to simulate external services or agents which have to connect to those services. So, an AUT can interact with this type of Mock Agent to get external information in the same way as the final MAS. The *MediatorMockAgent* has been designed to act as a filter of messages. In other words, this Mock Agent receives messages from an AUT and sends a different message to other AUT. So, this type of Mock Agent can be used as processes that have to perform some actions with the information enveloped in the first message and have to inform to another agent or system. Finally, the *ListenerMockAgent* has been designed as a mailbox. This Mock Agent can be used to check if the content of a message sent by an AUT.

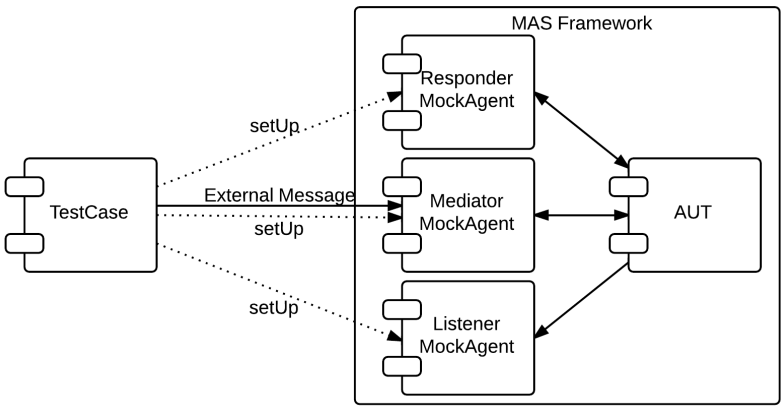

Figure 2.4: BEAST Mock Agents.

The proposed types of agents cover the most general communications among agents in a MAS. However, other Mock Agents can be designed if it is a need for the project requirements. For example, the proposed *ResponderMockAgent* could be modified to query a database to reply the message with real data that would be used by the AUT.

The solution designed for one of the scenarios of *Agent Stories* exposed in the previous section is shown in Figure 2.5 to exemplify how to test an agent in a specific scenario, included in the Table 2.5. There are three types of agents: a *Probe Agent* responsible for monitoring Video On Demand (VoD) sessions, an *Expert Agent* to collect metrics from other subscriber lines and the *Diagnosis Agent* to perform the diagnosis process itself. In the first iteration of the case study project, the *Probe* agent and the *Expert* agents had not been developed yet. Thus, the mocking facility of the proposed BEAST Methodology was used. In this

case, the *Mediator Mock Agent* is suitable for simulating *Probe Agent* to send symptoms to the *Diagnosis Agent* and the *Responder Mock Agent* is suitable for simulating *Expert* agents. Thus, these mocks are configured for sending symptoms and network information respectively to simulate both agent roles. Finally, the *Diagnosis Agent* is the Agent Under Test (AUT) for this scenario. The testing scenario starts when the *Probe Mock Agent* sends a message to the *Diagnosis Agent* (AUT). Then, the *Diagnosis Agent* requests information about the status of other subscriber lines. Finally, the tester checks if the AUT has got the right hypothesis of fault root cause. The implementation of this scenario as BEAST Test Case is exposed in Section 2.4.5 after the reminder features of the BEAST Tool are explained. To illustrate the explanation of this scenario, Figure 2.6 represents the interaction of the BEAST Test Case with the Mock Agents and the AUT in a sequence diagram.

```
1 Scenario: Diagnosis Agent diagnoses Damaged Splitter
2 Given a VoD streaming session,
3 When a 'high loss rate' symptom is received from a Probe Agent
4 And two or more geographically close users have loss rate higher to 1%,
5 Then the Diagnosis Agent must infer that the root cause of the
6   problem is 'Damaged Splitter'.
```

Table 2.5: Exemplified Scenario of an Agent Story.

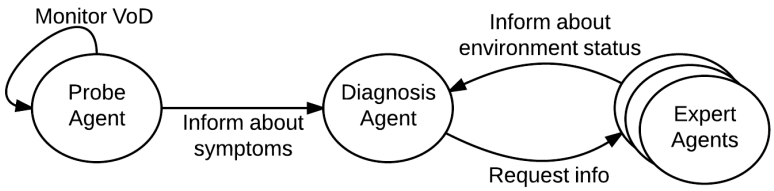

Figure 2.5: Overview of agents involved in the exemplified scenario.

As SCRUM Agile Methodology (Schwaber and Sutherland, 2009) had been chosen to manage the project progress, the result of the executable requirements are shown to the stakeholder (*Product Owner* in SCRUM terminology) periodically during progress review meeting (*Sprint Reviews* in SCRUM terminology). This helps to the *Product Owner* to know the status of the project and to modify the *User Stories* to represent better the idea of the stakeholder that is not always well translated in the initial *User Stories*. Furthermore, the traceability from a *User Story* to a test case makes easy to know which features or what test cases must be modified to fits the updated requirements.

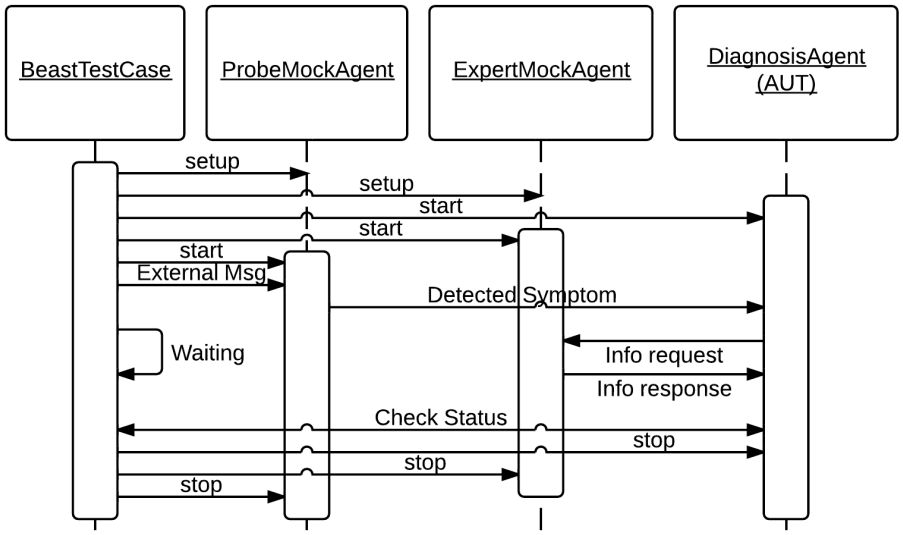

Figure 2.6: Steps of the exemplified scenario.

Traceability from user requirements to executable tests is one of the keys of success in any software project (Almeida et al., 2007). To ensure this traceability, the BEAST Methodology proposes a set of mapping rules that connect the outcomes of all phases of the methodology. Figure 2.7 shows that *User Stories* obtained in the *System Behaviour Specification* phase are used as input in the *MAS Behaviour Specification* phase. In this phase, a *User Story* is translated to one or more *Agent Stories*. Both *User Stories* and *Agent Stories* follow the same template format (see Tables 2.1 and 2.2). Finally, the scenarios of the *Agent Stories* are implemented to test the developed MAS. Thus, a *User Story* is broke down in *Agent Stories*. An *Agent Story* is composed by a set of scenarios that are implemented as test cases. So, the stakeholders know automatically which requirements are not fulfilled when a test fails.

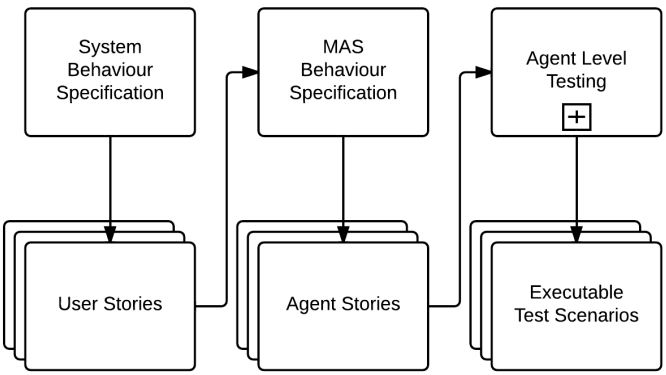

Figure 2.7: Outcomes of BEAST phases.

Following the *JUnit* framework, both *User* and *Agent Stories* can be tested at once using *TestSuites* to execute all test cases related with it. A *TestSuite* is a collection of test cases to show if a software has a specified behaviour. So, the set of scenarios which compose a story are joint in a *TestSuite* to check if a story feature is satisfied. Figure 2.8 shows an example of traceability in the BEAST Methodology.

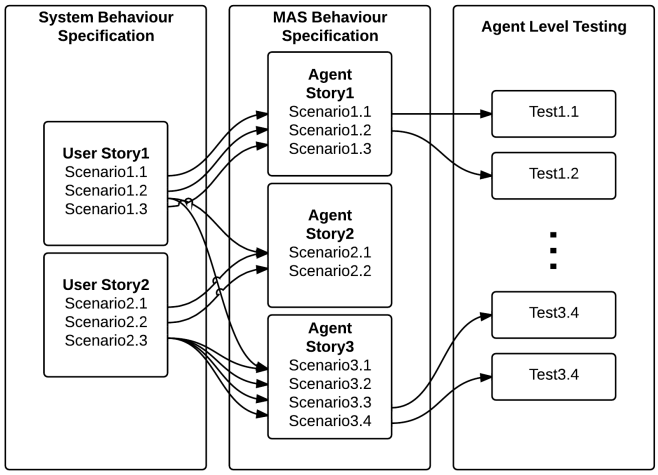

Figure 2.8: Example of traceability in the BEAST Methodology.

This traceability simplifies the global view of the project status. An example of traceability from *User Story* to test case is shown in Figure 2.9 with the result message of a failing *User Story*. The message of that failing test says what specific scenario of the story is failing to find exactly what test case must be reviewed. That information is useful for stakeholders, designers and developers to know in a look the status of the project and take decision accordingly.

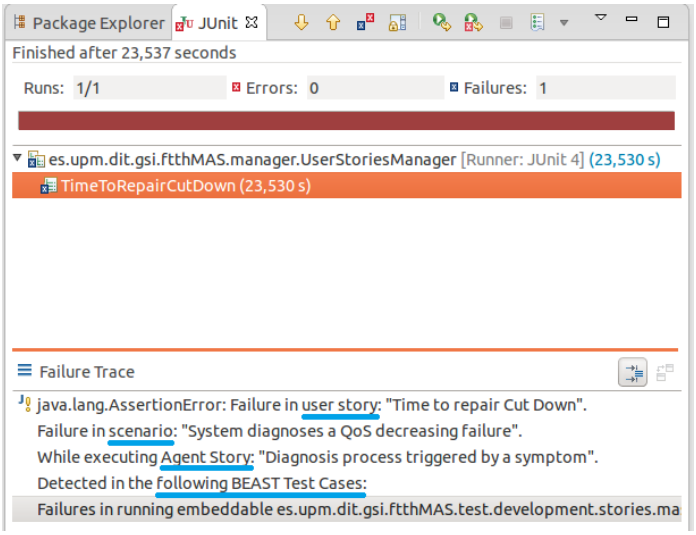

Figure 2.9: Screenshot of a failing BEAST Test Case.

2.3.4 MAS Level Testing

This phase has the aim of evaluating the complete systems offering metrics to measure the expected outcomes of the system. When all agents that compose the MAS have been developed and tested individually in the previous phases, integration testing is performed to ensure all agents work together as expected and, later, a set of metrics is collected to evaluate the behaviour of the developed system in a testing environment. Thus, the *MAS Level Testing* phase is divided in two different sub-phases: *Integration Phase* and *MAS Validation Phase*, as shown in Figure 2.10.

The *Integration Phase* has the aim of validating the behaviour of all agents working together. As soon as two different agents have been developed, integration tests can start using the same test case extracted from an *Agent Story* replacing the Mock Agents with the developed agents. However, these integration tests among agents cannot ensure the expected behaviour of the MAS executing in the final environment. In other words, the interaction between the agents and the environment in long term conditions are, generally, difficult to be measured using unit test cases. Thus, the evaluation of the system in a complete testing environment is required to validate the developed MAS.

Therefore, the second sub-phase of this *MAS Level Testing* phase is focused on those features that cannot be tested in a unit test case and require a complete testing environment, which replicate or emulate the production environment, to evaluate the global behaviour of the developed system under specific and general conditions. To measure that system behaviour, some metrics are defined based on the *User Stories*, requested by the stakeholders,

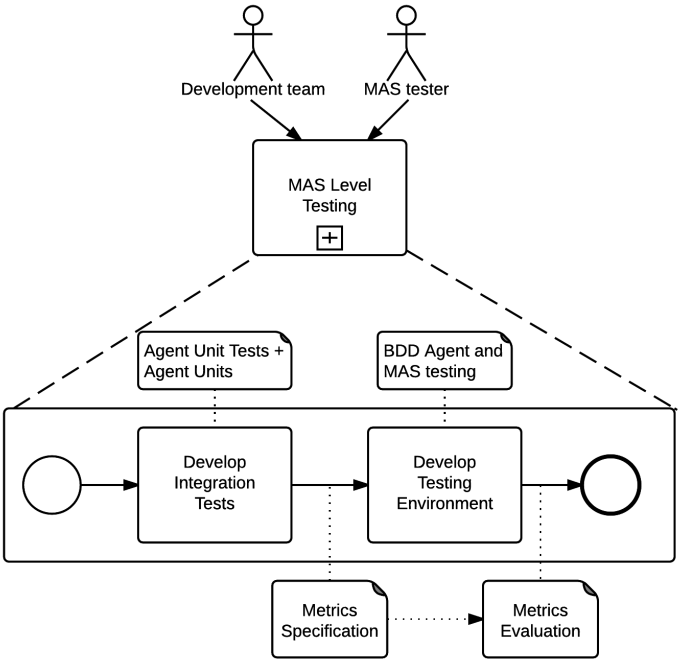

Figure 2.10: MAS Level Testing.

and the *Agent Stories*, requested by the MAS designer. Those metrics are collected during the execution of the system in the testing environment and analysed to know if desired requirements are met and to evaluate the validity of the designed solution. But, the definition of these metrics is not restricted only to the stories that define the expected behaviour of an agent or the system. Other non-functional features, such as effort balance or time metrics, can be measured to analyse the system behaviour and, consequently, the impact caused by the MAS when is deployed in the production environment.

We propose a framework for metrics definition based on their typology. There are two main types of metrics, the ones which are defined to measure the outcome expected by the stakeholders (**Requirement** metrics) and the ones which are defined to measure the behaviour of the developed systems and evaluate its design (**Design** metrics). Hence, we associate *Requirement* metrics with *User Stories* and *Design* metrics, with *Agent Stories*.

A *Requirement* metric can be classified as **Functional** or **Non-Functional**, based on the requirement that is being measured. For example, based on the *User Story* shown in Table 2.3, we can define some metrics about the number of correct diagnosis, the correction of detected symptoms as *Functional Requirement* metric, and metrics about the time to diagnose or the amount of resources required to carry out the diagnosis process as *Non-Functional Requirement* metrics.

Alternatively, *Design* metrics are classified by their **Granularity** and their **Domain**. *Granularity* is the extent to which a system is composed of distinguishable pieces. Thus, we propose two levels in this dimension: **Macro** and **Micro**. We say the granularity of a metric is defined as *Macro* when it observes behaviours of the global system, i.e. the MAS. Contrarily, when a metric is focused in a single agent, its granularity is defined as *Micro*.

Finally, some metrics can be applied to measure some features of any MAS. The *Domain* dimension of those metrics are defined as **Generic**. In contrast, other metrics are defined specifically to observe a behaviour of a system or an agent in a concrete environment or situation. Therefore, they are classified as **Specific**.

A set of the metrics for the case study project is defined to exemplify the use of this metrics framework in the development of a fault diagnosis system. First of all, a set of *Requirement* metrics are defined in Table 2.6, both *Functional* and *Non-Functional* (shown in column *Type* as **F** and **NF** respectively). These metrics are directly extracted from *User Stories* and, consequently, they do not include any reference about the solution designed. Only the concept of system is mentioned, but not the concept of agent.

A set of *Design* metrics is proposed to evaluate the validity of the designed MAS. These metrics are defined from the *Agent Stories*. Table 2.7 proposes a set *Design* metrics to evaluate the behaviour of the proposed MAS solution for the FTTH scenario exposed in the previous sections. Both dimensions, *Granularity* and *Domain*, are shown in column *Type* for the proposed metrics as *Macro* (**M**) or *Micro* (**m**), and as *Generic* (**G**) or *Specific* (**S**). Klügl (2008) proposes a set of metrics for measuring complexity of multi-agent systems. Thus, the *Generic Macro* (G/M) metrics exposed below are based on that work.

After a number of iterations of the case study project, the developed MAS is ready to execute in a testing environment. Then, the metrics defined previously are collected during the execution of the developed solution in the testing environment. After an analysis of the collected data, the metrics are shown to the stakeholders in charts and/or diagrams to expose the measured results of the system. All these metrics have been collected in a simulated environment where 3500 diagnosis cases where generated.

The number of completed diagnoses and their final states are shown in Figure 2.11 (M1 - Number of Completed Diagnoses.) where we can observe that only a few cases finishes with a warning status. We can observe that a 3.9% of the diagnosis processes finished with a *Warning* message, but zero cases finished with a fatal error.

But not only the final status is important, the correctness of the symptom detection process is a key factor to improve the efficiency of the fault diagnosis system. This metric

Metric ID	**Metric Name**	**Type**	**Description**
M1	Number of Completed Diagnoses	F	Measure of the success of the diagnosis processes properly performed by the system.
M2	Correct Symptom Detection	F	Measure of the correct detection of symptoms.
M3	Correct Diagnosis Conclusions	F	Measure of the processes which diagnoses properly a detected symptom.
M4	Heterogeneity of Diagnosis Cases	NF	Measure of the diversity of the faults diagnosed by the system.
M5	Time To Diagnose	NF	Measure of the time between a symptom is detected and a conclusion is reached by the system.
M6	Spectrometer Usage Rate	NF	Measure of the usage of a critical device that avoid the regular usage of the transmission line.

Table 2.6: Requirement Metrics for an autonomic Fault Diagnosis system.

Metric ID	Metric Name	Type	Description
M7	Agents Population	G/M	Measure of the heterogeneity of the MAS by number and type of agent.
M8	Available Resources	G/M	Measure of the heterogeneity of the systems, devices or services which the agents have to interact with.
M9	Global Efficiency Rate	S/M	Measure of the cost of a success diagnosis process based on the consumed resources associating a cost to every resource.
M10	Number of Messages per Diagnosis Case	S/M	Measure of the messages generated by the agents per diagnosis cases.
M11	Number of Sent Messages	G/m	Measure of the messages sent by one agent.
M12	Number of Received Messages	G/m	Measure of the messages received by one agent.
M13	Time To Reason	S/m	Measure of the time to infer a conclusion of a detected fault by a Diagnosis Agent.
M14	Time To Test	S/m	Measure of the time to perform a specific test to collect information from the network by an Expert Agent.
M15	Time to Detect	S/m	Measure of the time between a symptom appears and it is detected by a Probe Agent.

Table 2.7: Design Metrics for an autonomic Fault Diagnosis system.

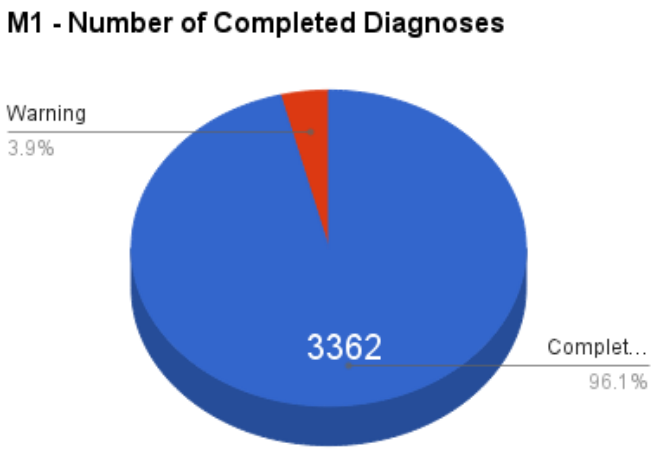

Figure 2.11: M1 - Number of Completed Diagnoses.

is shown in Figure 2.12 (M2 - Correct Symptom Detection.). It shows the number of well detected symptom (60.4% of *true positives*), bad detected symptoms (6.7% of *false positives*), non-detected symptoms (2.8% of *false negatives*) and non-false alarms (30.1% of *true negatives*).

In the presented case study, there are nine different fault root causes under consideration. The correctness of the diagnosis conclusions[3] for the simulations executed as testing environment is shown in Figure 2.13 (M3 - Correct Diagnosis Conclusions.).

Among all those cases, a fault root cause shows a different set of symptoms. We measure that fact with the entropy value. Thus, Figure 2.14 (M4 - Heterogeneity of Diagnosis Cases.) shows that some root causes have entropy values close to zero, because these fault types almost always present the same symptoms. In contrast, other fault root causes exhibit high entropy because these fault types can be manifested as different symptoms and test results. In other words, M4 is a measure of the complexity of every type of fault.

The time required by the system between a symptom is detected and a fault root cause is offered as conclusion is shown in Figure 2.15 (M5 - Time To Diagnose.). This a classical metric for any fault diagnosis process, either automatic or manual, known as Time To Diagnose (TTD); as exposed by FitzGerald and Dennis (2008). Other two interesting metrics

[3]We are considering the regular status of the network as a valid diagnosis conclusion, i.e. no problem diagnosed. This is to handle false positive cases in the symptom detection process.

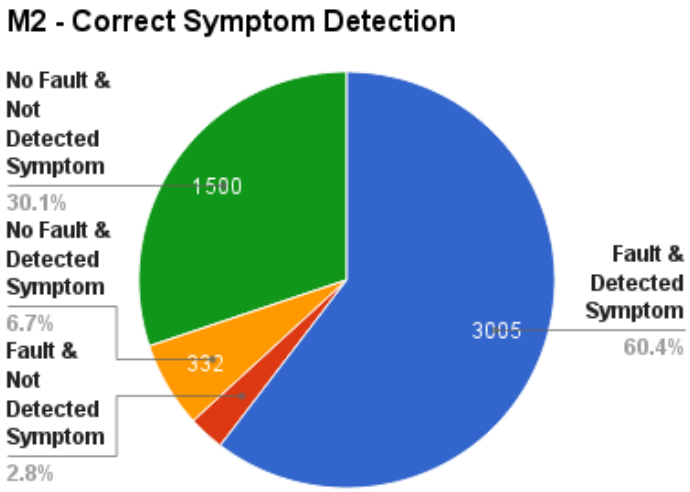

Figure 2.12: M2 - Correct Symptom Detection.

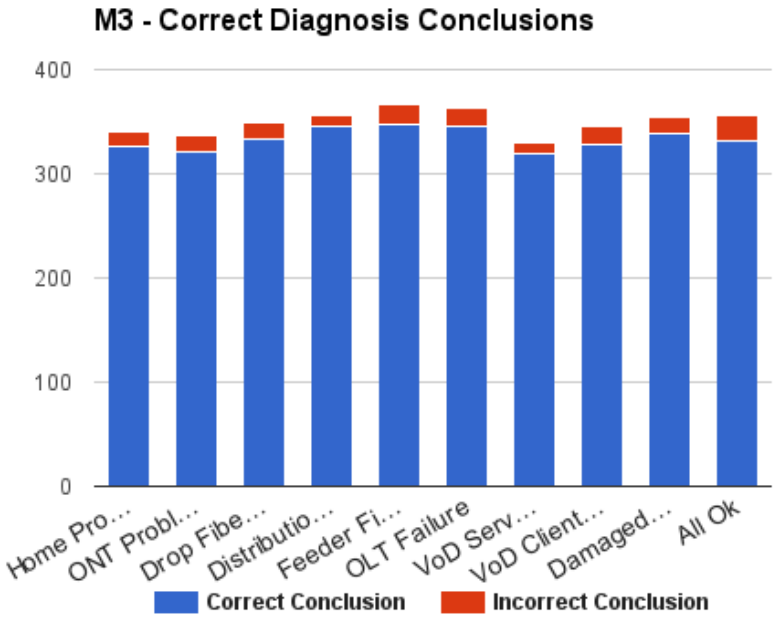

Figure 2.13: M3 - Correct Diagnosis Conclusions.

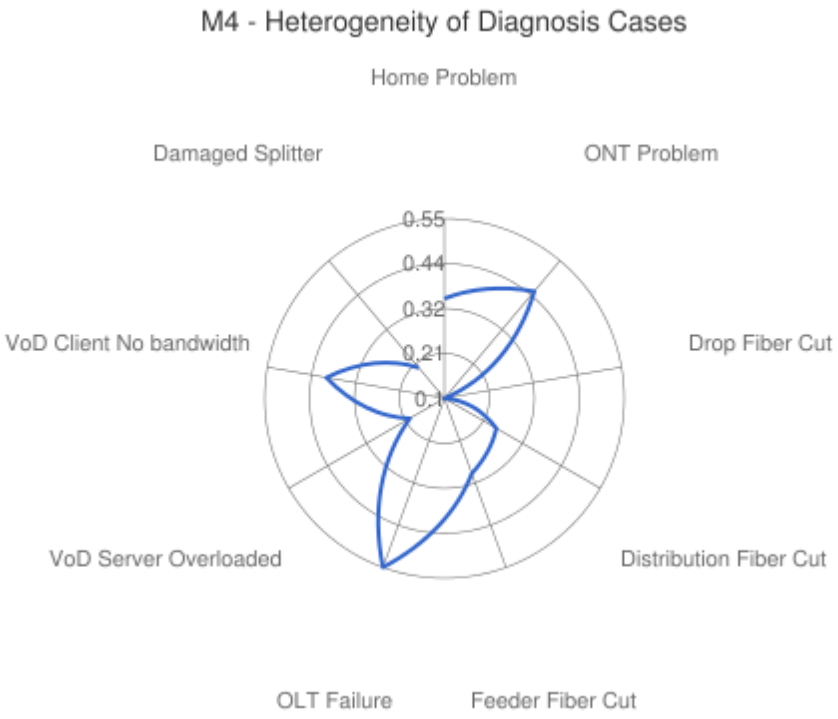

Figure 2.14: M4 - Heterogeneity of Diagnosis Cases.

for this time measurement are the mean μ and the standard deviation σ of the TTD. In the testing environment, the value of them were $\mu = 33.357$ seconds and $\sigma = 8.188$ seconds.

To conclude with the *Requirement* metrics, the usage of a critical device, such as a spectrometer, is quantified in Figure 2.16 (M6 - Spectrometer Usage Rate.). Notice that the spectrometer is a critical resource, because the final client/user of the diagnosed line cannot use their connection while the spectrometer is measuring. Thus, the impact of using this resource during the diagnosis process is quite high. Thus, thanks to M6, the stakeholder can observe that the spectrometer is only used in a few diagnosis cases (2.8%). The mean time while the spectrometer was being used during a diagnosis case was 24.31 seconds (with a standard deviation of 2.99 seconds).

So, thanks to these *Requirement* metrics, the stakeholders can decide if the status of the system is stable enough or further development and refinement is required. With this information, grounded decisions can be made. For instance, if they decide that the number of *Warning* cases shown in M1 (Figure 2.11) is too high, they should consider than the usage of the spectrometer could increase in further iterations to reduce that percentage of warning messages.

Focusing on *Design* metrics, the reminder charts and diagrams are focused on designers.

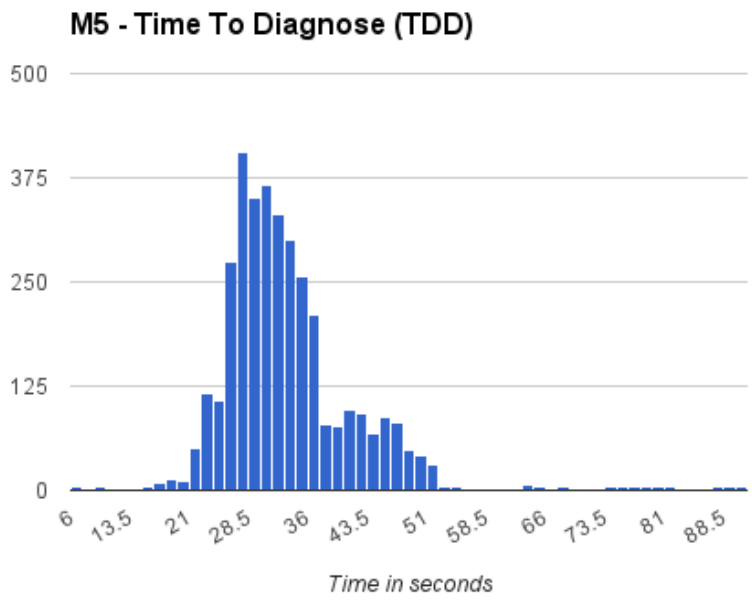

Figure 2.15: M5 - Time To Diagnose.

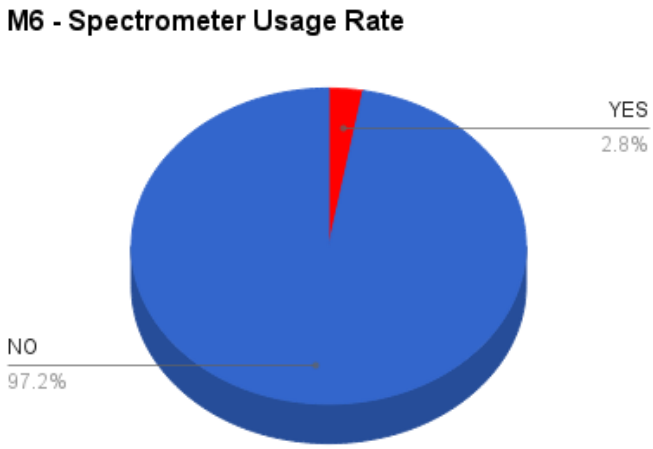

Figure 2.16: M6 - Spectrometer Usage Rate.

But, some interesting data can be shown to stakeholders to explain or justify design decisions. Figure 2.17 (M7 - Agents Population.) shows the number of agents running in the solution presented in this case study, offering a view of the heterogeneity of the agents roles required to carry out the diagnosis process. Thus, we have one Diagnosis Agent, one Probe Agent and four Expert Agents for this case study.

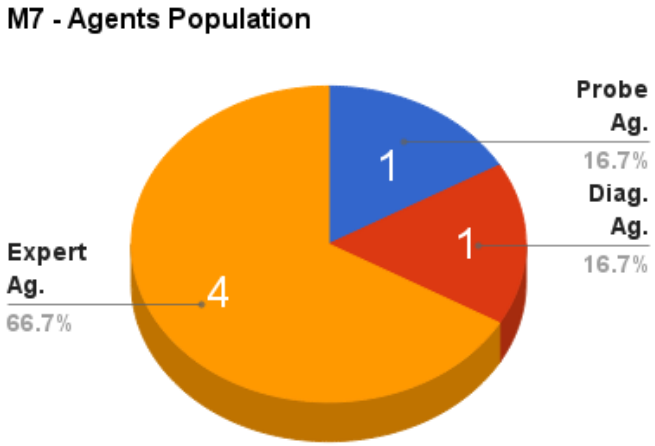

Figure 2.17: M7 - Agents Population.

The available resources for this Fault Diagnosis scenario are shown in Figure 2.18 (M8 - Available Resources.). As we are considering a simplified scenario for this case study, the devices shown in this figure are for a single client connection. Thus, we find one VoD client, one VoD server, one Rourter GW, one ONT, two Splitters, one OLT, one Spectrometer and two different Databases where some context information is stored and can be retrieved by the agents to complete the diagnosis process.

Associating a specific cost to every resource, other interesting measure of the behaviour of the system is how efficiently those devices and/or systems are being used while the system is diagnosing. Thus, Figure 2.19 (M9 - Global Efficiency Rate.) shows an histogram of a set of 3500 simulated diagnosis cases assigning the following costs to every resource[4]: access to Data Base (1), check ONT(1), check OLT (1), check Router GW connectivity (1), check VoD server (2), check VoD client (2) and usage of spectrometer (7)[5]. The metric shows a

[4]Splitters are passive elements which are used in any connection to the Home Area Network (HAN) but no cost is associated to them because no action can be performed over them.

[5]This high cost for the spectrometer is based on its critical nature as mentioned previously

Figure 2.18: M8 - Available Resources.

coherent resource consumption in a high percentage of cases and only a few with a high consumption, which is an acceptable behaviour.

The number of messages generated in a MAS can offer an idea of the complexity of the processes carried out by their agents. Thus, Figure 2.20 (M10 - Number of Messages per Diagnosis.) shows the messages generated by all agents in a single diagnosis case. Based on the resources status, some cases require more messages than others. For instance, depending on the load of a server, some tests could not be performed and new requests could be generated or an alternative should be found using some discovery techniques, such as directory facilitators queries or broadcast requests.

Focusing on single agents, we are considering some micro metrics for the presented case study. Figures 2.21 (M11 - Number of Sent Messages.) and 2.22 (M12 - Number of Received Messages.) show the number of messages sent and received, respectively, by each agent of the MAS. Those data have processed to show the minimum and maximum values highlighting the mean and the standard deviation of the data series. Analysing these message metrics, we can observe than the existence of only one diagnosis agent overload it with a number of messages higher than the reminder agents. This could be an scalability issue in large-scale scenarios. A multi-role agent approach could be explored to solve this problem, but it would increase the development time and the complexity of the solution. Anyway, for the FTTH scenario under consideration, this solution design offers satisfactory results.

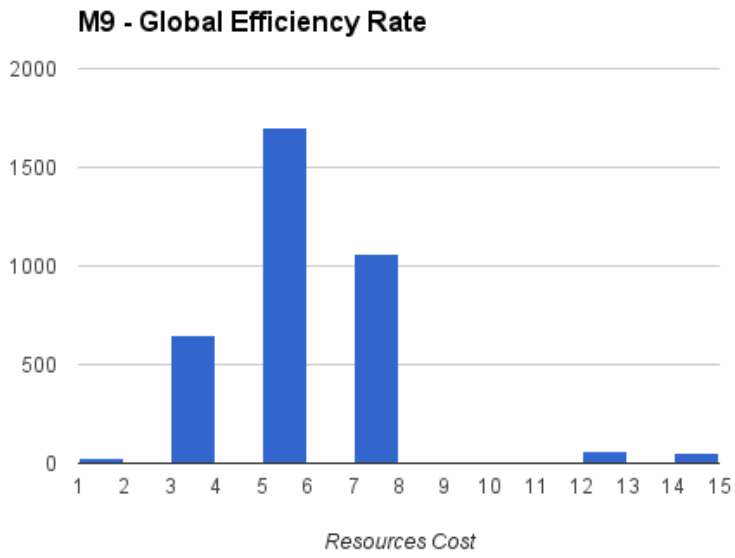

Figure 2.19: M9 - Global Efficiency Rate.

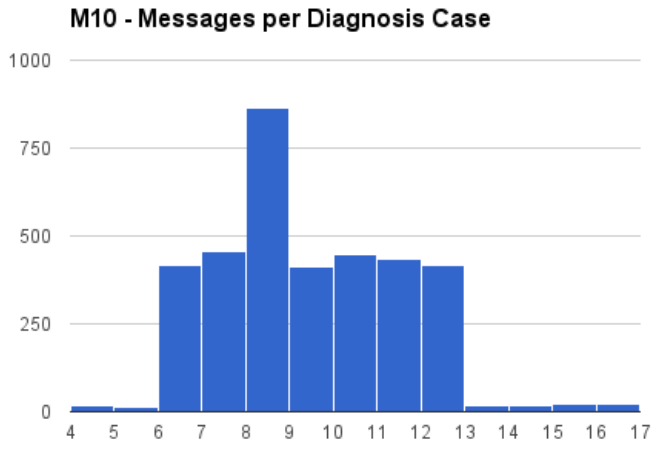

Figure 2.20: M10 - Number of Messages per Diagnosis.

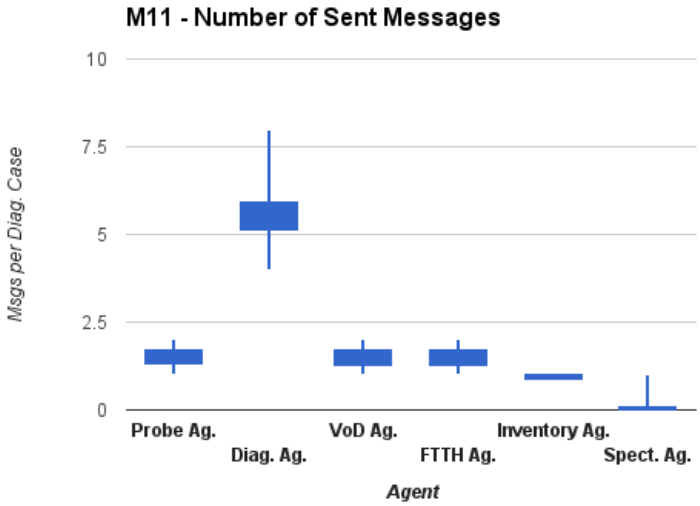

Figure 2.21: M11 - Number of Sent Messages.

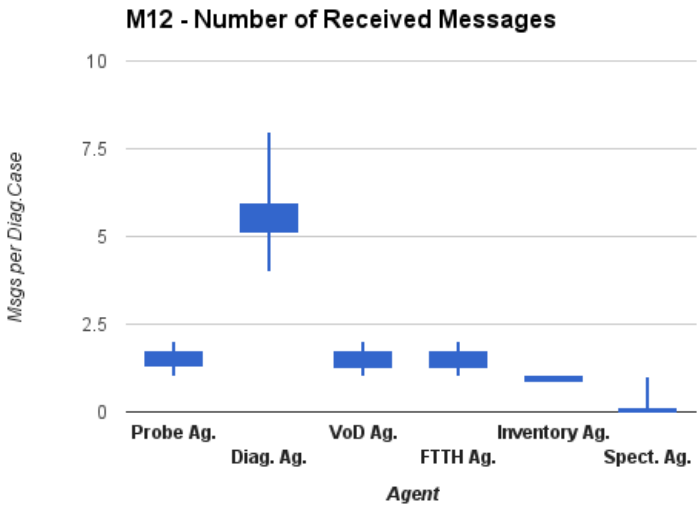

Figure 2.22: M12 - Number of Received Messages.

Focusing on the behaviour of every individual agent, we can measure the time spent by every of them to carry out their goals. Figure 2.23 (M13 - Time To Reason.) shows the time used to reason by the *Diagnosis Agent* to infer the possible causes of the fault. We can find that the reasoning time is always below 2 seconds.

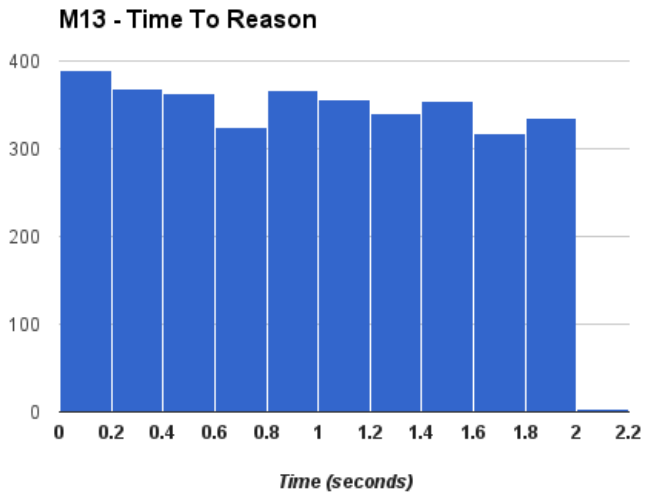

Figure 2.23: M13 - Time To Reason.

Other interesting point is the time spent by *Expert Agents* to perform tests and retrieve information from the network, as shown in Figure 2.24 (M14 - Time To Test.). As can be seen, the metrics for two agents (*VoD Agent* and *Spectrometer Agent*) has their mean and standard deviation close to zero. That is because they are two agents that perform tests with high costs (see Figure 2.13) and the system tries to reduce the execution of those tests.

Finally, as a quick detection of symptoms is crucial for any diagnosis, Figure 2.25 (M15 - Time to Detect.) shows the time spent by the *Probe Agent* to detect a generated symptom in the testing environment. This metric shows the time since the symptom is artificially generated until the symptom is notified to the *Diagnosis Agent*. That time varies based on the severity of the problem. For instance, a *fiber cut* problem is detected quickly (i.e. the connection is lost), but a *VoD server overload* could be more difficult to detect (i.e. the video streaming has some micro-cuts but it seems that works).

To conclude with the analysis of the presented metrics, some final remarks can be highlighted after the evaluation of the system. On the one hand, the shown metrics reveal possible improvements in the system both in performance and outcomes. But, on the other hand, those improvements could overload the resources that the system uses to carry out its objectives. The design of the solution presents some potential issues in large-scale sce-

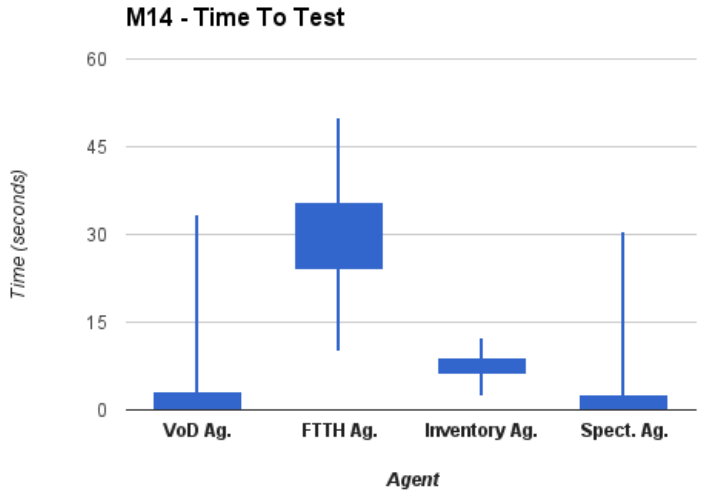

Figure 2.24: M14 - Time To Test.

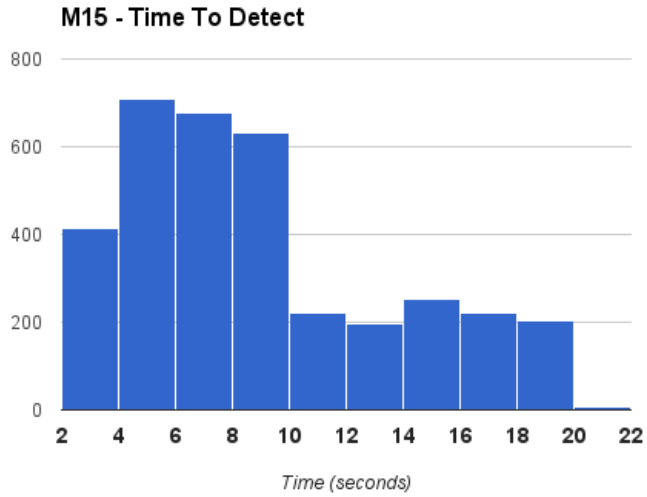

Figure 2.25: M15 - Time to Detect.

narios, but it is suitable for the problem under consideration. If any of the detected issues is considered as a design problem or stakeholders are not satisfied with the shown results, the MAS should be redesigned in the following iterations of the project which would have a direct impact on the *Agent Stories*.

2.4 BEAST Tool

This section exposes some features of the open source tool developed to support the BEAST Methodology detailed in the previous section, named BEAST Tool, which is hosted in a *Github* repository[6]. This tool has the aim of providing assistance in the application of the BEAST Methodology. The tool provides parsing facilities to support the translation of requirements to test cases, both in *System Behaviour Specification* phase and *MAS Behaviour Specification* phase. The tool translates story and scenario templates (see Tables 2.1 and 2.2 respectively) into Java templates. The tool provides a BEAST Test Case model which is used in the *Agent Level Testing* phase. This test case model is integrated with an extended version of JBehave framework (North, 2011), which is a framework that supports the execution of BDD scenarios. The implemented extension of that framework allows the execution of agents running in MAS platforms in a executable scenario test. Focusing on those platforms, one of the design principles of BEAST Tool has been that it must be valid for different MAS platforms. By now, the current version of the tool supports JADEX (Braubach et al., 2005) and JADE (Bellifemine et al., 2007) frameworks, but it can be extended to other frameworks with low effort as it is explained below in Section 2.4.3. Finally, the tool provides mocking facilities which allow to configure easily some Mock Agents extending the Mockito framework (Mockito Project, 2012).

The rest of this section is structured as follows. Section 2.4.1 exposes a reader (or parser) package which is used to manage the translation of user and agent stories to Java code. Section 2.4.2 explains the BEAST Test Case model. The adaptation of the tool to the MAS platforms is defined in Section 2.4.3. The use of mock agents is exposed in Section 2.4.4. The implementation of a specific test case is shown in Section 2.4.5. Finally, the impact of the application of this tool for the implementation of a set of tests for JADE and JADEX agents is shown in Section 2.4.6.

[6]BEAST Tool is hosted in the following public repository: http://github.com/gsi-upm/BeastTool/

2.4.1 Story Parser

This section presents the parsing capability of the tool to provide an automatic generation of test cases skeletons based on the text plain requirements. There are two different types of stories in the proposed methodology (Section 2.3): *User Story* and *Agent Story*. Following the BEAST Methodology, the first step is the *System Behaviour Specification* when the stakeholders and the development team (or at least one or two people of the development team) defines a set of *User Stories* in BDD format (see Tables 2.1 and 2.2). Then, these *User Stories* are processed with the parser included in BEAST Tool and a *TestSuite* is created for every *User Story*.

After *User Stories* are defined, the designer defines a set of *Agent Stories* that must fulfill the requirements specified in the *User Stories*. These *Agent Stories* contain the specification of all behaviours of any agent of the system. Then, the parser is used again to generate a new *TestSuite* per *Agent Story* and a set of BEAST Test Case templates (one per scenario). The parser is configured to generate BEAST Test Case templates or not depending on what type of story is being parsed (see Figure 2.26). In other words, the scenarios of an *Agent Story* are translated to a BEAST Test Case to implement them with agents. But, the scenarios of a *User Story* are translated to a *TestSuite* to be filled with other tests which implement those user requirements.

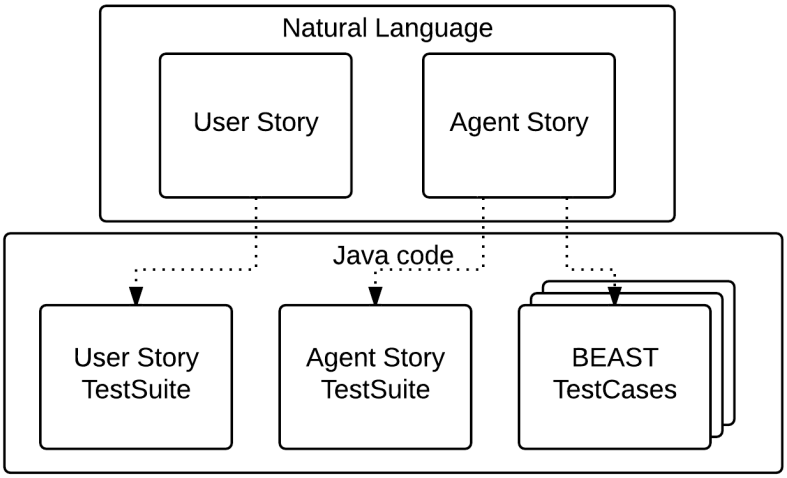

Figure 2.26: Java classes generated in the parsing process.

Defining which *Agent Story* is related with a *User Story* is a manual matching process which must be carried out by the designer. During this process, the designer must edit the *User Stories TestSuites* with the corresponding *Agent Stories TestSuites* or their specific scenarios. After this matching process, a *User Story* is completely traceable to the BEAST

Test Cases that implement a concrete test scenario. This allows stakeholders to execute requirements as test cases, getting *executable requirements* to have a quick look of the status of the project.

2.4.2 BEAST Test Case Model

This section explains how to translate the BDD specification of an agent story, in a *Given-when-then* template in Java source code to implement the test case. Our approach to agent level testing has consisted of extending JUnit framework in order to be able to test MAS systems. Mapping rules (Solis and Wang, 2011) have been defined in order to provide full traceability of acceptance tests defined previously in BDD format. Thus, *JBehave* has been extended with this purpose. In this framework, a *user story* is a file containing a set of *scenarios*. The name of the file is mapped onto a user story class. Each scenario step is mapped onto a test method using a Java annotation. In our case, BEAST Tool translates a scenario (see Table 2.2) to a test case class, termed BEAST Test Case, which extends *JUnitStory* class of *JBehave* framework and contains three key methods that directly related with the three parts of a scenario ("Given-When-Then").

The three methods that a tester must implement are depicted in Figure 2.27. The *setUp* method represents the "Given" scenario condition. This method typically initialises agents and configures their state (goals, beliefs, etc.) as well as initialises the environment. The *launch* method represents the "When" scenario condition. This method generates and schedules the trigger event to start the test. The *verify* method represents the "Then" scenario condition. The expected states, such as goals or beliefs, are checked in this method once test execution is over.

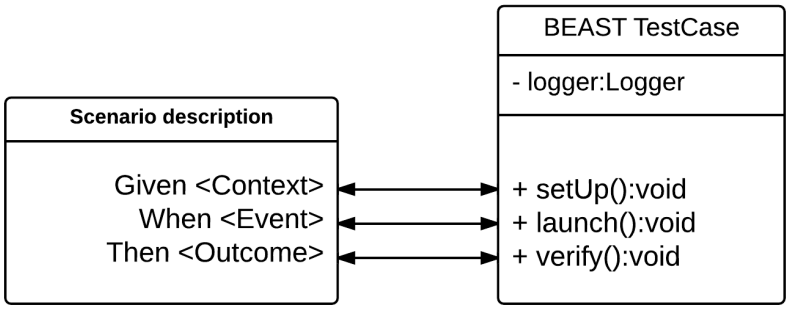

Figure 2.27: Relation between BEAST TestCase class and a BDD scenario.

BEAST Test Case has several methods that allows the interaction with a generic interface to interact with the MAS platform. This interface offers some facilities to prepare a concrete

state of the agent. For example, external messages can be sent to the MAS platform, agents can be started and stopped, or internal information of an Agent Under Test (AUT) can be configured, such as beliefs or goals. This generic MAS platform interface is shown in the following section.

2.4.3 MAS Platform Interface

This section explains how the BEAST Tool connects the test case with the MAS platform to start and manage agents under testing. To provide MAS platform independence, three different interfaces have been defined to interact with the MAS platform from a BEAST Test Case. Each of them is responsible of different aspects on the platform management. The first one, *Connector* interface, provides an abstract interface to agent managing functions, such as launching platform or starting agents. The second one, *Messenger* interface, declares methods for sending and receiving messages to or from the platform respectively. Finally, the third one, *Agent Introspector* interface, provides access to the agent model to deal with its beliefs, goals, etc.

To integrate BEAST Tool with any MAS platform, these three interfaces must be implemented to get access to their agents. In the current version of BEAST Tool, JADE 4.0 (Bellifemine et al., 2007) and JADEX 2.0 (Braubach et al., 2005) are completely integrated. To make easier the generation of BEAST Test Case classes, a testing interface selector has been defined, so called *PlatformSelector*. This selector provides the proper platform access from a BEAST Test Case to the MAS platform in which the system is being implemented as shown in Figure 2.28.

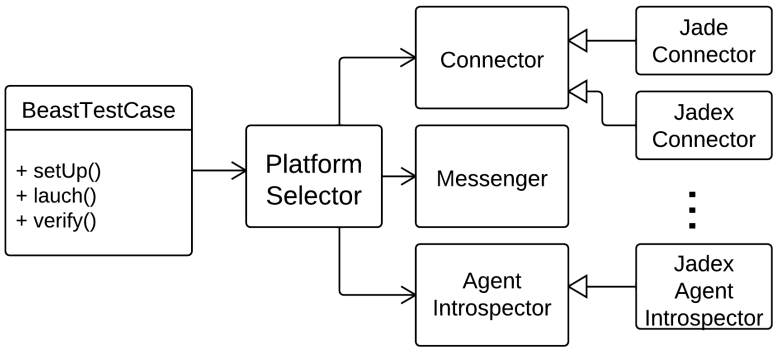

Figure 2.28: MAS Platform Selector for Beast Test Case.

2.4.4 Mock Definition

The BEAST Methodology proposes three basic mock patterns for messaging, as exposed in Section 2.3.3. BEAST Tool includes those patterns in three Mock Agents completely developed and ready to be used in any BEAST Test Case. As these agents have to execute in the MAS platform as any other agent, they have been implemented for JADE 4.0 and JADEX 2.0 with mock behaviours. So, we can say that a BEAST Mock Agent is an agent that contains mock plans and/or mock behaviours and can be configured and started from a BEAST Test Case.

After analysing available mocking frameworks, we selected *Mockito* (Mockito Project, 2012) framework, because of its easiness to be learnt, its popularity and its wide coverage of mocking functionalities. Thus, we have extended *Mockito* in order to be able to use it in MAS environments. In addition, the mocking framework allows an easy configuration of the mock objects (or agents), with patterns such as *when(< some input >).thenReturn(< some answer >)*. Mock Agents allow the specification of the emulated behaviour using *Mockito* constructions. Here follows an example.

```
when(mockAgent.processMessage(
  eq(``REQUEST''),
  eq(``Connection Loss Rate'')))
  .thenReturn(``INFORM'',``Loss Rate=0.2'');
```

Using this type of constructions, a tester can emulate the behaviour of Mock Agents as complex as required. The tester can consider a Mock Agent as a *black box*, i.e. the inputs and the outputs are known but the internal process is unknown. Thus, the use of Mock Agents is not restricted to the messaging. The BEAST Methodology presents only three Mock Agents for messaging (see Section 2.3.3) because those agents are completely generic models and can be used in any MAS development. But, the tester can implement other specific Mock Agents for a concrete project to make easier and faster its work as this agents can be configured with a few lines of code with different behaviours in a BEAST Test Case.

2.4.5 Implementing Test Cases

This section shows an example of implementing a specific test case. As exposed in Section 2.4.2, there are three methods that a tester must implement to implement a BEAST Test Case: *setup()*, *launch()* and *verify()*. Other methods, such as *startAgent*, *sendMessageToAgent* or *checkAgentsBeliefIsEqualTo*, are provided by the parent class (i.e. *BeastTTestCase*

class) that interacts with the MAS Platform interface to access to the agents, as described in Section 2.4.3. The mock agents used in this example are configured with the constructions presented in Section 2.4.4. For further information about how to implement test cases, please refer to the BEAST Tool wiki in the Github repository[7].

Table 2.8 shows the required code to implement a test case for agents that run on JADEX 2.0[8]. The example contains a *setup* method (lines 3-30) where two different mock agents are configured (lines 6-11 and 14-19) to interact with a *Diagnosis Agent*, which is the AUT in this test case. During the setup, two Mock Agent are started (lines 26-29), and the agent under testing, a *Diagnosis Agent*, is started too (line 23)). Later, the *launch* method (lines 32-35) generates the trigger event of the scenario (line 33) and set an execution time of the test (line 34). Then, the agents running in the MAS platform during that time until the *verify* method (lines 37-39) is executed. In that moment, an agent belief is checked in the *Diagnosis Agent* (line 38). With that check, the test case is finished and the result is shown properly to the tester.

2.4.6 Impact of BEAST Tool

The impact of the proposed BEAST Tool have been evaluated in a quantifiable way using source code metrics. In particular, we have measured the number of test code lines required to implement tests. One of the most important benefits of developed BEAST Tool is that automatically creates a wrapper of the MAS platform and allows developers to interact with a friendly interface simplifying the implementation of tests. These metrics are strongly associated with the test implementation time that a developer consumes during this phase of development. The savings in number of code lines and in percentage are shown because they are quantifiable objective data. In contrast, the time to develop a test depends on the programming skills of the developer.

BEAST Tool is already adapted to test two MAS, one implemented on JADE and the other on JADEX, and the evaluation process has been carried out for both platforms. To simplify the comparison, twelve different test cases have been chosen for this evaluation. These test cases are quite different among them, different Mock Agents are used, different number of agents are involved in each one of them and the interaction protocols among agents are different too.

Figures 2.29 and 2.30 shows the analysed code metrics for the JADEX and the JADE

[7]BEAST Tool Wiki: `https://github.com/gsi-upm/BeastTool/wiki`.

[8]Note that some comments and other Java code lines, such as logging lines or constants definition, have been omitted in the table to clarify the code.

```
1 public class DiagnosisAgentDiagnosesDamagedSplitter extends BeastTestCase {
2
3  public void setup() {
4
5   // Configure Probe Mock Agent
6   AgentBehaviour mockProbe =  mock(AgentBehaviour.class);
7   when(mockBeh.processMessage(eq(INFORM),
8        eq("Generate High Loss Rate Symptom")))
9    .thenReturn("DiagnosisAgent", INFORM, "Loss rate=0.15");
10  MockConfiguration mockConfProbe = new MockConfiguration();
11  mockConf.addBehaviour(mockConfProbe);
12
13  // Configure Expert Mock Agent
14  AgentBehaviour mockExpert =  mock(AgentBehaviour.class);
15  when(mockBeh.processMessage(eq(REQUEST),
16       eq("Loss Rate - User Line ID: 14")))
17   .thenReturn(INFORM, "Loss rate=0.09");
18  MockConfiguration mockConfExpert = new MockConfiguration();
19  mockConf.addBehaviour(mockConfExpert);
20
21
22  // Start Diagnosis Agent
23  startAgent("DiagnosisAgent","DiagnosisAgent.agent.xml");
24
25  // Start mocks agents
26  MockManager.startMockJadexAgent("ProbeMockAgent","MediatorMock.agent.xml",
27    mockConfProbe,this);
28  MockManager.startMockJadexAgent("ExpertMockAgent","ResponderMock.agent.xml",
29    mockConfExpert,this);
30 }
31
32 public void launch() {
33  sendMessageToAgent("ProbeMockAgent",INFORM,"Generate HighLossRateSymptom");
34  setExecutionTime(2000);// Waiting time in milliseconds
35 }
36
37 public void verify() {
38  checkAgentsBeliefIsEqualTo("DiagnosisAgent",ROOT_CAUSE,DAMAGED_SPLITTER);
39 }
40
41 }
```

Table 2.8: Implementation of the exemplified scenario in a BEAST Test Case.

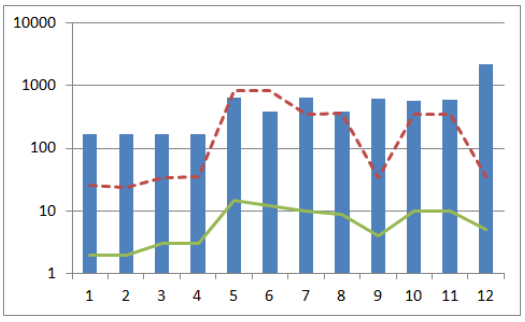

Figure 2.29: Test code lines (Y axis) per Test Case (X axis) comparison for JADEX.

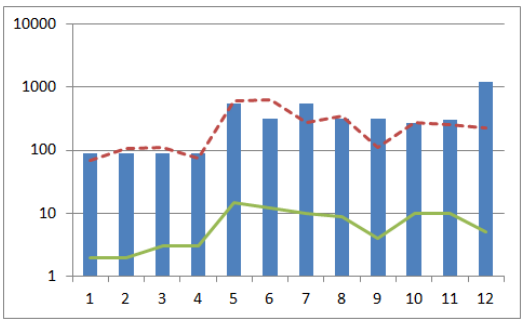

Figure 2.30: Test code lines (Y axis) per Test Case (X axis) comparison for JADE.

MAS respectively[9]. In both graphics, columns represent the code lines of AUTs and the lines represent the code lines required to implement the test with (solid line) and without (dashed line) BEAST Tool. Figure 2.29 shows the benefits of BEAST Tool in number of code lines required to implement the same test using BEAST Tool and without it for JADEX. The improvement is, in average, 247.91 lines per test, i.e. a saving of 97.22%. Figure 2.30 shows the same comparison for JADE with similar test cases. The improvement in this case is, in average, 262,08 lines per test, i.e. a saving of 97,36%.

Nevertheless, the main advantages of the BEAST approach do not come from the saving in coding tasks. The main benefit of our approach is the significant increase in communication between the stakeholders of the software project and the development team, thanks to the use of the ubiquitous language and its formalisation using BDD templates. The traceability from user requirements to the executable tests is a key point to know which tests must be executed to know if the system meets a concrete requirement. Thus, the stakeholder can check easily the status of the project at each iteration.

[9]Notice that the vertical axis of the graphics shown in Figures 2.29 and 2.30 are in logarithmic scale.

2.5 Summary

This chapter has proposed an agile testing methodology for Multi-Agent Systems based on BDD, termed BEAST Methodology, and a support tool, called BEAST Tool. The main conclusion of this work is that the BDD approach has been suitable for its application in MAS development. Furthermore, the use of BDD facilitates the communication between stakeholders and designers or developers which, usually, it is a gap between both of them. To solve this problem, the BEAST Methodology establishes that stakeholders must generate a set of behaviour specifications that describes the global system. Later, MAS designers must generate the set of agent behaviour specifications that fits the solution of the problem. These behaviours in BDD format are translated automatically with BEAST Tool to *JUnit* test cases. During this process, text plain in natural language is always available to facilitate the specification compression and communication between both stakeholders and development team.

Other common issue in MAS development is the need of other third-party agents to test the behaviour of an Agent Under Test (AUT). As these agents could be non-developed yet, BEAST uses Mock Agents to allow developers to ensure the correct behaviour of an AUT during the testing phase. To add flexibility to mocking technique, *Mockito* (Mockito Project, 2012) framework has been integrated with BEAST Tool to allow the use of the facilities of the mocking framework, such as mock agents, mock web services or mock Java objects. Besides, the use of MAS testing techniques or methodologies are commonly strongly connected to a specific MAS platform or design methodology (Coelho et al., 2006; Gómez-Sanz et al., 2009; Nguyen et al., 2008). BEAST Tool is easily adaptable for MAS frameworks as there is a clear interface between the tool and the MAS platform. Currently, JADE 4.0 (Bellifemine et al., 2007) and JADEX 2.0 (Braubach et al., 2005) are supported. Furthermore, the methodology does not impose any restriction about the design methodology. As BEAST Methodology deals with specifications of the system behaviours and tests to check if the final agents meet those requirements, the internal design of the agents is not covered by the proposed testing methodology allowing the use of any MAS design methodology.

Furthermore, for the MAS Level Testing (see Section 2.3.4), a metrics framework is proposed. When the MAS is sufficiently developed to be executed in a testing enviroment, those metrics are used to measure the outcomes of the system (*Requirement* metrics) and the suitability of the designed solution (*Design* metrics). That environment can be a testbed or a simulated scenario where the agents executes under similar conditions that the expected in the production environment. After the execution of the system in the testing environment, a set of metrics are collected and synthesised to show the behaviour of the system to the

stakeholders in a quantifiable way.

Finally, the proposed BEAST Methodology defines mechanisms to extract the domain knowledge that will be required to develop the agent architecture described in the following chapter. That knowledge must be gathered from human experts in network engineering in order to design and implement suitable solutions for an autonomic fault diagnosis approach which improves the current state of the art on this topic.

CHAPTER 3

Agent Architecture for Autonomic Fault Diagnosis

This chapter presents an Agent Architecture for Autonomic Fault Diagnosis of Telecommunication Networks, which is based on an extended BDI model that combines heterogeneous reasoning processes: ontology-based reasoning and Bayesian reasoning. The proposed architecture is focused on handling the uncertainty of the complex systems during the diagnosis process. The domain knowledge obtained applying the methodology proposed in the previous chapter is used to feed the models an agent applies to carry out specific diagnostic tasks. The proposed agent architecture uses a Diagnosis Model that bridges a Structural Model which semantically describes the network and a Causal Model that expresses the fault-symptom relations. The architecture is described in detail in the chapter and evaluated in two different scenarios. One of them is a real network environment, in which the behaviour of the system has been analysed with data collected while the system was working on a real-live network which offers an enterprise service during one and a half years. The other one is a simulated environment where a wireless sensor networks is running for a motion detection application.

3.1 Introduction

As introduced in Section 1.1, Network Management is one of the most expensive tasks for telecommunication operator companies (Agoulmine, 2011). Consequently, there is a trend of delegate the management of the network to itself. This is know as Autonomic Networking (Behringer et al., 2015). But, today, Fault Diagnosis is still being a non-autonomic task (Jiang et al., 2015). Traditionally, this process has been carried out by humans experts supported by surveillance systems for symptom detection. But, even with those systems, the diagnosis task is mainly a manual process. The constant increase in the size and complexity of the network makes fault diagnosis a critical task that should be handled quickly and in a reliable way. For that, high skilled engineers are required to carry out this task, although even these individuals are not always able to deal with the increasing heterogeneity and complexity of the networks making diagnosis a difficult, time-consuming and expensive task.

In consequence, operators have the goal of fully automating fault diagnosis (Laurent et al., 2013; Kephart et al., 2007) to reduce operation costs and improve customers' experiences through the automated operation of standardised diagnostic processes. The increasing heterogeneity of networks increases their complexity and that complexity produces a high level of uncertainty which is one of the main challenges to face an autonomic diagnostic process (Pras et al., 2007). At the beginning of the diagnostic process, only partial information is known, and it is not feasible to collect or even model all the available information in complex environments such as telecommunications networks which are composed of multiple subnetworks, equipment and technologies in constant evolution. Thus, diagnosing telecommunications networks requires reasoning under uncertainty. However, uncertainty is not the unique challenge for an automated diagnostic process. The distributed location of data in the network and the existence of federated domains can be an issue related with privacy and scalability. Hence, the capability to reason under uncertainty in a distributed way keeping coherence is also an important feature for autonomic diagnostic system.

Based on these requirements, this chapter presents an Agent Architecture designed to perform autonomic fault diagnosis tasks of telecommunication networks under uncertainty. The capability of perform distributed diagnosis processes for federated environment is presented in the next chapter as a coordination framework where agents can discuss keeping coherence during a distributed fault diagnosis process. The agent architecture presented in this chapter is based on an extended BDI model, named BDI for Bayesian Diagnosis (B2D2), that combines Bayesian reasoning to handle the uncertainty with ontology-based reasoning for domain knowledge inference. The rest of this chapter is structured as follows. First, we

discuss related works in Section 3.2. Section 3.3 describes the knowledge model used during a diagnostic process. Section 3.4 presents the B2D2 Agent Architecture for autonomic fault diagnosis in telecommunication networks. Section 3.5 shows two different case studies: one real-life telecommunication network where a fault diagnosis agent is deployed and a simulated environment that complements the evaluation of the proposed architecture. Finally, Section 3.6 presents some concluding remarks of this chapter.

3.2 Related Work

The concept of autonomic networking is being introduced in order to build the Future Internet (Tselentis and Galis, 2010), but the current Internet does not facilitate that objective (Rubio-Loyola et al., 2010). Nonetheless, some autonomic solutions can be found in the literature. Massie et al. (2004) propose a passive scalable distributed monitoring approach for high-performance computing focus on programmability of network interfaces. Other decentralised passive approach based on monitoring probes is proposed by Di Pietro et al. (2010). In contrast, Song et al. (2009) propose a centralised monitoring control which includes some active measurement mechanisms. But, autonomic networking is not only about monitoring. Further actions should be taken if any anomaly or undesired state is detected (Agoulmine, 2011). A self-healing approach based on Peer-to-Peer (P2P) management for an intrusion detection scenario is proposed by Duarte et al. (2011). That P2P-based self-healing service is composed of independent monitoring and healing services, which shows an example of the cooperation among different services required for the Future Internet (Chai et al., 2010).

In recent years, several research projects have proposed autonomic architectures and reference models, such as ANA FP7 Project[1], Autonomic Internet FP7 Project[2] or ASCENS FP7 Project[3]. Furthermore, the ETSI and the IRTF have shown their interest about this research field generating some reference models, such as GANA (Laurent et al., 2013) and RFCs, such as RFC 7575 (Behringer et al., 2015) and RFC 7576 (Jiang et al., 2015). But, although these projects remark the importance of the fault-management in an autonomic architecture, there is no a clear fault-management approach in the current reference models and RFCs. This encouraged us to focus our work to propose some fault-diagnosis mechanisms and models applicable to generic autonomic architectures, but without a strong dependence of any them due to the lack of standardisation.

Nevertheless, all autonomic approaches agree about the need of an appropriate knowl-

[1]ANA Project Website: http://www.ana-project.org/
[2]Autonomic Internet Project Website: http://www.autoi.ics.ece.upatras.gr/
[3]ASCENS FP7 Project Website: http://www.ascens-ist.eu/

edge management. Ontology models are the most ground approach for knowledge modelling (Subbaraj and Venkatraman, 2015; Monteiro et al., 2014), as they provide a common terminology which is essential to enable cooperation among self-managed entities (Staab and Studer, 2013) and that necessity of coordination is a point where all autonomic approaches agree. That cooperation grounds in the idea that autonomic management entities need to cooperate among them to achieve a desired goal, but this is not a new idea in the agent engineering (Wooldridge, 2009) field where the concept of agent grounds exactly in the same idea. Consequently, some authors explore the possibility of applying Agent Technology for Autonomic Networking (Paolino et al., 2011; Sanz-Bobi et al., 2010; Cox et al., 2007). Martín et al. (2012) presented a framework based on intelligent agents for network management that used rule-based reasoning. Both Leitão et al. (2012) and Mendoza et al. (2011) present MAS focused on the reconfigurability of the system for collaborative tasks using adaptive agents. Luo et al. (2012) proposed a fault diagnosis system using Dempster-Shafer evidence theory (Dempster, 1967; Shafer, 1967) in combination with rules to resolve the possible conflicted information from multi-sensors systems. In our work, we solve this issue using Bayesian networks (Pearl, 1985) to handle the uncertainty of the incomplete or unreliable data.

All in all, today fault-management is a non-autonomic network management task (Jiang et al., 2015), but there is a clear interest to achieve it. This motivated us to propose an autonomic fault-diagnosis solution based on agent technology focusing on some challenging features of the Future Internet, such as handling uncertainty, heterogeneity and complexity.

3.3 B2D2 Knowledge Model

A knowledge model is composed of different models, each capturing a related group of knowledge structures. Those models describe different aspects of a specific problem to enable the development of a solution to solve it. This section presents a set of models which enables an agent to carry out an autonomous diagnosis process. The agent architecture proposed to use these models is presented in Section 3.4. Those knowledge models are divided in two main areas: *Domain Model* and *Inference Model*. They are presented in the following sections as exposed below. Section 3.3.1 presents the *Domain Model* used to describe the domain knowledge, in our case, fault diagnosis of telecommunication networks. Section 3.3.2 shows the *Inference Model* used to define the tasks required to carry out a diagnosis process.

3.3.1 Domain Model

The domain model describes the main static information and knowledge in an application domain, in our case, the fault diagnosis task for telecommunication networks. This section exposes two type of domain models which offer complementary views of the domain knowledge. Those models must be instantiated by agents with their own knowledge bases which must contain information about specific networks they are diagnosing. Those models are presented in the following sections as exposed below. Section 3.3.1.1 presents a Structural Model that contains knowledge about the network itself using Infrastructure and Network Description Language (INDL). Section 3.3.1.2 defines a Causal Model that relates the symptoms with the possible fault root causes while handling uncertainty using Bayesian networks.

3.3.1.1 Structural Model

A Structural Model (SM) of a telecommunication network is defined as a representation of the components of the network and their properties. Formally, a SM is a pair $< E, P >$, where E is a set of elements that compose the telecommunication network and P is a set of properties that define the set of elements contained in E and the relation among them. Focusing on the application of a SM in the Fault Diagnosis task under consideration, this model describes semantically the elements of the network, E, that could cause or be affected by a fault. The model must define how those elements are related among them and their expected status, as properties P, to know when something is out of the admissible range.

The use of ontologies for modelling the required knowledge to build the autonomic Future Internet is a grounded idea (Monteiro et al., 2014). There are many ontologies proposed for different fields in telecommunications. The Common Information Model (CIM) (CIM Specification Group, 2015) provides a network device information model based on Unified Modelling Language (UML) commonly used in enterprise settings. Moreover, some extensions have appeared, such as the Directory Enabled Networking-next generation (DEN-ng) model (Strassner, 2002) which extends the CIM with the description of business rules. The Virtual private eXecution infrastructure Description Language (VXDL) (Koslovski et al., 2009) is mostly focused on modelling requests for virtual infrastructures. Some ontologies, such as Semantic Resource Description Language (SRDL) (Abosi et al., 2009) and Media Applications Description Language (MADL) (Ntofon et al., 2012), suffer from a limited ability to model connectivity, but do not the properties for the network resources which makes them semantically indistinguishable and unusable within a semantic context. To cover that

lack, Infrastructure and Network Description Language (INDL) (Ghijsen et al., 2013) is focused on the semantic properties to describe the storage and computing capabilities of the resources in combination with a very generic network description schema from the Network Markup Language (NML), which is being standardised[4]. Therefore, we propose the adoption of INDL to describe the Structural Models (SMs) for a generic network schema. Moreover, INDL offers the application of self-discovery techniques for an automatic SM generation. A toolkit to generate INDL automatically based on a set of traffic protocols, such as Open Shortest Path First (OSPF), Traffic Engineering Extensions to Open Shortest Path First (OSPF-TE) or Inter-Domain Controller Protocol (IDCP) is available as an open-source tool[5].

Some basics classes of INDL are briefly mentioned below and shown in Figure 3.1. The *NetworkObject* class represents the elements of the network, denominated above as E in the SM, and it is instantiated with complementary concepts, such as *Link*, *Node* or *Service*. These concepts are extended with others that specifies more information about them in type, such as *BidirectionalLink* or *AdaptationService*, or in structure, such as *NodeComponent*. As can be seen in Figure 3.1, these are only a few concepts of the ontolgy. For further information, please refer to the complete INDL ontology[6].

Finally, INDL is suitable to describe our structural model considering its classes as elements E of our model and its semantic relations as properties P. In Table 3.1, we illustrate its use for representing some network elements in a simplified WSN scenario. This scenario consists of a sink tree topology, where all messages are routed to a sink node, called Base Station. For this scenario, we have extended the basic INDL concepts with the appropriate classes for this type of network. That extension classes are shown in example with the *b2d2-wsn* prefix. In this example shown in Table 3.1, there are three different elements: a device, a link, and a link group, which is an inferred element from the other ones. We find the description of a sensor device (lines 10-20), specifically, a *ZigBeeRouter* (line 11) (following the ZigBee[7] notation). This device receives messages from other sensors what is represented with the property *isSink* (lines 14-17). At the same time, this node sends messages to others what is described with the propery *isSource* (line 18). Both *isSink* and *isSource* properties connects a network elements with links. Hence, those links are represented as instances of the *Link* class (lines 22-24). Based on this simple concept, the instance of the *PathToBaseStation* class (lines 26-33) represents a group of links that forward

[4]NML Standardisation Work Group: `http://redmine.ogf.org/projects/nml-wg`

[5]Public Github Repository of Pynt toolkit : `https://github.com/jeroenh/Pynt/`

[6]The INDL ontology file can be found in: `https://github.com/jeroenh/indl/blob/master/owl/indl.owl`

[7]ZigBee Alliance: `http://www.zigbee.org/`

Figure 3.1: Main classes of the Infrastructure and Network Description Language.

messages to the sink node. This composed element can be inferred by the agent based on the information of isolated links between the sink node and any sensor, which is an interesting capability when a Fault Diagnosis agent is running in a dynamic network scenario, as in the WSN scenario under consideration.

In conclusion, this section presented the Structural Model (SM) that allows agents to reason about the network elements while carrying out the Fault Diagnosis task. But, gathering the required knowledge to build the SM for specific networks is not a trivial task. Hence, we propose the application of the BEAST Methodology for that knowledge extraction, as proposed in Chapter 2. With the knowledge extracted from network operators in the *System Specification Behaviour* phase, we can described the network in a Structural Model using the INDL language to feed agents with that domain knowledge.

3.3.1.2 Causal Model

A Causal Model (CM) is defined as an abstract model that describes the causal mechanisms of a system. Concretely, Judea Pearl (Pearl, 2000) defines a CM as an ordered triplet $< U, V, E >$, where U is a set of exogenous variables whose values are determined by factors outside the model; V is a set of endogenous variables whose values are determined by factors within the model; and E is a set of structural equations that express the value of

```
@prefix owl:
        <http://www.w3.org/2002/07/owl#> .
@prefix indl:
        <http://www.science.uva.nl/research/sne/indl#> .
@prefix nml:
        <http://schemas.ogf.org/nml/2013/05/base#> .
@prefix b2d2-wsn:
        <http://www.gsi.dit.upm.es/ontologies/b2d2/wsn/ns#> .

b2d2-wsn:sensor-15 a nml:Node,
        b2d2-wsn:ZigBeeRouter,
        b2d2-wsn:ZigBeeSensorNode,
        b2d2-wsn:ZigBeeSoftware ;
    nml:isSink b2d2-wsn:path-sensor-15-sensor-22,
        b2d2-wsn:wifi-sensor-11,
        b2d2-wsn:wifi-sensor-12,
        b2d2-wsn:wifi-sensor-9 ;
    nml:isSource b2d2-wsn:path-sensor-68-sensor-15 ;
    nml:locatedAt b2d2-wsn:location-sensor-15 ;
    nml:name "sensor-15"^^xsd:string .

b2d2-wsn:path-sensor-60-sensor-15 a nml:Link,
        b2d2-wsn:RoutePathLink ;
    nml:name "path-sensor-68-sensor-15"^^xsd:string .

b2d2-wsn:pathToBaseStationFrom-sensor-15 a nml:LinkGroup,
        b2d2-wsn:PathToBaseStation ;
    nml:hasLink b2d2-wsn:path-base-station-sensor-68,
        b2d2-wsn:path-sensor-68-sensor-15 ;
    nml:hasNode b2d2-wsn:base-station,
        b2d2-wsn:sensor-15,
        b2d2-wsn:sensor-68 ;
    nml:hasTopology b2d2-wsn:cluster-sensor-15 .
```

Table 3.1: Examples of the usage of INDL to describe the Strcutural Model.

each endogenous variable as a function of the values of the other variables in U and/or V.

Following Pearl's definition, a causal model can be represented as a network with nodes (variables $U \cup V$) and links between those nodes (structural equations E), called causal network. A Bayesian Network (BN) (Pearl, 1988) can be considered as a causal network if the relations between its variables are causal relations. Formally, a BN is a probabilistic graphic model that represent a set of random variables $\mathcal{V}$ and their conditional dependencies $\mathcal{R}$ via a Directed Acyclic Graph (DAG). Hence, the equivalence between a CM and a BN is given if the variables of the network are both exogenous and endogenous variables ($\mathcal{V} = U \cup V$) and the conditional relations are only causal relations ($\mathcal{R} = E$).

Focusing on the structure, a BN is composed by layers of variables that are related with variables in other layers. The simplest models, such as QMR (Myers, 1987) or QMR-DT (Shwe et al., 1991), are composed only by two layers: symptoms and fault root causes, as shown in Figure 3.2. In contrast, other models, such as the BN3M Model (Kraaijeveld et al., 2005), adds context information to the variables set. Thus, a third layer is added to the model, as shown in Figure 3.3. This three layer structure is still simple and its performance is reasonably similar to the two layer models (Middleton et al., 1991) and the usage of only three types of variables keeps it easy for a human to identify which variables belong to each type. Additional layers, i.e. more types of variables, will increase the complexity of the model and will not increase the diagnostic performance (Provan, 1995).

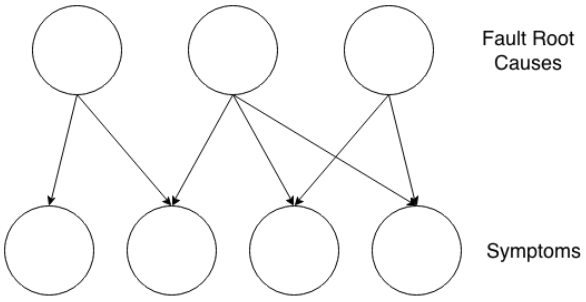

Figure 3.2: Two layers model of Bayesian network.

The models presented above can be used to build causal models that uses evidences collected from the environment (in our case, from the telecommunication network) to discriminate between the possible fault root causes. While two layer models use only symptoms and fault root causes (and the relations among them) to form the structure of the network, three layer models add context variables to be able to use the same model in different contexts. For our purpose of fault diagnosis in telecommunication networks, we can use both types of model, depending of the information that can be collected from the environment.

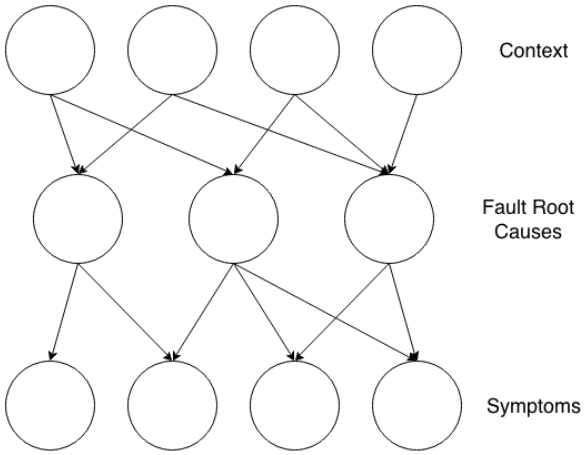

Figure 3.3: Three layers model of Bayesian network.

As the Structural Model, the Causal Model is built based on the domain knowledge extracted in the *System Specification Behaviour* phase of the BEAST Methodology. Network operators specify in that phase what data can be collected from the network and they are modelled as evidence variables. In the same way, the possible fault root causes or any context information are included in the causal model as variables completing the variables set ($\mathcal{V}$). The causal relations ($\mathcal{R}$) are required to complete the model. But, the structural equations that express those relations are not always easily defined by network operators. If that information is not available after the knowledge gathering process, we can apply self-learning algorithms to find those equations (Voortman, 2005; Oniśko et al., 2001) based on data collected from the network.

We illustrate the result of this knowledge gathering process with an example of the CM defined for a real-life telecommunication network corresponding to one of the case studies shown in Section 3.5. Figure 3.4 shows the structure of the causal model following the BN3M Model with three types of variables: evidences, fault root causes and context variables. The example model is composed of 27 evidence variables (yellow nodes), 17 fault root causes (blue nodes) and 18 auxiliary nodes (white nodes) which sum a total of 62 variables ($\mathcal{V}$). Every node in the model is associated to a Conditional Probability Table (CPT) that contains the structural equations ($\mathcal{R}$) that complete the causal model. A simplified CPT is shown in Figure 3.5. Those conditional probability equations define the probability that a variable of the model is in a concrete state given other related variables are in specific states.

Following the decision of using ontologies for knowledge modelling, Probabilistic OWL (PR-OWL) extends the OWL language to represent probabilistic ontologies. This is based on Bayesian first-order logic what fits perfectly to describe the Bayesian networks of the Causal Models using ontologies. Thanks to some concepts, such as *ProbAssign* and *ProbDist*,

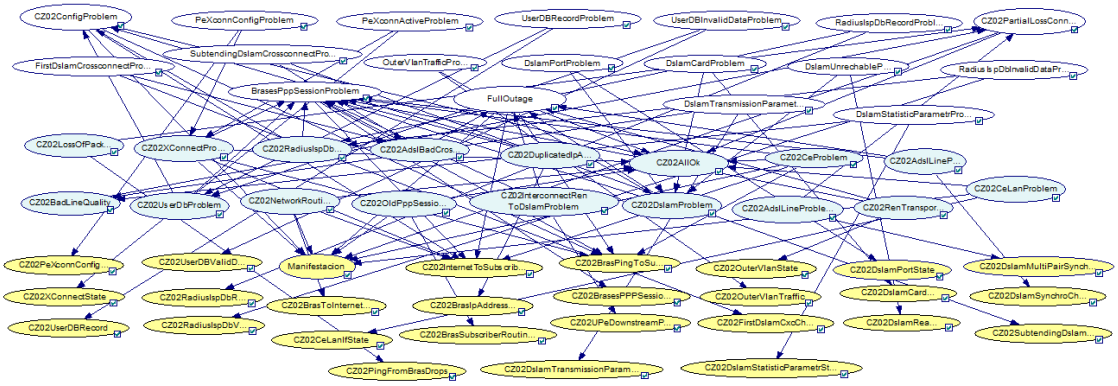

Figure 3.4: Example of the structure of a Causal Model following the BN3M Model.

PeXconnConf...	YES		NO	
PeXconnActi...	YES	NO	YES	NO
YES	0.97	0.68	0.59	0.05
NO	0.03	0.32	0.41	0.95

Figure 3.5: Example of a CPT which relates variables of a Causal Model.

PR-OWL allows to define complex conditional probability distributions or simple declarative confidence values for any entity of the model. For further information, please refer to the complete PR-OWL ontology[8].

In conclusion, this section presents the Causal Model (CM) which enables agents to reason under uncertainty about the possible fault root causes based on the evidences collected from the network. As the SM, the application of BEAST Methodology, exposed in Chapter 2, is proposed to gather the knowledge required to define the CM and, if required, self-learning algorithms can be applied to complete and refine the model.

3.3.2 Inference Model

The inference model describes how the domain knowledge defined in the previous section can be applied to carry out the reasoning process. Section 3.3.2.1 shows a Task Model that defines the different phases of a diagnosis process and proposes several solving methods to perform it. Section 3.3.2.2 proposes a Diagnosis Model which combines the Structural and the Causal Models to allow a B2D2 agent to handle that knowledge during the different phases of the diagnosis process defined by the Task Model.

[8]PR-OWL Website: `http://pr-owl.org/`

3.3.2.1 Task Model

This section presents a generic task model following the MAS-CommonKADS methodology (Iglesias et al., 1998) based on the analysis exposed by Benjamins (1995). In this model, a **task** is a specification of a goal that is needed to be achieved. It can be decomposed into subtasks by a **Problem-Solving Method (PSM)**. A PSM is the definition of the way to achieve the goal specified in a task. A task can be realised by several methods consisting of subtasks that can be reused in different PSMs. In the following diagrams, PSM are represented by rectangles and tasks by ellipses. Following this nomenclature, the Fault Diagnosis task is realised by the Prime Diagnostic Method (see Figure 3.6), which is decomposed into three subtasks: (i) *Symptom Detection*, finding out whether complaints are indeed symptoms, (ii) *Hypothesis Generation*, generating hypotheses of possible causes based on the symptoms, and (iii) *Hypothesis Discrimination*, discriminating among the hypotheses based on additional observations.

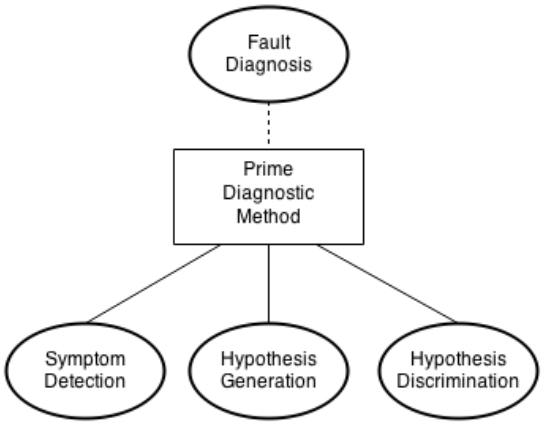

Figure 3.6: Prime Diagnostic Method.

Symptom Detection As shown in Figure 3.7, there are three main methods to perform the *Symptom Detection* task:

- The *Compare Symptom Detection* method compares the value of a variable obtained from the environment with the expected value obtained from a model. After that comparison, if the detected behaviour is not the expected one, we say that a symptom is detected.
- The *Classify Symptom Detection* method is based on a classification task. The information observed from the environment is filtered by a classification engine that decides if the observations are a symptom or not.

- The *User Symptom Detection* is the manual report from a user that is supposed to be truth. Then, no check or classification process is required.

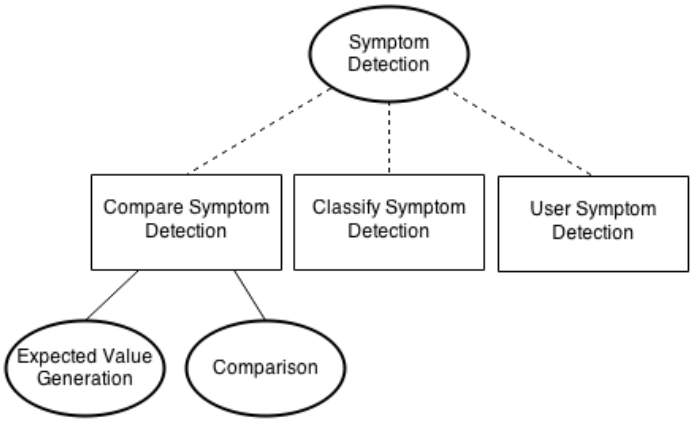

Figure 3.7: Symptom Detection Task.

Hypothesis Generation After a symptom is detected, the Prime Diagnostic Method continues with the next task: *Hypothesis Generation*. The two main options to achieve this goal are the use of a *Compiled Method* or a *Model-based Method*, as depicted in Figure 3.8.

- The *Compiled Method* is composed by three subtasks which use associations between symptoms and causes and can be followed by a probability filter. This method starts with an abstraction process that translates between raw observations to qualitative and generalised observations. The method continues with the association of that abstracted observations with possible causes of fault. Finally, an optional task is the addition of a probability filter to discard non-probable hypotheses.
- The *Model-based Method* follows other approach based in non-abstracted observations. Firstly, the finding of a set of model entities that could contribute to an abnormality observation is required. Then, that set is transformed into a hypothesis set in which every element is a possible explanation for the observed symptoms. Finally, a prediction-based filtering can be used to discard inconsistent hypotheses.

The *Causal Model* presented in Section 3.3.1.2 is used as *Compiled Method* in the Agent Architecture for this *Hypothesis Generation* task, as exposed appropriately in Section 3.4.2.

Hypothesis Discrimination After a hypothesis set is generated, the *Hypothesis Discrimination* task is performed to find the final conclusion of the Fault Diagnosis Task. The

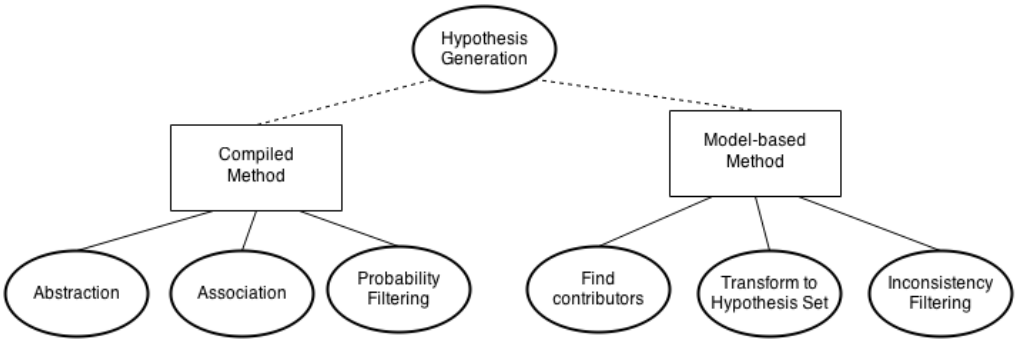

Figure 3.8: Hypothesis Generation Task

method described by Benjamins (1995) is divided in four subtasks, as shown in Figure 3.9.

The *Discrimination* task starts with a decision about the next action to gather more information about the fault under diagnosis. This decision can be made based on different criteria, such as time restrictions or computational cost of the possible actions. When the action is decided, data about possible observations are collected and analysed to find new symptoms or evidences. Finally, the hypothesis set is updated based on the observations collected in the previous task. If a reliable conclusion is found in the *Hypotheses Update* task, the *Fault Diagnosis* task can be considered as finished. If not, the discrimination process continues repeating the previous tasks (*Hypothesis Selection*, *Data Collection* and *Symptom Detection*) until a reliable conclusions is achieved.

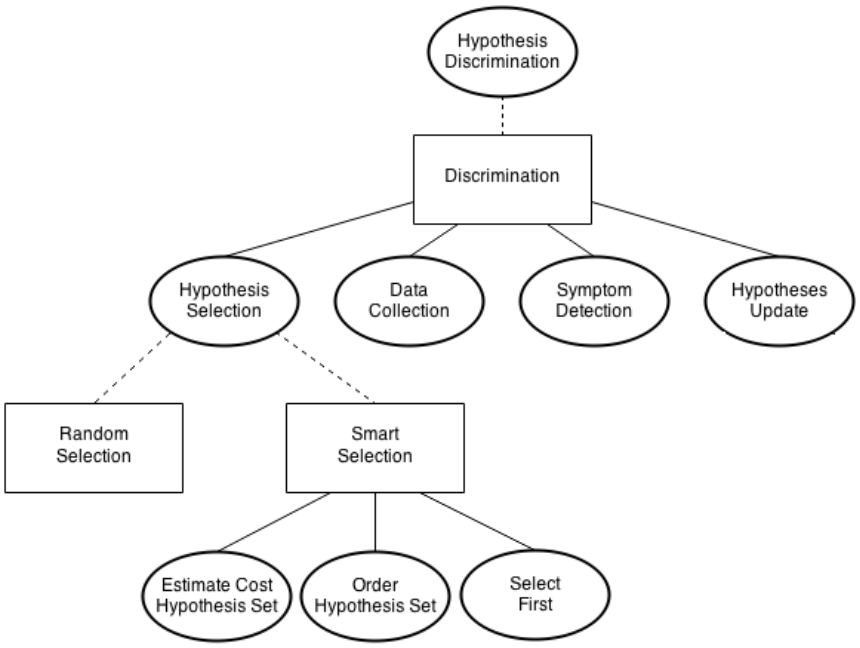

Figure 3.9: Hypothesis Discrimination Task

Summarising, the task model presented above describes a generic Fault Diagnosis process that can be carried out by an agent or a set of agents, both human, software agents or a combination of them. This general task model is particularised for the reasoning cycle of proposed Agent Architecture in Section 3.4.

3.3.2.2 Diagnosis Model

This section proposes a Diagnosis Model designed to allow an agent to carry out an autonomic fault diagnosis process. The developed model is formalised as an ontology which covers concepts that combine the three models presented previously in this chapter: the *Task Model*, the *Structural Model* and the *Causal Model*. The main classes of the ontology are shown in Figure 3.10. The most important conceps in the model[9] is the *Diagnosis*, which is performed by *actors* that execute *actions* to collect *observations* from the supervised *network objects*. Based on those *observations*, a *hypothesis* set is generated and discriminated until a *conclusion* with enough confidence is reached. This simplified overview is exposed to facilitate the understanding of the Diagnosis Model, but the complete model is available on the web[10]. Nevertheless, further explanations are included below.

The *diagnosis* process considered in the model is composed by three different *phases*. The model defines three different *states* for these *phases*: pending, ongoing or finished, which are used to handle the progress of the *diagnosis*. Firstly, during the *Symptom Detection* phase, *monitoring actions* are being executed by the *agent* until a *symptom* is detected. That *symptom* is considered an *observation* collected from a specific *network object*[11]. Then, during the *Hypothesis Generation* phase, the *agent* analyses the *Causal Model*[12] to know what are the possible fault root causes[13] of the detected *symptom* and generates the corresponding *hypotheses*. Every *hypothesis* holds an initial *confidence*[14] about a type of *fault* is occurring on a *network object*. That *confidence* is a key concept to allow agents to reason under uncertainty. When the hypothesis set has been generated, the *hypothesis discrimination* phase starts looking for any *available testing action* to collect information about suspicious *network objects*. The *availability* of a specific *action* is defined by a set of required *condi-*

[9] Notice that all the *italic* concepts refer to classes of the proposed Diagnosis Model to simplify the explanations.

[10] Diagnosis Model Ontology Specification: `http://www.gsi.dit.upm.es/ontologies/b2d2/diagnosis`

[11] *Network Object* is one of the concepts included in INDL

[12] In the Diagnosis Model, the *Causal Model* is equivalent to the *MTheory* concept of PR-OWL.

[13] Using the PR-OWL language combined with the BN3M Model, fault root causes are modelled as domain resident nodes and evidences and contexts variables, as finding resident nodes.

[14] The hypothesis confidence is equivalent to the *ProbAssign* concept of PR-OWL.

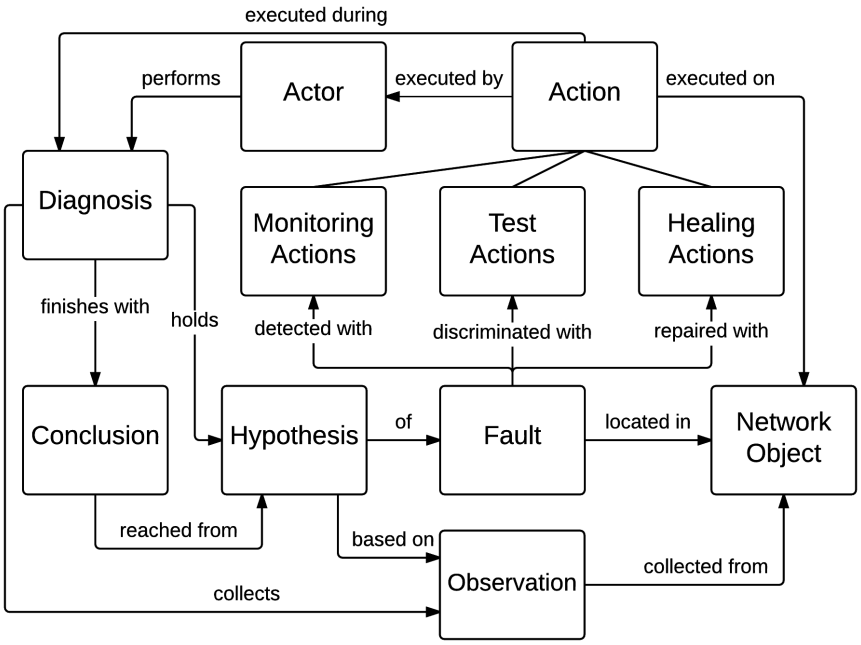

Figure 3.10: Main classes of the Diagnosis Model.

tions that must be satisfied to execute it. For example, the most simple condition is that an *actor* must have the capability to execute a specific *action* to consider it as *available*. After a *testing action* is executed, a new *observation* is gathered. That *observation* is used to feed the *Causal Model* and update the *hypothesis* set[15]. If any *hypothesis* reachs enough *confidence*, it is selected as *conclusion* and the *diagnosis* finishes. If not, additional *testing actions* are executed until any *hypothesis* is clarified or there is no *available testing actions*, in which case the most probable *hypothesis*, i.e. the one with higher *confidence*, is picked as *conclusion*.

The concept of *action* has a strong relation with the concept of *network object*. The specification of possible *actions* that could be executed on every *network object* is an essential knowledge which should be included in the *Structural Model* to allow agents to plan their behaviours autonomously. Accordingly, the definition of *conditions* for those actions is required too. That information is used to defined the capabilities of a specific agent. That enables the possibility to specialise agents in specific type of actions, which execution could be requested by third-party agents as required for distributing the work load.

To illustrate the use of the proposed model, Table 3.2 shows a simplified example extracted from one of the case studies presented in Section 3.5. Some data have been removed

[15] *Observations* are modelled as input to the *Causal Model* using the *Finding Input* class of PR-OWL.

from the original example to facilitate the reading and understanding of the example. It includes a *Diagnosis* case (lines 8-15) with *Observations* (lines 9-11) and *Hypotheses* (lines 12-13). The *Observation* (lines 17-20) included in the example contains information about from what *network element* was collected (line 18), what variable of the *Causal Model* feeds (line 19) and what action was performed to collect it (line 20). The *hypothesis* (lines 22-25) exposed in the example has an associated confidence value (line 23) for a specific *fault* (line 25) and it is associated with a variable of the *Causal Model* too (line 24). Moreover, that *fault* (lines 30-33) has a suspicious *network element* as *location* (line 32).

In conclusion, this section presented a *Diagnosis Model* that covers the concepts required to carry out an autonomic fault diagnosis process. It is formalised as an Ontology Web Language (OWL) ontology which is publicly available on the web[16]. The proposed model follows the *Task Model*, described in Section 3.3.2.1 and uses the *Structural Model*, shown in Section 3.3.1.1, to retrieve information from the network, and the *Causal Model*, shown in Section 3.3.1.2, to deal with the uncertainty of the diagnosis process and inference possible causes of fault; which makes this is the core model of the proposed B2D2 Agent Architecture shown in Section 3.4.

[16]Diagnosis Model Ontology Specification: `http://www.gsi.dit.upm.es/ontologies/b2d2/diagnosis`

```
@prefix pr-owl:
        <http://www.pr-owl.org/pr-owl.owl#>
@prefix b2d2-diag:
        <http://www.gsi.dit.upm.es/ontologies/b2d2/diagnosis/ns#> .
@prefix b2d2-wsn:
        <http://www.gsi.dit.upm.es/ontologies/b2d2/wsn/ns#> .

b2d2-wsn:diagnosis-034 a b2d2-diag:Diagnosis ;
                b2d2-diag:hasCollectedInformation b2d2-wsn:observation-011,
                                b2d2-wsn:observation-012 ;
                                b2d2-wsn:symptom-002 ;
                b2d2-diag:hasHypothesis b2d2-wsn:hypotesis-03,
                    b2d2-wsn:hypothesis-04 ;
                b2d2-diag:hasPerformedAction b2d2-wsn:testAction-002 ;
                b2d2-diag:isStartedBySymptom b2d2-wsn:symptom-002 .

b2d2-wsn:observation-011 a b2d2-diag:Observation ;
                b2d2-diag:collectedFrom b2d2-wsn:sensor-15 ;
                b2d2-diag:isCausalModelInput b2d2-wsn:fd-input-003 ;
                b2d2-diag:gatheredWithAction b2d2-wsn:testAction-002 .

b2d2-wsn:hypothesis-03 a b2d2-diag:Hypothesis ;
                b2d2-diag:hasConfidence b2d2-wsn:prob-hyp-03 ;
                b2d2-diag:isCausalModelOutput b2d2-wsn:bn-var-005 ;
                b2d2-diag:representsPossibleFault b2d2-wsn:poss-fa-002 .

b2d2-wsn:prob-hyp-03 a pr-owl:ProbAssign ;
                pr-owl:hasStateProb 0.78 .

b2d2-wsn:poss-fa-002 a b2d2-wsn:OverloadedDevice,
                                b2d2-diag:Fault ;
                b2d2-diag:hasLocation b2d2-wsn:sensor-15 ;
                b2d2-diag:canBeRepairedWith b2d2-wsn:healing-action-004 .
```

Table 3.2: Example of application of the Diagnosis Model.

3.4 B2D2 Agent Architecture

This section describes the BDI for Bayesian Diagnosis (B2D2) Agent Architecture which aim is to achieve an Autonomic Fault Diagnosis in Telecommunication networks based on the Belief-Desire-Intention (BDI) model (Wooldridge, 2000) and in the knowledge models presented in Section 3.3. A simplified overview of the proposed agent architecture is shown in Figure 3.11. The Reasoning Cycle of a B2D2 Agent follows the *Task Model* exposed in Section 3.3.2.1. The three tasks that compose the diagnosis process are formalised using the AgentSpeak language (Rao, 1996) in the following subsections where the reasoning processes are explained.

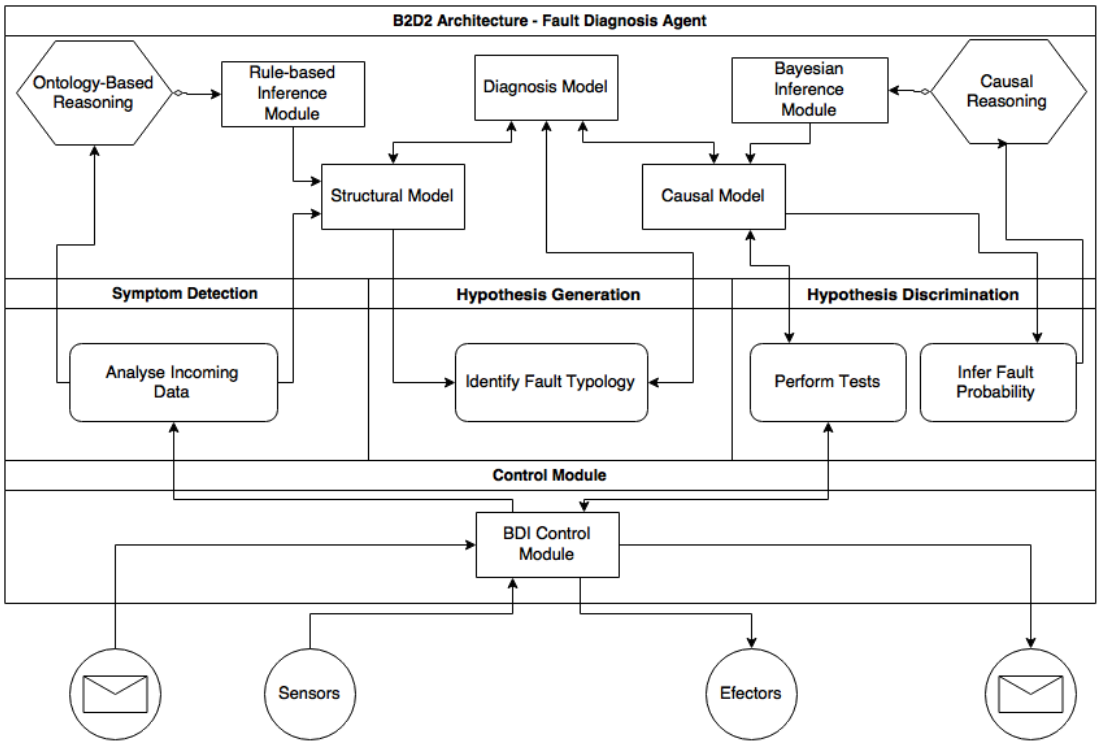

Figure 3.11: B2D2 Agent Architecture.

3.4.1 Monitoring the network

The *Symptom Detection* task is realised monitoring the network and analysing the incoming data. This analysis is based on a rule-based inference process. As the Diagnosis Model is described in OWL language, we propose the use of the SPARQL Inference Notation (SPIN) rules[17] to have a direct integration between the knowledge models and the inference engine.

[17]SPIN Web Site: http://spinrdf.org/

```
1 CONSTRUCT {
2     ?path nml-base:hasNode ?nextRouter .
3     ?path nml-base:hasLink ?link .
4 }
5 WHERE {
6     ?this a b2d2-wsn:ZigBeeRouter .
7     ?path a b2d2-wsn:PathToBaseStation .
8     ?path nml-base:hasNode ?this .
9     ?this nml-base:isSource ?link .
10    ?nextRouter nml-base:isSink ?link .
11 }
```

Figure 3.12: Example of SPIN rule to add routers to a Path.

These rules can be used to detect anomalies in the expected behaviour of network elements and/or to updated the dynamic network features, such as topology, network load or services status, represented in the *Structural Model*. A simple example of a SPIN rule extracted from one of the case studies included in Section 3.5 is shown in Figure 3.12. This rule is used to discover the path from any router to the sink node in a sink tree topology. So, the agent can infer the topology of the network dynamically if it changes during the operation of the system. The example rule infers that if a router (line 6) is in a specific path (line 7-8) and that router sends packages to other router (line 9-10), this other router which receives the packages is in the path too (lines 2-3).

This task is formalised in Table 3.3. As mentioned, previously, the agent starts its execution with the initial goal of monitoring specific network elements described in the *Structural Model* (line 8). Always a diagnosis process is not in progress, this monitoring goal will be active (lines 12-15). While monitoring, the agent analyses the incoming data from the supervised network elements until an anomaly is detected (lines 17-20). The appearance of this symptom actives a new goal to diagnose it. Then, the next diagnosis phase starts (lines 22-29).

3.4.2 Detecting possible faults

The *Hypothesis Generation* task is realised analysing the structural relations among variables contained in the *Causal Model*. As explained in Section 3.3.1.2, the *Causal Model* is composed of evidences observed from the network which are used to discriminate between

```
1  // B2D2 Agent
2  /* Agent beliefs */
3  causalModel(evidences, contexts, faultsRootCauses, relations).
4  structuralModel(networkElements).
5  diagnosisModel(hypotheses, observations, actors, actions).
6
7  /* Initial goal */
8  !monitor(networkElements).
9
10 /* Symptom Detection - Plans */
11 // Start to monitorise
12 +!monitor(networkElements) :
13          not .desire(diagnose(symptom))
14          <-
15          !analyseData(networkElement).
16 // Update system info
17 +!analyseData(networkElement)
18          <-
19          update(strucuturalModel);
20          !lookForAnomaly(networkElement).
21 // Start a diagnosis or continue monitoring
22 +!lookForAnomaly(networkElement) :
23          anomaly(symptom)
24          <-
25          !diagnose(symptom).
26 +!lookForAnomaly(networkElement) :
27          not .desire(monitor(networkElement))
28          <-
29          !monitor(networkElement).
```

Table 3.3: Symptom Detection Task in AgentSpeak language.

the possible fault root causes. Those variables are related among them by a set of structural equations. In this phase, those relations are explored to know what possible faults are, or are not, related with the detected symptom. For this purpose, the agent checks the definition of the *Causal Model* to get those relations among variables. This model definition must be instantiated in a Bayesian inference engine to allow the agent to reason under uncertainty during the discrimination task. Inputs and outputs of that Bayesian inference engine are modelled in the *Diagnosis Model* as *Observations* and *Hypotheses* respectively, as exposed in Section 3.3.2.2.

This task is formalised in Table 3.4. The agent starts this task with the goal of diagnosing a new symptom which has been generated in the previous phase (line 9). The first subgoal to achieve that objective starts generating a set of possible hypotheses about the faults that could cause the detected symptom (line 10-12). Then, the agent analyses the possible actions that could be performed to collect more information about the status of the supervised elements (line 13). Hence, a new goal to look for more evidences is created (line 14).

```
1  // B2D2 Agent
2  /* Agent beliefs */
3  causalModel(evidences, contexts, faultsRootCauses, relations).
4  structuralModel(networkElements).
5  diagnosisModel(hypotheses, observations, actors, actions).
6
7  /* Hypothesis Generation - Plans */
8  // Generate initial hypothesis
9  +!diagnose(symptom) :
10         anomaly(symptom)
11         <-
12         generateHypothesis(symptom,faultRootCauses,hypotheses);
13         detectPossibleActions(actions,actors);
14         !lookForEvidences(actions,networkElements).
```

Table 3.4: Hypothesis Generation Task in AgentSpeak language.

3.4.3 Reaching diagnosis conclusions

The *Hypothesis Discrimination* task has the aim of finding the most probable cause of fault for the detected symptom based on the collected information from the network. The agent has to perform test actions on network elements to know its current status and infer

about the probability of fault. The results of those tests are used to feed the *Causal Model* instantiation with the corresponding evidence inputs. To collect this information efficiently, an intelligent test selection strategy must be followed by the agent. We propose the following strategies for the test selection process:

- **Parallel Execution**: Time To Diagnose (TTD) can be minimised if all possible tests are executed at the same time. But, this strategy could overload the network avoiding its correct operation.
- **Random Sequential Execution**: In this case, test are randomly executed one by one until a hypothesis reaches a confidence threshold[18] or all available tests have been executed.
- **Strength-based Execution**: This strategy proposes to used the concept of *strength* of an evidence collected executing a specific test for the discrimination hypothesis process (Kjaerulff and Madsen, 2008). This *strength* concept defines how relevant is the information collected by a test for the current diagnosis case.
- **Cost-based Execution**: Any test action has an associated *cost*[19], such as overloading a network element or consuming some resources. This strategy proposes the *cheapest* tests should be executed first.
- **Strength-Cost Balanced Execution**: This strategy proposes a combination of the previous strategies. Balancing the tests *strength* and *cost*, tests can be executed in parallel or sequentially. In other words, if a test has high *strength* and high *cost*, it could be selected to execute before than others with lower *strength*. In contrast, several *low cost* tests can be executed in parallel if they do not overload the tested elements.

This task is formalised in Table 3.5. The agent starts this task with the goal of looking for evidences to discriminate the cause of fault (line 9 and 14). To get those evidences, it executes testing actions (lines 15-17) following any of the test selection strategies proposed above (line 22-25). The hypothesis set is updated every time new observations are available (lines 18-19) The execution of tests continues until a hypothesis reaches the confidence threshold value or no more tests can be executed (line 10). Then, the diagnosis process finishes and the most reliable hypothesis (i.e. the hypothesis with highest confidence) is picked as conclusion of the diagnosis process (line 12). Notice that several hypotheses with similar confidence values can be selected as conclusions of one diagnosis. This is considered

[18] A notion *confidence threshold* for hypotheses is included in the *Diagnosis Model*.
[19] This *cost* property is included in the *Diagnosis Model*.

to cover the possibility of several faults are occurring simultaneously. Finally, when the agent achieves a conclusion, we consider the fault diagnosis task is finished. The proposed agent architecture could be extended with further autonomic capabilities, such as healing actions to fix the diagnosed faults, but that is out of the scope of this thesis and it is considered as future work.

```
1 // B2D2 Agent
2 /* Agent beliefs */
3 causalModel(evidences, contexts, faultsRootCauses, relations).
4 structuralModel(networkElements).
5 diagnosisModel(hypotheses, observations, actors, actions).
6
7 /* Hypothesis Discrimination - Plans */
8 // Check if the diagnosis is finished
9 +!lookForEvidences(actions,networkElements) :
10         enoughConfidence(hypothesis) | not pending(actions)
11         <-
12         finishDiagnosis(observations,hypotheses,conclusion).
13 // Pick a test plan to execute
14 +!lookForEvidences(actions,networkElements) :
15         pending(actions)
16         <-
17         !getEvidence(action);
18         updateEvidences(observations);
19         disciminateHypothesis(observations,hypotheses,causalModel);
20         !lookForEvidences(actions,networkElements).
21 // Get a new evidence
22 +!getEvidence(actions)
23         <-
24         selectNextAction(actions,observations,relations);
25         !getTestResult(action).
```

Table 3.5: Hypothesis Discrimination Task in AgentSpeak language.

3.5 Case Study

This section presents two different network scenarios where the proposed Agent Architecture has been evaluated and validated. Section 3.5.1 presents a practical experience in the application of a fault diagnosis multi-agent system deployed at Telefónica O2 Czech Re-

public network to manage the faults of an enterprise service. In this real-life application scenario, the *Symptom Detection* task was carried out for human operators due to a stakeholder requirement and only a simplified version of the *Structural Model* was applied due to other knowledge systems were already running in the scenario. Thus, Section 3.5.2 shows a Wireless Sensor Network (WSN) scenario which has been developed to evaluate the proposed architecture in a simulated scenario where a motion detection application is running focusing specially in the detection of symptoms performing ontology-based reasoning using the *Structural Model*.

3.5.1 Internet Business Scenario

Internet Business is a service for business subscribers offered and operated by Telefónica O2 Czech Republic. The service provides secure Internet access to corporate users based on Virtual Private Network (VPN) technology. This system was selected for this case study because it is easy to understand and provides an interesting scenario with enough complexity to evaluate the proposed agent architecture. Figure 3.13 depicts the technical infrastructure required to offer this service. In this infrastructure, subscribers communications equipment connections are realised via a multi Symmetric Digital Subscriber Line (SDSL) to a Digital Subscriber Line Access Multiplexer (DSLAM). Traffic from the DSLAM is transported through the Regional Ethernet Network (REN) to the entry point of the Multiprotocol Label Switching (MPLS) network where MPLS pseudo-wire connections are established to the MPLS provider equipment at a central site. Finally, the customers communication equipment establishes a Point-to-Point Protocol (PPP) session with the Broadband Remote Access Servers (BRASs) at the central site using this transport path. Moreover, the use of technology from different vendors for the different network elements increases the complexity of the diagnosis task for this scenario.

This comprehensive technical solution imposes strong requirements on the inventory and configuration systems. One of the main causes of failure in this service is configuration issues of inventory systems. An inventory system enables the precise identification of the network elements (physical or virtual, including their technical features) that are being used to offer a service to a specific subscriber. The scenario includes a combination of automatic configuration systems based on network events and on a human based configuration of the inventory system (e.g. assigning a new IP address or VLAN). When a configuration change request process is initiated, a service outage or a decrease in the quality of service could occur if the Operation Support Systems (OSS) or inventory systems fail or are missconfigured. Other potential causes of service outages may be hardware or software failures or last mile

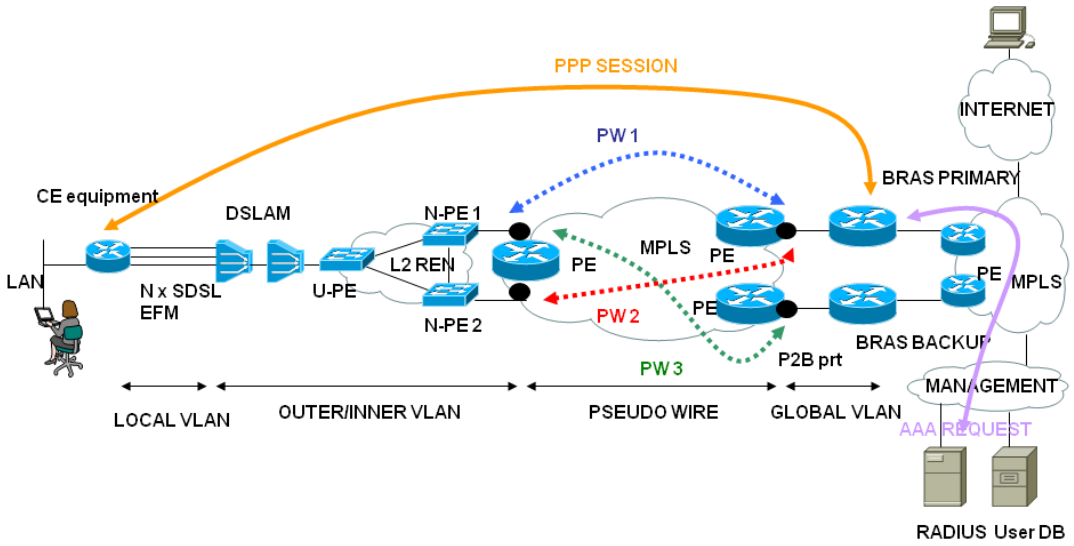

Figure 3.13: Technical infrastructure for providing the Internet Business service.

problems. Hence, in summary, there are many possible root causes of failure in this scenario. The data required to carry out a diagnosis process are geographically distributed. Moreover, that information can be missing, outdated or even unreachable. Thus, this diagnosis scenario is suitable for the application of uncertainty reasoning techniques.

Following the BEAST Methodology nomenclature, the main outcome expected by operators for using the proposed diagnosis system in Telefónica O2 Czech Republic is to decrease the Mean Time to Diagnose (MTTD) (FitzGerald and Dennis, 2008). In addition, a more effective diagnosis system would also increase customer satisfaction and decrease the human resources required for maintenance tasks. The Internet Business service was selected for automated diagnosis because of the high number of customers using this solution, the high number of trouble tickets and the high complexity of the service.

Following the proposed B2D2 Agent Architecture presented in Section 3.4, a MAS was developed and deployed in the OSS servers. That system was implemented using the JADE platform (Bellifemine et al., 2007), which is supported by the BEAST Tool. Outlining the designed solution, the system is composed by a set of agents with different roles, such as, *Interface Agent* which receives requests from operators to diagnose a detected symptom or *Expert Agent* which can executed tests on network elements. But, our interest for this case study is the design of the *Diagnosis Agent* which is the main responsible of carrying out the fault diagnosis process. As mentioned, human operators are able to interact with the Fault Diagnosis MAS to request diagnoses using a web interface. In this request, they provide the detected symptom to an *Interface Agent* that is responsible for collecting data from inventory databases. With that human interaction, the *Symptom Detection* task was delegated to a

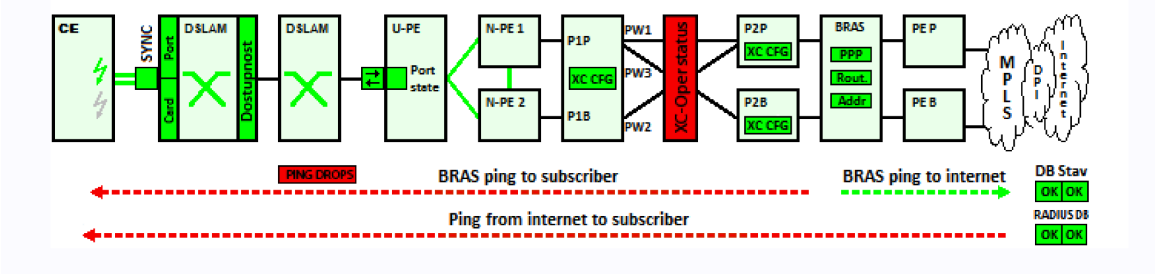

Figure 3.14: Diagnosis results graph.

human operator as requested by stakeholders. Moreover, the inventory systems plays a similar role as the *Structural Model* identifying specific suspicious network elements which simplifies the development of the system. After the required information has been collected, the *Interface Agent* sends a diagnosis request message to the *Diagnosis Agent*, the one that follows the B2D2 Agent Architecture. Detected symptoms are added to the Bayesian inference engine, which instantiates the *Causal Model*, to generate a set of hypotheses that represents possible root causes of the fault. With this step, the *Hypothesis Generation* task finishes. Later, the *Diagnosis Agent* requests *Expert* agents to perform tests. There are six *Expert* agents deployed in this scenario, with each agent specialising in a device type: customer equipment, provider equipment, BRAS, DSLAM, REN and inventory databases. For this task, shell scripts and Simple Network Management Protocol (SNMP) commands are used to interact with network elements. Each time a test is performed, the result is fed back to the *Causal Model* and *Diagnosis Agent* generates new updated hypotheses for the *Hypothesis Discrimination* task.

If one or more hypotheses attain a sufficiently high level of confidence (i.e. probability above 95%), the final conclusion set is shown to the network operator who requested the diagnosis through the web interface. To make the diagnosis results easy to understand, the output of the diagnosis system is presented to the human operator in a diagram, as shown in Figure 3.14. This diagram summarises all the information collected during the diagnosis process.

In this scenario, the *Diagnosis Agent* carries out the hypothesis generation and reasons under uncertainty as shown in Section 3.3.1.2. The *Causal Model* used in this case study is shown in Section 3.3.1.2 as an example. It is composed of 62 different variables: 27 evidence nodes, 18 auxiliary nodes and 17 fault root cause nodes. The knowledge contained in this model was provided by a human operator and its experience was translated in a Bayesian network. The results of this model was validated by human operators rounding the 99% of accuracy. Thus, the application of self-learning algorithms after collecting data from the

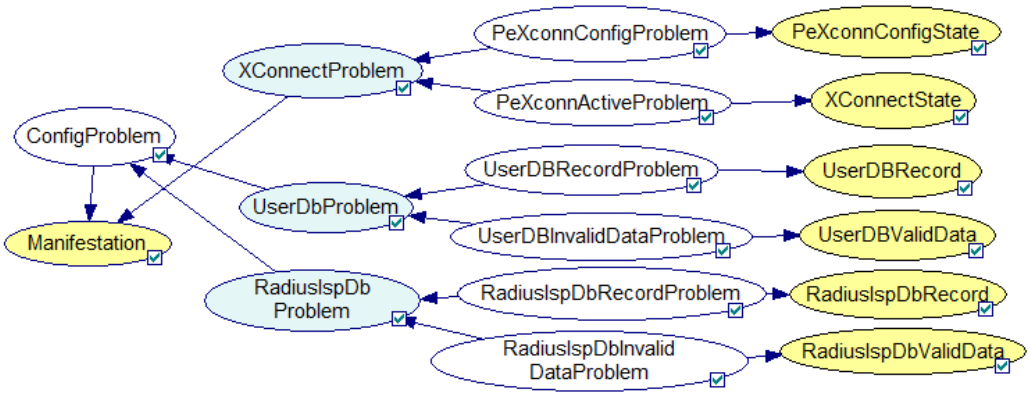

Figure 3.15: A portion of the *Causal Model* used in the case study. The associated CPT of each node is omitted.

network was not required. As the complete *Causal Model* has been shown in Figure 3.4, a small portion is shown in Figure 3.15 to offer more details. In this portion, we highlight the node that represents the detected symptoms (e.g., *Manifestation*), the three nodes that represent fault root causes hypotheses (shown with a blue background) and the six nodes that represent tests results (shown with a yellow background). The rest of the nodes are auxiliary variables. The application of the *Diagnosis Model* shown in Section 3.3.2.2 is exemplified with a diagram of some individuals of the ontology shown in Figure 3.16. The diagram contains three different hypotheses with their corresponding confidence values, one symptom, one observation which is the result of a test execution.

To conclude, in this section we shown how the proposed fault diagnosis model has been successfully applied to an *Internet Business* service. A web interface has been developed to allow human operators to interact with the diagnosis MAS through an *Interface Agent* that receives symptoms and collects information to start a diagnosis. The *Diagnosis Agent* performs Bayesian inference using a *Causal Model* that represents the relationship between network element states and possible failures of the *Internet Business* service. Six different *Expert Agents* perform tests using shell scripts in accordance with the demands of the *Diagnosis Agent*. Finally, the diagnosis result is shown graphically to the human operator, as shown in Figure 3.14.

Evaluation

The presented architecture was evaluated during the period of one and a half years from November 2010 to March 2012. During that period of time, the fault diagnosis MAS was operating and recording data of diagnosis cases stored in a database in internal Telefónica

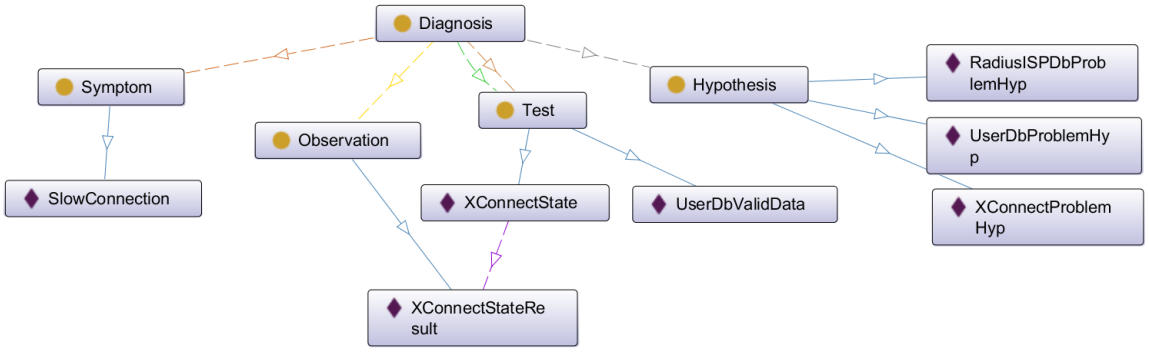

Figure 3.16: Example of Ontology Individuals obtained with the mapping process.

Czech Republic servers. That database contains information about thousands of different diagnosis cases. Every of them contains information about what tests were performed, what information was available during the diagnosis process, what final conclusions were reached at the end of the diagnosis process and other complementary information, such as timestamps or parameters.

The evaluation methodology consisted of two steps. First, we analysed the coverage of the dataset relative to the global problem to check whether these data are sufficiently representative as explained below. Then, we defined several Key Performance Indicators (KPIs) to evaluate the business benefit of the system. To analyse the complexity of the scenario, the entropy of possible diagnosis cases has been calculated and compared with the entropy of each possible root cause[20]. This entropy represents how a same fault root cause can be manifested in the environment for different test results. In other words, different test results can be collected for a single fault type which brings a high level of uncertainty. In this case, 17 different fault types are under consideration. The normalised entropies of fault root causes were compared among them to determine which fault root cause (fault type) is more complex to diagnose. Figure 3.17 shows that some root causes have entropy values close to zero, because these fault types almost always present the same symptoms. In contrast, other fault root causes exhibit high entropy because these fault types can produce many different observations, which increases the diagnosis complexity.

To graphically represent all diagnoses stored in the database, a Sammon mapping algorithm (Sammon, 1969) has been used to represent the diagnoses in a two dimensional graph in Figure 3.18. Using this algorithm, the relative euclidean distance among all diagnosis cases is relatively maintained. As shown in Figure 3.18, diagnosis cases with the same conclusions are clustered in the graph. To highlight these clusters, all of them have

[20] As proposed in metric "M4 - Heterogeneity of Diagnosis Cases" of the BEAST Methodology in Section 2.3.4

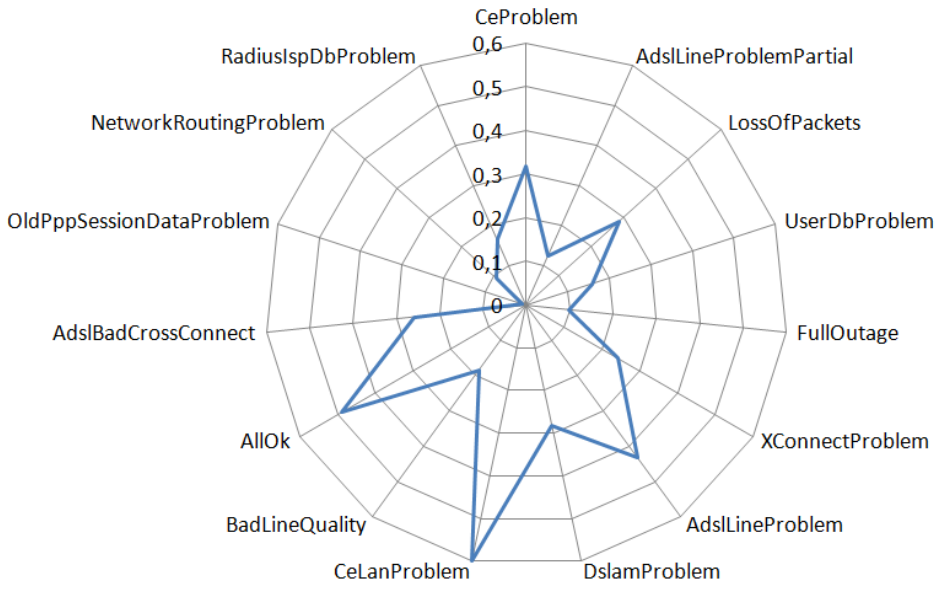

Figure 3.17: Normalised entropy of various root causes of faults.

been rounded and labelled properly. Furthermore, we can observe that the clusters area are directly related with the entropy value of the fault root causes which is associated, which confirms that the Sammon mapping algorithm is suitable to represent the complexity of fault root causes. To interpret this graph, note that two diagnosis cases that are graphically in the same place in Figure 3.18 represent cases with the same symptoms and the same final hypotheses, i.e., the euclidean distance between those cases is close to zero.

The duration of diagnosis process[21] is presented as a histogram[22] in Figure 3.19. The result of this histogram can be understood as a normal distribution with a mean value of 48.365 seconds and a standard deviation of 7.462 seconds.

After this analysis of the data collected, several KPIs were defined to evaluate the business outcomes of the system. We have two measures of these KPIs, after one month of the system deployment and eighteen months later (i.e. until the dataset is analysed), as shown in Table 3.6 . Notice the KPIs values are meaningful as the available dataset is sufficiently representative relative to the global problem and the system was enough time executing to know its global performance.

KPI1 is used to measure the usage of the system by human operators, i.e., the acceptance rate of the diagnosis system. This KPI was initially 24.74% and posteriorly, it increased to 92.00%. In other words, human operators use the developed fault diagnosis systems in the

[21] As proposed in metric "M5 - Time To Diagnose" of the BEAST Methodology in Section 2.3.4
[22] Notice that Y axis of Figure 3.19 is on a logarithmic scale.

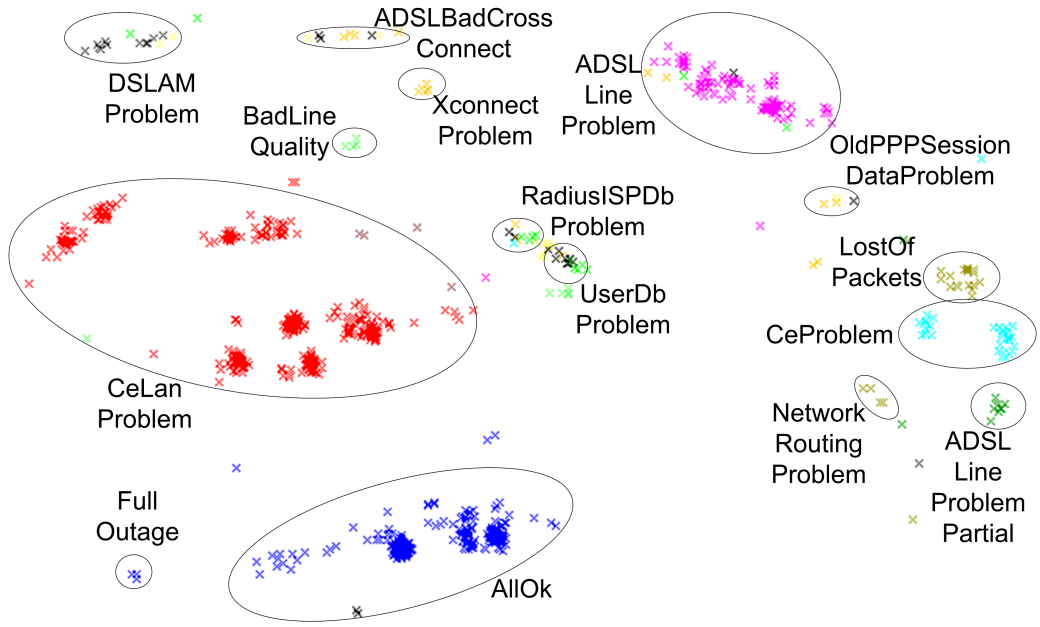

Figure 3.18: Fault root cause clusters.

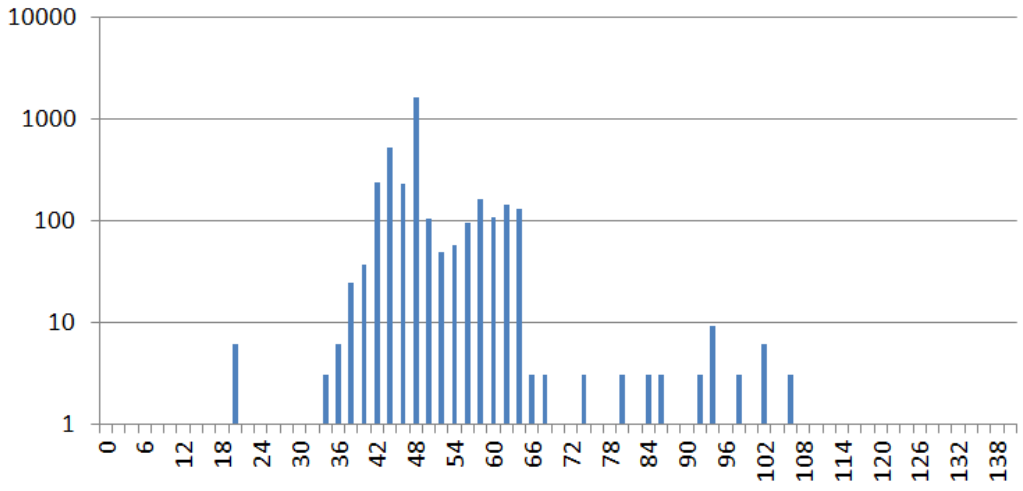

Figure 3.19: Histogram of diagnosis duration (in seconds).

KPIs	1st Month	18th Month
KPI1	24.74%	92.00%
KPI2	9.51 hours	5.2 hours
KPI3	0.56 hours	0.37 hours

Table 3.6: System KPIs

92% of the cases, what can be considered as a successful acceptance rate. Operators explain this increase due to initial reservations to use a new system, but the final usage rate shows a massive acceptance level.

KPI2 is used to measure the average incident solution time (i.e., the diagnosis and repair time). Initially, this KPI was 9.51 hours initially. It decreased to 5.2 hours, representing 45.32% time savings. That time reduction was not only produced by the application of the system. During the same period of time, other optimization techniques where applied for the maintenance of the network.

KPI3 is used to measure the mean time before a work order is created (i.e., the diagnosis time). This KPI was initially 0.56 hours, and the MAS system has decreased this metric to 0.17 hours, representing 33.93% time savings. This time includes the TDD shown in Figure 3.19 and the time between the work order is created and an operator requests a diagnosis process to the system. We can consider the time to start a diagnosis by an operator (using or not the system under evaluation) has not changed in the period of time. Thus, this time saving is associated directly with the use of the developed fault diagnosis system.

Summarising, the performance of the system has been measured with several KPIs that demonstrate the business outcomes of the diagnosis system. Furthermore, the diagnosis results stored in a database during the one and a half years of operation of the diagnosis MAS have been analysed to measure the entropy of all available diagnoses (see Figure 3.17) and to graphically represent the similarity among the diagnoses (see Figure 3.18). Based on those findings, the validity of the proposed B2D2 Agent Architecture has been demonstrated.

3.5.2 Wireless Sensor Network Scenario

This section describes the simulated WSN scenario that has been developed to validate some components of the agent B2D2 Agent Architecture, proposed in Section 3.4, which

were not fully validated in the previous case study. As mentioned in Section 3.5.1, the *Symptom Detection* task was delegated to human operators due to a stakeholder requirement. Moreover, some of the knowledge that is covered by the *Structural Model* was allocated in the inventory systems that were already running in the scenario. Thus, only a simplified version of the *Structural Model* was used. For these reasons, this complementary case study is exposed focusing on ontology-based reasoning for symptom detection.

The simulated application over the WSN is a motion detection application where sensors can detect any motion in a range and a random mobile target is in the simulated environment. The developed simulation is built on the top of the MASON simulation framework (Luke et al., 2005) and it is available in a public Github Repository[23]. In the simulation, the WSN topology[24] is automatically generated with two parameters: number of ZigBee End Devices (ZEDs) and number of ZigBee Routers (ZRs). One ZigBee Coordinator (ZC) is deployed as sink node in the topology and the B2D2 Agent is executed in that node which we consider without any power or computational restrictions. The ZR nodes route all packages received from the ZEDs, i.e. motion detection notifications, to the sink node, i.e. the ZC. All ZED devices connected to the same ZR device conform a *cluster*, and that ZR is known as *cluster head*.

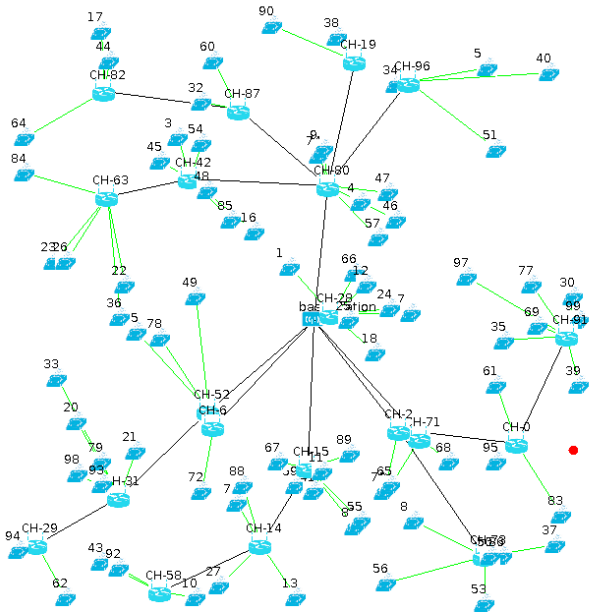

Figure 3.20: Snapshot of the simulated WSN scenario.

[23]WSN Simulation Public Repository: https://github.com/gsi-upm/shanks-wsn-module
[24]We are using the ZigBee Alliance nomenclature: http://www.zigbee.org/

Thus, when the simulation starts, the network topology is generated in three steps. Firstly, the ZC node (sink node) is placed in a fixed position and all ZR and ZED nodes are randomly placed. Secondly, ZR and ZED nodes are moved until they have at least one router (or directly the ZC sink node) in range. Finally, Dijkstra's algorithm (Skiena, 1990) is used to create routes (i.e. links) between ZR nodes using minimum power consumption criteria based on accurate power consumption values provided by Landsiedel et al. (2005) and ZED nodes are linked to the closest ZR node. An example of the result of this process is illustrated in Figure 3.20.

The behaviour of the simulated devices have been implemented following the specifications of MICAz device[25]. The emulated MICAz devices are equipped with emulated Panasonic IR Motion Detection sensors[26] which detect the mobile target. When the target is in range, the detecting nodes generate a message to notify that detection. To reduce the messaging cost of the simulated scenario, no ACK messages are sent to confirm the reception. So, some messages can be lost due to several causes, such as network overflow or weather/noise conditions, as exposed below in the evaluation of this case study. These messages are forwarded including trace information and some data about the nodes in the path. In other words, every node which receives a message adds data about the node itself, such as, node id, message id, cpu load or memory load. Thus, the message received by the sink node contains information about all nodes which have forwarded the original message. That information is used by the Fault Diagnosis Agent to update its *Structural Model* and to detect symptoms. The packages between nodes are parsed as plain text and sent using the maximum throughput in ZigBee WSN obtained by Burchfield et al. (2007). When the WSN is ready to work, the mobile target starts its random movement generating traffic over the network. Then, the Fault Diagnosis Agent that is running in the ZigBee Coordinator node, i.e. the sink node, processes the incoming information about the network status and topology and updates the structural network model in real-time.

The B2D2 Agent deployed in the ZC uses a *Diagnosis Model* which includes the *Structural Model* that uses the INDL language to describes the network elements emulated in this WSN scenario, such as *ZigBeeSensorNode* or *ZigBeeMessage*, network properties, such as *StarTopology* or *TreeTopology*, or the possible observation detected from the network, such as *LostMessage*. For further information, please refer to the complete B2D2 WSN Ontology[27]. The B2D2 Agent uses this model in combination with a set of SPIN rules to reason about the network elements and its properties. An example of one of those rules is shown in

[25]MICAz Datasheet: http://www.memsic.com/wireless-sensor-networks/
[26]IR Motion Detection Sensor Datasheet: https://www3.panasonic.biz/ac/e/control/sensor/human/wl/index.jsp
[27]B2D2 WSN Ontology Specification: `http://www.gsi.dit.upm.es/ontologies/b2d2/wsn`

```
1 CONSTRUCT {
2     ?this a wsn-ndl:ZigBeeLeafRouter .
3 }
4 WHERE {
5     ?this a wsn-ndl:ZigBeeRouter .
6     {
7         SELECT ?this
8         WHERE {
9             ?path a wsn-ndl:RoutePathLink .
10            ?this nml-base:isSink ?path .
11        }
12        GROUP BY ?this
13        HAVING (COUNT(?path) = 0)
14    } .
15 }
```

Figure 3.21: Example of SPIN rule to detect edge routers.

Figure 3.21. This example is used to detect edge routers nodes in the network dynamically. The rule picks a router (line 5) which does not receive any message from other routers (lines 7-13) and labels it as a *ZigBeeLeafRouter* (line 2). Another example of these rules is shown in Section 3.4.1 and the rest of them are available in the Github repository of the simulation[28]. The *Causal Model* developed for this scenario is available in the Github repository too and its structure is shown in Figure 3.22. For Bayesian reasoning and learning, the B2D2 Agent uses SMILE library (Druzdzel, 1999). The reminder reasoning capabilities of the agent, such control loop or plan selection strategy, are implemented with a MASON agent (Luke et al., 2005), which is the simulation framework used for implementing this case study as mentioned previously.

Evaluation

The outcomes of the B2D2 Agent in this WSN scenario are evaluated in two different ways as exposed below. Firstly, the number of detected lost messages by the B2D2 Agent are analysed comparing different network configurations to check if the simulation model generates consistent results during the *Symptom Detection* task. Secondly, we have used

[28]Other examples of SPIN rules can be found in our Github public repository: https://github.com/gsi-upm/shanks-wsn-module/tree/master/src/main/resources/rules

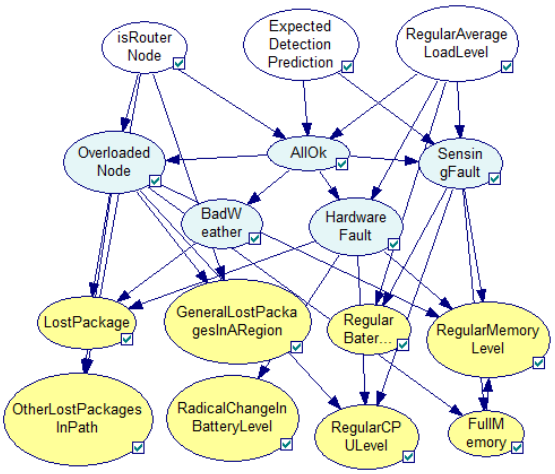

Figure 3.22: Structure of the *Causal Model* developed for this case study.

a dataset of diagnosis cases, based on the simulated scenario, to analyse the benefit of reasoning with the *Structural Model* and the *Causal Model* or only with the *Causal Model*.

The data collected for the evaluation of the *Symptom Detection* task was gathered executing 200 simulated scenarios with different parameters. We simulated a WSN where nodes were deployed on a two-dimensional space of 100 square meters during 10 minutes[29] with one mobile target with 5 kilometres per hour speed that moved randomly in the simulation space. The perception range of the motion detectors was 5 meters. We consider the simulation was indoor for radio range of MICAz devices. This setup generates around 8000 messages that should be received in the ZC sink node in every execution of the simulation scenario. Under these conditions, we designed our experiments to quantify the behaviour of the B2D2 Agent running in the ZC sink node. Specifically, we measured the number of lost messages were detected[30].

During the execution of the 200 simulated scenarios, a dataset was collected to analyse the results of the *Symptom Detection* task. The results of the simulations have been analysed comparing the ratio of detected lost messages and some metrics of the WSN topology. Figure 3.23 shows the ratio of lost messages to the number of sensors deployed in the simulation. A gradual increase of the ratio can be observed with the number of deployed sensors. Other interesting relation is the ratio of lost messages compared with the ratio of routers to total sensors, shown in Figure 3.24. A decrease is detected when the number

[29]The simulation manages the time for tasks such as movement (with a specific speed) or devices throughput.

[30]As no ACK messages are sent in the simulated scenario, if a node cannot handle a received message, it will be lost.

of routers is similar to the number of deployed sensors, i.e. there are small sensor clusters in the network topology. Finally, we analyse the lost message ratio with the ratio of edge routers[31], i.e. the first router in a path to the sink node. Figure 3.25 shows a similar behaviour independently of the ratio of edge routers in the network. Finally, the ratio of lost messages to correctly received messages in the ZC sink node is 1.63%, with an average of 125.96 lost messages detected and 7707.06 received messages in the ZC sink node in every execution. In a real WSN this lost rate could be considered too high, but remember that we are considering no ACK messages to incentive the lost of messages for testing purposes. In conclusion, the results show that B2D2 Agent performs a consistent *Symptom Detection* task using SPIN rules to infer about the network elements in this error-prone scenario.

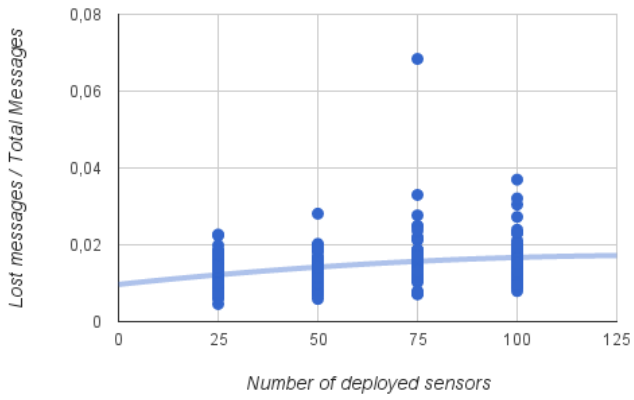

Figure 3.23: Ratio of lost messages to number of deployed sensors.

As mentioned previously, only a simplified version of a *Structural Model* was used in the previous case study. Thus, a complementary evaluation has been performed focusing on the ontology-based reasoning process for symptom detection. The aim of this evaluation is to analyse the improvement of using the *Structural Model* with a post-processing rule-based inference process to feed the *Causal Model* compared with the use of an isolated *Causal Model* without rules (i.e. no *Structural Model* and no rules).

First of all, we have calculated the entropy of the fault root causes of the scenario under consideration[32]. As shown in Figure 3.26, there is a variety of entropy among the different fault root cause selected for this study, which means that some diagnosis cases will be more observations than others to reach a conclusion.

[31] This edge routers are detected by the SPIN rule shown in Figure 3.21.

[32] As proposed in metric "M4 - Heterogeneity of Diagnosis Cases" of the BEAST Methodology in Section 2.3.4

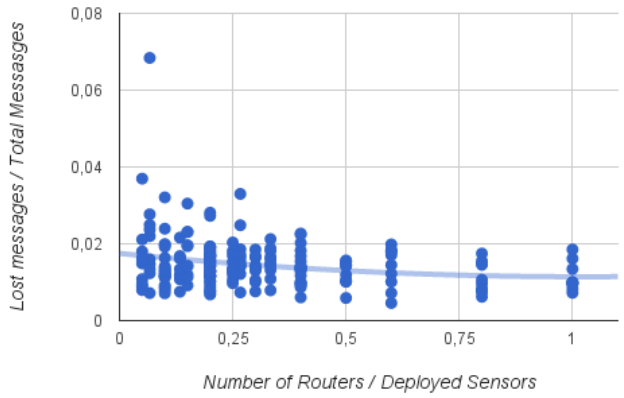

Figure 3.24: Ratio of lost messages to ratio of number of routers and number of deployed sensors.

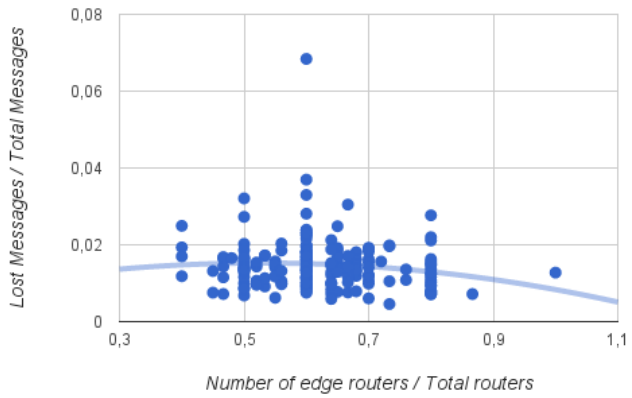

Figure 3.25: Ratio of lost messages to ratio of number of edge routers and number of total routers.

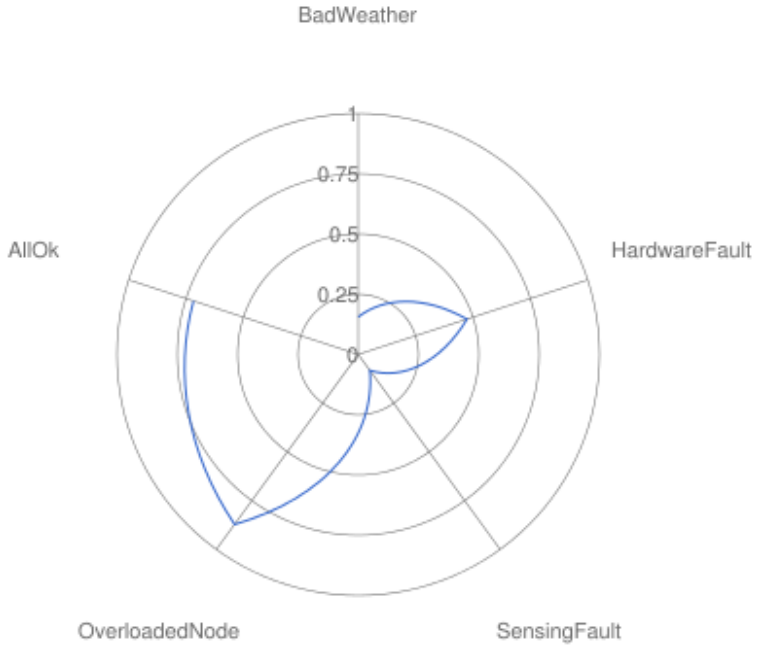

Figure 3.26: Normalised entropy of the fault root causes of the simulated WSN scenario.

To perform this evaluation, we trained two different *Causal Models* applying self-learning algorithms: one with all information available after the rule-based inference process (labelled as *SM+CM* in the figures) and the other with the information available directly from the network, without post-processing (labelled as *CM* in the figures). Both *Causal Models* have been validated with three different levels of uncertainty: no missing data, 25% of missing attributes and 50% of missing attributes. The global success rate for those cases are shown in Figure 3.27 where we can observe an improvement around a 3% in all cases.

Moreover, we have analysed the success rate per fault root causes to know in what cases the rule-based inference process is improving the accuracy of the diagnosis conclusion. Figures 3.28, 3.29 and 3.30 present the accuracy of both alternatives per fault root cause[33] with (i) no missing data, (ii) 25% of missing attributes and (iii) 50% of missing attributes, respectively. As we can observed in the figures, there are some causes which its accuracy are very related to the rule-based inference process with an accuracy close to zero for the CM alternative. While other fault root causes show a similar accuracy for both alternatives in the three scenarios with different level of uncertainty. Therefore, we can conclude that

[33]These metrics have been not presented while proposing the BEAST Methodology in Section 2.3.4 because the metrics of that section are not defined to compare the accuracy of two alternative reasoning processes. The reminder metrics of this section has been defined for that specific objective.

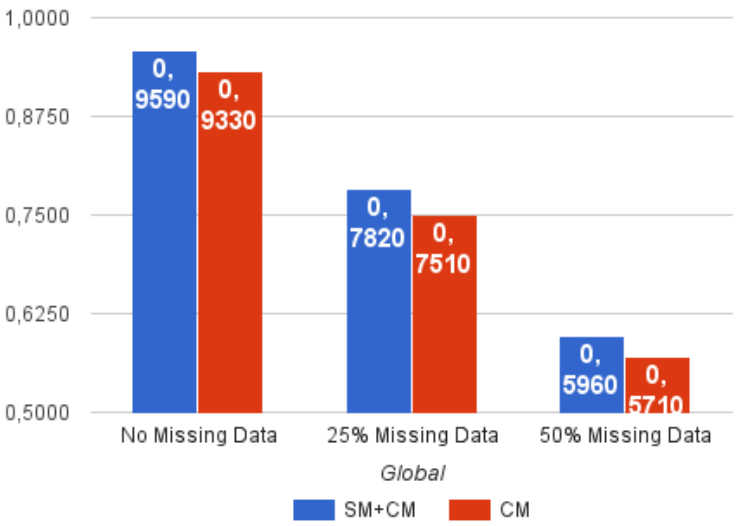

Figure 3.27: Global Success Rate of the Validation Process.

the use of a complete *Structural Model* is important to reach correct conclusions for specific faults, while for others is not so relevant obtaining similar results with and without an *Structural Model* and a rule-based inference process.

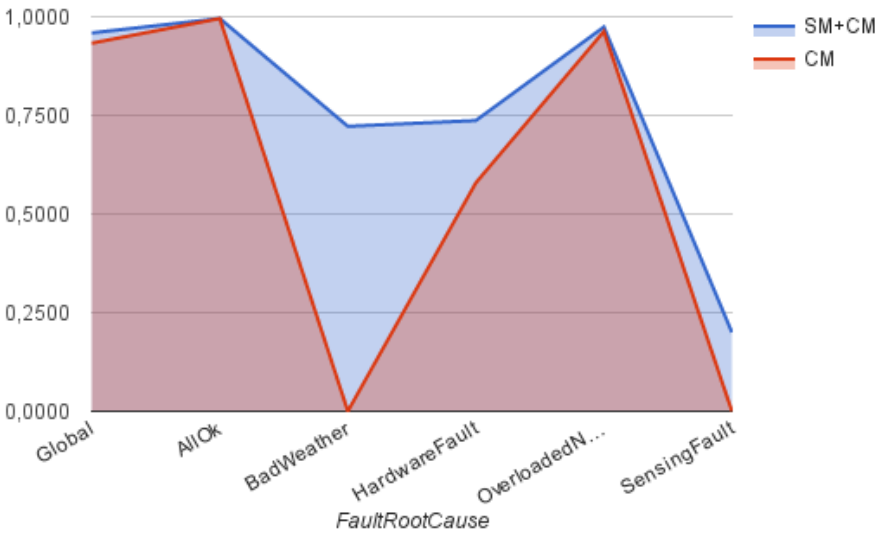

Figure 3.28: Success Rate with no missing data.

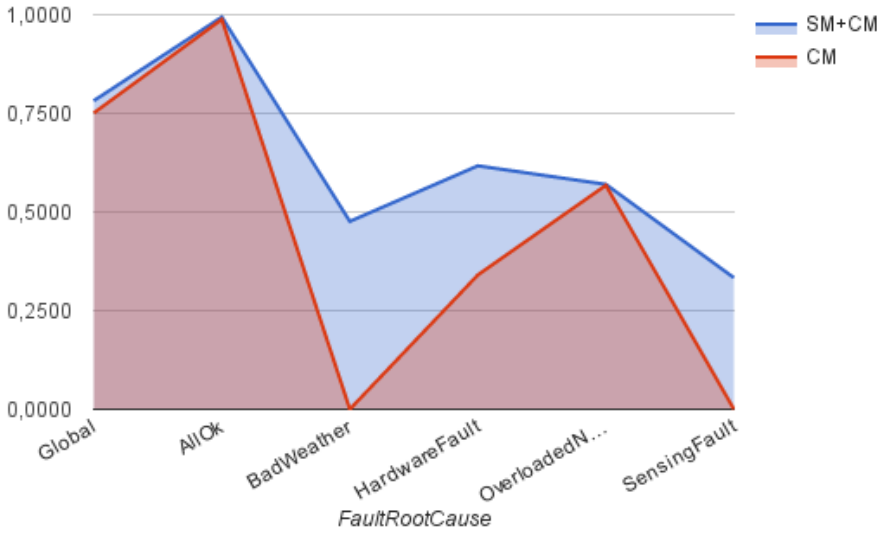

Figure 3.29: Success Rate with 25 % missing data.

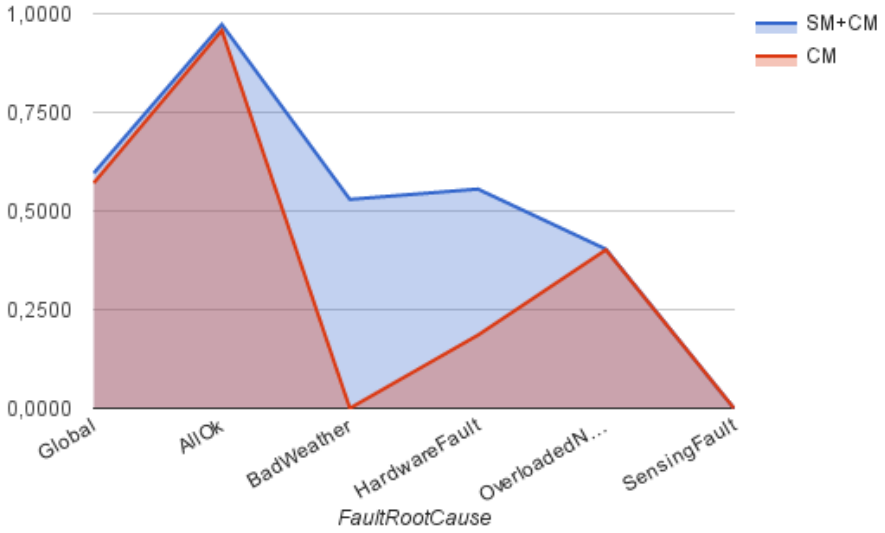

Figure 3.30: Success Rate with 50 % missing data.

3.6 Summary

This chapter has proposed an Autonomic Fault Diagnosis Agent Architecture, named BDI for Bayesian Diagnosis (B2D2) Agent Architecture. The proposed architecture uses several knowledge models that are combined in the proposed B2D2 Diagnosis Model[34]. This model covers the domain knowledge required to carry out an autonomic fault diagnosis process including a *Structural Model*, which describes network elements and their relations and properties, and a *Causal Model*, which relates the observations collected from the network with their possible fault root causes. The agent follows a Task Model which divides the diagnosis process in several tasks which are formalised in the chapter using the *AgentSpeak* language.

The proposed agent architecture has been validated in two complementary case studies. One of them is a practical experience in the application of a fault diagnosis multi-agent system deployed at Telefónica O2 Czech Republic network to manage the faults of an enterprise service. The data collected during one and a half years of system execution in a real telecommunication network has been used to analyse and validate the proposed architecture. The second case study is a simulated WSN scenario which complements the evaluation process of the first case study focusing on the automation of the symptom detection task, which was manually performed in the first case study.

Finally, the proposed B2D2 Agent Architecture defines mechanisms to perform autonomic fault diagnosis of telecommunication networks. The proposed B2D2 Diagnosis Model covers the knowledge domain required for the autonomic diagnosis process. Nevertheless, the execution of monolithic agents is not suitable for the Future Internet due to the presence of federated domains and scalability constraints. Therefore, the coordination mechanisms proposed in Chapter 4 allow different agents to discuss the possible fault root cause to perform the discrimination process in a distributed way, which allows the execution of the proposed agent architecture in federated domains solving possible scalability problems.

[34]Diagnosis Model Ontology Specification: `http://www.gsi.dit.upm.es/ontologies/b2d2/diagnosis`

CHAPTER 4

Coordination for Autonomic Fault Diagnosis in Federated Domains

This chapter presents an argumentation framework as coordination mechanism to perform distributed autonomic fault diagnosis tasks in federated domains. The argumentation framework is used by agents to discuss uncertain information for discriminating hypotheses of fault. The arguments of that framework are composed of probabilistic statements, whose aim is to handle the uncertainty of the diagnosis process under access restriction situations. The framework is applied in a coordination protocol which defines different phases for the distributed diagnosis process. Moreover, an extension of the B2D2 Agent Architecture exposed in the previous chapter is presented to include in the agent the argumentation capability required to apply the proposed argumentation framework. The proposed argumentation framework has been evaluated measuring the correctness of the conclusion obtained after the argumentation process. This evaluation process has been performed in a simulation environment in which the conditions of federated domains were reproduced to assess the validity of the proposed argumentation framework as coordination mechanism.

4.1 Introduction

One of the most important aspects of the Future Internet is the federation of the autonomic systems to enable optimal end-to-end service provisioning (Müller, 2012). That federation is defined as the agreement between different autonomic systems belonging to different domains which have to cooperate to achieve an objective (Laurent et al., 2013). That agreement capability of the federated systems is considered a key property to solve scalability issues in large scale systems in the Future Internet (Cerf, 2013). For instance, the fault diagnosis solution proposed in the previous chapter could not monitor and diagnose the whole Internet, because the knowledge models required to build that global solution would be unmanageable and would require constant updates and extensions to keep the pace of the growing of the network. Moreover, the computational resources required to reason for a global diagnosis process in a single monolithic agent would be huge to handle the global domain models. Thus, the B2D2 Agent Architecture for Autonomic Fault Diagnosis, proposed in Chapter 3, could not be considered as completely suitable for the Future Internet without coordination mechanisms to enable that capability of agreement and ensure scalability of the solution.

Several coordination mechanisms have been considered to build that cooperative capability for the proposed agent architecture during the development of this thesis. Our first attempt was to apply distributed Bayesian inference techniques, such as Multiply Sectioned Bayesian Networks (MSBNs), Distributed Perception Networks (DPNs) or Prior/Likelihood Decomposable Models (PLDMs). But, some aspects make them not completely suitable for the considered federated network scenario. Some scalability and flexibility issues would have implied an extra management cost for the final solution if we had applied them. For further information about these techniques, please refer to the work of Méndez (2011). Then, we looked for alternative technique which fits with our agent architecture. We found argumentation theory is commonly applied as communication mechanism to reach agreements among agents (Walton, 2009). But, we did not find an argumentation framework fully compatible with the uncertainty management of the diagnosis process supported by the B2D2 Agent Architecture. Therefore, this chapter proposes an argumentation framework which uses arguments composed of probabilistic statements to represent the uncertainty of the knowledge they contain. Based on that framework, a coordination protocol is proposed. That protocol enables agents to cooperate during the diagnosis process until they reach an agreement about the most probable fault root cause. The protocol establishes different phases to conform a coalition of agents, to discuss among them about a specific diagnosis case, and to get conclusions for the diagnosis process.

The rest of this chapter is structured as follows. Firstly, Section 4.2 discusses related

works in the research field of application of argumentation techniques in multi-agent systems. Section 4.3 presents the formal definition of the argumentation framework. Section 4.4 proposes the coordination protocol based on the proposed framework for a distributed autonomic fault diagnosis. Section 4.5 introduces an extension of the B2D2 Agent Architecture for adding the argumentative capability required to apply the argumentation framework as coordination mechanism. Section 4.6 shows a case study where the proposed coordination mechanism is applied in a federated network scenario and exposes the evaluation of the proposed argumentation framework. Finally, Section 4.7 presents some concluding remarks of this chapter.

4.2 Related Work

Argumentation is a crucial communicative activity in society (Moor and Aakhus, 2006). Consequently, the argumentation theory has many applications in both theoretical and practical works. It is defined as the interdisciplinary study of the method to obtain conclusions through logical reasoning (van Eemeren et al., 1996) and has been studied in many different fields, such as rhetoric (Wallace, 1963; Perelman and Olbrechts-Tyteca, 1969), philosophy (Toulmin, 2003), law (Feteris, 1999) computer science (Bench-Capon and Dunne, 2007) or artificial intelligence (Walton, 2009). It covers situations such as debate and negotiation, which are both directed toward achieving valid conclusions and/or agreements. In the literature, we find Dung's work (Dung, 1995) as one of the most influential approach to apply argumentation in the artificial intelligence field. However, other approaches are widely used too, as summarised in Table 4.1.

For further information about the application of argumentation techniques in multi-agent systems, please refer to a systematic review about that topic published during the development of this thesis (Carrera and Iglesias, 2015). That review was conducted following the principles provided by Kitchenham and Charters (2007). Intensive searches in the main electronic scientific databases were performed in July 2012, March 2013 and January 2015 to ensure the coverage of the systematic review. Based on a set of inclusion and exclusion criteria, 64 studies were included and properly analysed. The extraction and synthesis process was established to answer the most relevant aspects of the included studies from the authors point of view. The results are clearly presented, both graphically and literally. Some of the concluding remarks of that review are summarised below.

Regarding the goal of the dialogue among agents, the most common objective of an argumentative system is to decide the best course of action. But, the goal of achieving

Argumentation Framework	Studies
Dung's Argumentation Framework (DAF) (Dung, 1995)	(Tannai et al., 2011; Caiquan et al., 2010; Yuan et al., 2011; van der Weide et al., 2011; Amgoud and Serrurier, 2007; Huang and Lin, 2010; Xiong et al., 2012; Rowe et al., 2012; Vreeswijk, 2005; Wang et al., 2014; Bedi and Vashisth, 2014)
Preference-based Argumentation Framework (PAF) (Amgoud and Cayrol, 1998)	(Obeid and Moubaiddin, 2009; Bulling et al., 2008; Amgoud and Serrurier, 2008; Amgoud et al., 2005; Amgoud, 2006)
Value-based Argumentation Framework (VAF) (Bench-capon et al., 2002)	(Heras et al., 2013b,a; dAvila Garcez et al., 2014; Thomopoulos et al., 2014)
Assumption-based Argumentation Framework (AAF) (Bondarenko et al., 1993)	(Gaertner and Toni, 2007, 2008; Fan et al., 2014, 2013)
Argumentation-based Negotiation (ABN) (Rahwan et al., 2003)	(Ye et al., 2010; Hsairi et al., 2006; Zhang et al., 2012; El-Sisi and Mousa, 2012; Harvey et al., 2007; Brandao Neto et al., 2013; Xue-jie et al., 2013; Sierra et al., 1998; Amgoud et al., 2008; Morge and Beaune, 2004; Alonso, 2004)
Three-Layer Argumentation Framework (TLAF) (Maio and Silva, 2012)	(Maio et al., 2011; Maio and Silva, 2014)
Logic Programming without Negation as Failure (LPwNF) (Kakas et al., 1994)	(Moraitis and Spanoudakis, 2007; Kakas and Moraitis, 2003)

Table 4.1: Studies per argumentation framework.

Behaviour	Studies
Collaborative	(Hsairi et al., 2006; Moraitis and Spanoudakis, 2007; Yuan et al., 2009; Obeid and Moubaiddin, 2009; Liu et al., 2010; Ge et al., 2010; Hsairi et al., 2010; Ye et al., 2010; Maio et al., 2011; Zhang et al., 2012; El-Sisi and Mousa, 2012; Aulinas et al., 2012; Chow et al., 2013; Grando et al., 2012; Harvey et al., 2007; Marreiros et al., 2005; Rowe et al., 2012; Velaga et al., 2012; Letia and Groza, 2012; Amgoud and Serrurier, 2008; Monteserin and Amandi, 2011; Wang et al., 2010; Das, 2005; Bulling et al., 2008; Amgoud and Serrurier, 2007; Huang and Lin, 2010; Janjua and Hussain, 2012; Tao et al., 2014; Wang et al., 2014; Kakas and Moraitis, 2003; Fogli et al., 2013; Fan et al., 2014; Bedi and Vashisth, 2014; Ferrando and Onaindia, 2013; Maio and Silva, 2014; Vicari et al., 2003; Tang and Parsons, 2005; Morge and Beaune, 2004)
Competitive	(Yuan et al., 2011; Keppens, 2011; Heras et al., 2013b,a; Brandao Neto et al., 2013; Xue-jie et al., 2013; Pashaei et al., 2014; Sierra et al., 1998; Amgoud and Parsons, 2002; Gaertner and Toni, 2007; Caiquan et al., 2010; Tannai et al., 2011; Xiong et al., 2012; Wardeh et al., 2012; van der Weide et al., 2011; Vreeswijk, 2005)

Table 4.2: Studies per agent level behaviour.

agreements or reasonable settlements is quite popular too. Paying attention to the agents' interactions in their society, we highlight that a collaborative behaviour is observed more often than a competitive behaviour. Nevertheless, it is clear that argumentation techniques have been applied successfully in both cooperative and competitive environments, as shown in Table 4.2.

Focusing on the application field of argumentation techniques in real-life problems, we highlight their application in e-commerce and virtual organisations. For e-commerce, they are used for tasks such as finding potentially interesting products (Huang and Lin, 2010), making deals with providers and customers (Ge et al., 2010) or negotiating supply strategies (Wang et al., 2010). For virtual organisations, we remark their application for dealing with incomplete and conflicting information (Janjua and Hussain, 2012), analysing emotional factors (Marreiros et al., 2005), deciding benefit concessions (Wardeh et al., 2012), approving credit assignment (Pashaei et al., 2014) or building reputation models (Hsairi et al., 2010).

Another interesting finding of the systematic review is the evolution of the studies from pure theoretical studies to prototypes or real-life applications. The number of prototypes/applications described in the period from 2011 to 2015 depicts this technology offers suitable

Year	Theory	Prototype	Application
1998-2005	11	0	1
2006-2010	15	5	1
2011-2014	9	17	5

Table 4.3: Maturity level of included studies per year.

approaches for solving real-life problems, as shown in Tables 4.3 and 4.4. Based on this evolution, we can place the argumentation technology for multi-agent systems in an early third stage of the maturation classification proposed by Redwine and Riddle (1985). That is because, as an interdisciplinary field of study, argumentation has a robust and solid theoretical background and it is beginning to be applied in industrial applications. In conclusion, the interest for applying of argumentation theory in multi-agent systems has increased in recent years (Maudet et al., 2007).

Although the application of argumentation techniques in a MAS can be an appropriate approach to solve many different problems, these techniques by themselves do not solve all of the issues that a MAS has to solve for facing real-life problems. Hence, authors decide to combine argumentation frameworks with other reasoning techniques, such as case-based reasoning (Heras et al., 2013b) or rule-based reasoning (Hartfelt et al., 2010). Following this approach, we propose the use of an argumentation framework compatible with the reasoning techniques used in the B2D2 Agent Architecture presented in the previous chapter. Specially, the argumentation framework will require an appropriate management of the uncertainty of the diagnosis process. In the proposed architecture, the agent applies a Bayesian reasoning technique to reason under uncertainty during the hypothesis discrimination phase. Thus, the argumentation framework must be compatible with that reasoning process of the agent.

Maturity level	No. studies	Studies
Theory	35	(Yuan et al., 2009; Liu et al., 2010; Wang and Luo, 2010; Caiquan et al., 2010; Ge et al., 2010; Hsairi et al., 2006, 2010; Obeid and Moubaiddin, 2009; Yuan et al., 2011; Maio et al., 2011; Das, 2005; van der Weide et al., 2011; Bulling et al., 2008; Amgoud and Serrurier, 2007; Keppens, 2011; Rowe et al., 2012; Gaertner and Toni, 2008; Vreeswijk, 2005; Amgoud and Serrurier, 2008; Brandao Neto et al., 2013; Tao et al., 2014; Amgoud et al., 2000; Kakas and Moraitis, 2003; dAvila Garcez et al., 2014; Sierra et al., 1998; Fan et al., 2013; Amgoud and Parsons, 2002; Amgoud et al., 2008; Amgoud and Prade, 2009; McBurney et al., 2003; Amgoud et al., 2005; Amgoud, 2006; Tang and Parsons, 2005; Morge and Beaune, 2004; Alonso, 2004)
Prototype	22	(Moraitis and Spanoudakis, 2007; Ye et al., 2010; Wang et al., 2010; Monteserin and Amandi, 2011; Heras et al., 2013b; Huang and Lin, 2010; Janjua and Hussain, 2012; Xiong et al., 2012; Zhang et al., 2012; El-Sisi and Mousa, 2012; Heras et al., 2013a; Grando et al., 2012; Harvey et al., 2007; Wardeh et al., 2012; Xue-jie et al., 2013; Wang et al., 2014; Fogli et al., 2013; Fan et al., 2014; Bedi and Vashisth, 2014; Ferrando and Onaindia, 2013; Maio and Silva, 2014; Pashaei et al., 2014)
Application	7	(Tannai et al., 2011; Gaertner and Toni, 2007; Aulinas et al., 2012; Chow et al., 2013; Velaga et al., 2012; Thomopoulos et al., 2014; Vicari et al., 2003)

Table 4.4: Maturity level.

4.3 B2D2 Argumentation Framework

This section proposes the B2D2 Argumentation Framework. This framework is used by agents to discriminate the most probable cause of fault during a distributed diagnosis process in federated domains. In those federated domains, every agent manages its own domain and has a partial view of the global problem. This ability to divide the global problem into domains combined with coordination mechanisms ensures the scalability for large scale systems (Cerf, 2013). Therefore, the coordination mechanism provided by this argumentation framework is required to ensure that scalability of the B2D2 Agent Architecture presented in Chapter 3. In that architecture, the model applied during the hypothesis discrimination phase to reason under uncertainty is the *Causal Model*. This model is used to update the hypothesis set every time new observations are collected from the network. The information used as input and output of the *Causal Model* is used in the proposed argumentation framework to build arguments keeping the uncertainty management capability offered by the model. Thus, the argumentation framework exposed in this section requires that every agent has a *Causal Model* to build and process arguments.

All in all, the formal definition of the argumentation framework is presented in Section 4.3.1, and the possible relations that can exist between arguments of this framework are exposed in Section 4.3.2.

4.3.1 Framework Definition

The proposed argumentation framework relies on the idea of probabilistic statements built using a *Causal Model*. That model is composed by a set of variables and their conditional probabilistic dependencies, as explained in Section 3.3.1.2. Accordingly, we consider that the problem domain for this argumentation framework is described by a set of variables $V = \{v_1, ..., v_n\}$ and a set of states $S = \{s_1, ..., s_m\}$ in which the variables can be. Each variable $v_i \in V$ can be in a state $s_j \in S$ with a given probability. The set of states a variable v_i can be in is denoted by $S_{v_i} \subseteq S$, and is defined as the *variable state set*. We define two types of variables: observations, *obs*, and fault root causes, frc, which compose the set $V = obs \cup frc$. Those observations and fault root causes are modelled as variables of the agent's *Causal Model*. This allows the agent to infer the probability of a variable is in a given state. That probability represents the agent's degree of certainty about the state of a given variable, which is the key concept to handle the uncertainty of the diagnosis process. In this argumentation framework, we denote that probability as $p(i, j) = Pr(v_i, s_j) = [0, 1]$, where $s_j \in S_{v_i}$. To condense the probabilities of all states of a given variable v_i, we define

a set of probabilities on that variable, as a statement st_{v_i}. Formally,

Definition 4.3.1.1 *A **statement** st_{v_i} is a pair $\langle v_i, \mathcal{D} \rangle$ where $v_i \in \mathcal{V}$ and $\mathcal{D}$ is a set of probabilities $p(i,j)$, which represent the probability of the variable v_i is in the state s_j.*

A statement st_{v_i} on a variable v_i is coherent if and only if $\forall p(i,j) \in st_{v_i}, \sum p(i,j) = 1$. That means that a statement is coherent if represents a *probability distribution* for the possible states of the variable v_i. Formally,

Definition 4.3.1.2 *A **statement** st_{v_i} is **coherent** $\iff \forall p(i,j) \in \mathcal{D} \mid \sum \forall p(i,j) = 1$. Otherwise, st_{v_i} is **incoherent**.*

We define three different types of statements: *evidence*, *assumption* and *proposal*. On one hand, an *evidence* is based on an observation obtained from the network and represents that a variable is in a specific state. As observed directly from the network, we consider that information is certain and cannot be discussed. Formally,

Definition 4.3.1.3 *Given a coherent statement $st_{v_i} = \langle v_i, \mathcal{D} \rangle$, st_{v_i} is an **evidence** $\iff v_i \in obs \wedge \exists \langle p(i,j) \rangle \in \mathcal{D} \mid p(i,j) = 1$.*

On the other hand, an *assumption* represents an unobserved variable. That means the agent cannot gather that information for any reason, such as technical issues or privacy restrictions. An agent can infer this assumption based on the knowledge contained in its *Causal Model*. As an assumption is based on background knowledge and is not a certain information, this type of statement can be discussed among agents to clarify the state of the variable. Formally,

Definition 4.3.1.4 *Given a coherent statement $st_{v_i} = \langle v_i, \mathcal{D} \rangle$, st_{v_i} is an **assumption** $\iff v_i \in obs \wedge \nexists \langle p(i,j) \rangle \in \mathcal{D} \mid p = 1$.*

Finally, a *proposal* represents a hypothesis for the states of a specific variable, that can be a conclusion of possible fault root cause or a possible clarification for an assumption. Formally,

Definition 4.3.1.5 *Given a coherent statement $st_{v_i} = \langle v_i, \mathcal{D} \rangle$, st_{v_i} is a **proposal** $\iff v_i \in V \wedge \nexists \langle o, p \rangle \in \mathcal{D} \mid p = 1$.*

To summarise all statements about a specific diagnosis case, statements about different variables in the domain are collected into a *set of statements* to conform arguments. Based on the three types of statements, we define an argument as a triplet of sets of statements. Formally,

Definition 4.3.1.6 *An* ***argument*** *arg is a triplet* $\langle \mathcal{E}, \mathcal{A}, \mathcal{P} \rangle$*, where* $\mathcal{E}$ *is the* ***evidence set*** *of arg,* $\mathcal{A}$ *is the* ***assumption set*** *of arg, and* $\mathcal{P}$ *is the* ***proposal set*** *of arg.*

In conclusion, this argumentation framework defines three different types of statements which represent different types of knowledge. Arguments are built as a triplet of sets of statements: evidence set, assumption set and proposal set.

4.3.2 Relations between arguments

The framework defined in the previous section has been proposed to perform hypothesis discrimination tasks among sets of agents in distributed fault diagnosis processes. Thus, those agents have to generate and evaluate arguments to try to conclude the process with the most reliable diagnosis conclusion. That evaluation process is based on the relations between every pair of arguments which are exposed below.

To explain the relations between arguments, we define a pair of agents, Ag_i and Ag_j, can agree or disagree, because they have different background knowledge and different views of the global problem when they are diagnosing in federated domains. Hence, if Ag_i, generates an argument, arg_i, and Ag_j generates another as response, arg_j; there can be two main types of relation between those arguments: a **support** relation, if both agents agree, or an **attack** relation, if not. Moreover, there are different types of attacks. But, before starting with the definition of those attack types, we must define the relations of *similarity* and *preferability* between two statements, α and β, generated by two different agents about a specific variable. These concepts of *similarity* and *preferability* are explained in Sections 4.3.2.1 and 4.3.2.2 respectively. Finally, the types of attacks between arguments are exposed in Section 4.3.2.3.

4.3.2.1 Similarity

We define *Similarity* between statements as a measure of equivalence between them. If two statements are similar enough, we say they are equivalent for the fault diagnosis task. To measure the *similarity* between two statements, we process those statements as *probability distributions* that represent the possible states of the variable v_i, as defined in Section 4.3.

Thus, similarity will be used to know if two agents agree or disagree about the state of a specific variable, i.e. if their statements are *similar enough* or not. Strictly, two statements are equal if both have equal probabilities for every state of a variable $p(i, j)$. As agents have their own causal models, it is not probable that two statements from different agents have equal probabilities. For that reason, the definition of similarity between statements includes some permissibility to allow that agreement was found with more flexibility. This reduces the number of arguments needed to achieve a reliable conclusion. Moreover, for our fault diagnosis task, we do not need a strict equity between statements. Two *similar* statements are tolerable agreement between agents to continue with the argumentation process.

Therefore, to measure the *similarity* of two statements α, β about the same variable v_i, we need to apply a **distance function**, Δ, to get a numeric measure, $\Delta(\alpha, \beta) \in \mathbb{R}$, about how similar two statements are between them to know if agents agree or disagree. This similarity can be measured using different distance metrics, such as Euclidean distance, Hellinger distance, Kullback-Leiber distance, J-divergence distance or Cumulative Distribution Function (CDF) distance. For a review of distance metrics between probability distributions, please refer to the work of Koiter (2006). For our fault diagnosis field, we pick the Hellinger distance (Nikulin, 2002) which offers the following interesting features. Firstly, it can be normalised to bound the metrics in $[0, 1]$, which simplify its processing in contrast with other unbounded metrics, such as Kullback-Leiber distance or J-divergence. Secondly, it does not require any order sequence among the states of a variable in contrast with CDF distance that is targeted towards ordinal distributions. Thirdly, it is symmetric, in contrast with others, such as Kullback-Leiber distance. That is an interesting feature do not require any order between statements to measure the distance between them. Because *similarity* must be a symmetric measure. Finally, it is more sensitive near 0 and 1, in contrast with Euclidean distance. That is a desirable feature because probabilities near to those values in a statement represents that an agent is almost sure that a variable is $(p(i, j) \approx 1)$ or is not $(p(i, j) \approx 0)$ in a given state. This is interesting because a statement that is *more sure* about the state of a variable should be less similar than other less certain.

Then, with the normalised Hellinger distance (Nikulin, 2002), shown in Definition 4.3.2.1, chosen to measure the similarity between two statements, $\Delta(\alpha, \beta) \in \mathbb{R}$, we define a **threshold** $th = [0, 1]$ to establish the bound distance between two statements to be classified as *similar enough*. Therefore, two statements α, β about the same variable v_i are *similar enough*, if the distance between them is below the **threshold**, $\Delta(\alpha, \beta) < th$.

Definition 4.3.2.1 *Given two discrete probability distributions* $P = (p_1, ..., p_k)$ *and* $Q = (q_1, ..., q_k)$*, their normalised Hellinger distance is defined as:*

$$H(P,Q) = \frac{1}{\sqrt{2}}\sqrt{\sum_{i=1}^{k}(\sqrt{p_i} - \sqrt{q_i})^2}$$

Based on this definition, a threshold value near to 0 would imply a strict behaviour, as agents will only agree when the distance between them is really narrow. This would increase the number of arguments to achieve a conclusion. In contrast, a threshold value near to 1 would entail a permissive behaviour, as agents would almost always agree, which would reduce the duration of the argumentation. But, any convergence would not be achieved. Accordingly, a threshold value between the two bounds should be adjusted depending on the preferences between these two behaviours. Anyway, a value above 0.5 would have no sense to get agreement at the end of the argumentation, because it would diverge the beliefs of the agents instead to converge to a common conclusion. Thus, the threshold value should be between 0 and 0.5 to foster agreements, $0 < th < 0.5$.

Finally, we formally define *similarity* as follows:

Definition 4.3.2.2 *Given two coherent statements α, β about the same variable v_i, a distance function Δ and a threshold th, α is **similar** to β, and viceversa $\iff \Delta(\alpha, \beta) < th$. Otherwise, they are not similar.*

4.3.2.2 Preferability

We define *Preferability* between two statements as an order of preference between them. As the goal of the system is to diagnose fault root causes, a statement that contains more reliable information about a variable is preferred against other. As mentioned above, when two statements are not similar enough, i.e. they are not equivalent, we can define an order of preference between them. In other words, we can define how to pick a statement among a set.

As mentioned previously, a statement is composed by a set of probabilities $p(i, j)$ that a variable is a given state. Those probabilities represent the agent's degree of certainty about the state of a given variable. In other words, a probability $p(i, j) \simeq 1$ represents the agent is almost sure the variable v_i is in the state s_j. Contrarily, a probability $p(i, j) \simeq 0$ means the agents is almost sure the variable is not in that state. For the fault diagnosis task, that certainty is more valuable than other less certain probabilities, such as $p(i, j) \simeq 0.5$. Thus, we would prefer the statement that represent the highest level of certainty to get more certain conclusions. Notice that this preference is valid because we are considering all agents have a common goal of diagnosing faults and they cooperate to achieve it. In competitive

environments, every agent could have different preferences and this decision could be made based on different criteria. We formally define *preferability* as follows:

Definition 4.3.2.3 *Given two coherent statement α, β about the same variable v_i with their respective sets of probabilities $\mathcal{D}_\alpha, \mathcal{D}_\beta$, α is **preferred** to $\beta \iff \exists p(i,j) \in \mathcal{D}_\alpha \mid \forall p(i,k) \in \mathcal{D}_\beta \wedge p(i,j) > p(i,k)$.*

Finally, the orderability of statements provided by this preferability property is an interesting feature to solve conflicts and to choose the most preferable statement of a set. This property is used in the conflict resolution strategies exposed in Section 4.4.3.

4.3.2.3 Types of Attacks

This section defines the different type of attack relations that can exist between two arguments, arg_j and arg_i, based on the concepts of *similarity* and *preferability*. We define three different **attack** types if agents disagree on a specific type of statement: *discovery*, *clarification* and *contrariness*.

Definition 4.3.2.4 *Given two arguments arg_i and arg_j, generated by agents Ag_i and Ag_j, respectively. If arg_j contains any new evidence, $st_{v_i}|v_i \in obs$, about the diagnosis in progress that is not in arg_i, we define that arg_j is a **discovery** for arg_i.*

If arg_j is a *discovery* for arg_i, arg_i is discarded and agent Ag_i should generate a new argument including the new evidences. The *discovery* is the most basic attack type because it modifies the ground of the reasoning process. Hence, if the ground (the evidences) changes, the output of the inference process (the conclusions) could change too.

Definition 4.3.2.5 *Given two arguments arg_i and arg_j, generated by agents Ag_i and Ag_j, respectively. When arg_j contains a proposal about the variable, $st_{v_i}|v_i \in obs$, and arg_j contains an assumption of that variable, if both statements are not similar and the proposal is preferred to the assumption, we define that arg_j is a **clarification** for arg_i.*

If arg_j is a *clarification* for arg_i, arg_i is discarded and agent Ag_i should accept the proposal and generate a new argument including it. A *clarification* attack tries to offer a more certain information about an unknown variable represented in an assumption.

Definition 4.3.2.6 *Given two arguments arg_i and arg_j, generated by agents Ag_i and Ag_j, respectively. When arg_j contains a proposal that contains a possible conclusion of the diagnosis process, $st_{v_i}|v_i \in frc$, and arg_j contains other possible conclusions, $st_{v_j}|v_j \in frc$; if both conclusions are not similar enough, we define that arg_j is a* ***contrariness*** *for arg_i, and vice versa.*

As no statement is discarded in this attack type, both agents stop of arguing until new *discoveries* or *clarifications* appear during the argumentation process. This type of attack is resolved globally at the end of the argumentation process of the coordination protocol as shown in Section 4.4.

Finally, we define that an argument **supports** another one if no attack relation exists between them. If a support relation exists among all pairs of non-discarded arguments, a global *agreement* has been achieved. But, as every agent has its own domain knowledge synthesised in the *Causal Model*, we cannot ensure a global agreement for any argumentation. Thus, a conflict resolution mechanism must should be applied if required, as proposed in the protocol shown in Section 4.4.3.

4.4 B2D2 Coordination Protocol

This section proposes a coordination protocol for distributed autonomic fault diagnosis in federated domains, named B2D2 Coordination Protocol. In this protocol, there are two different agent roles, namely the ***Argumentative*** role and the ***Manager*** role. An *argumentative* agent is responsible to generate and process arguments. A *manager* agent is responsible (i) to establish a coalition of *argumentative* agents to argue, (ii) to decide when an argumentation process has finished and (iii) to figure out the conclusion of the argumentation process. In our case, a B2D2 Agent can play these roles following the argumentation framework proposed in Section 4.3. A simplified overview of an *Argumentative B2D2* Agent is shown Figure 4.1. The figure includes the argumentative model and the argumentative capability of the extended B2D2 Agent Architecture, which are exposed in Section 4.5.

The proposed protocol has three different phases[1], as shown in Figure 4.2. The initial phase for the formation of a group of agents capable to reach a reliable conclusion for a specific diagnosis case is called *Coalition Formation Phase*, as explained in Section 4.4.1. After the argumentation coalition is established and every agent knows the rest of the

[1]For the sake of clarity, the diagrams included in the following subsections follows the Business Process Model and Notation (BPMN) 2.0 standard specification.

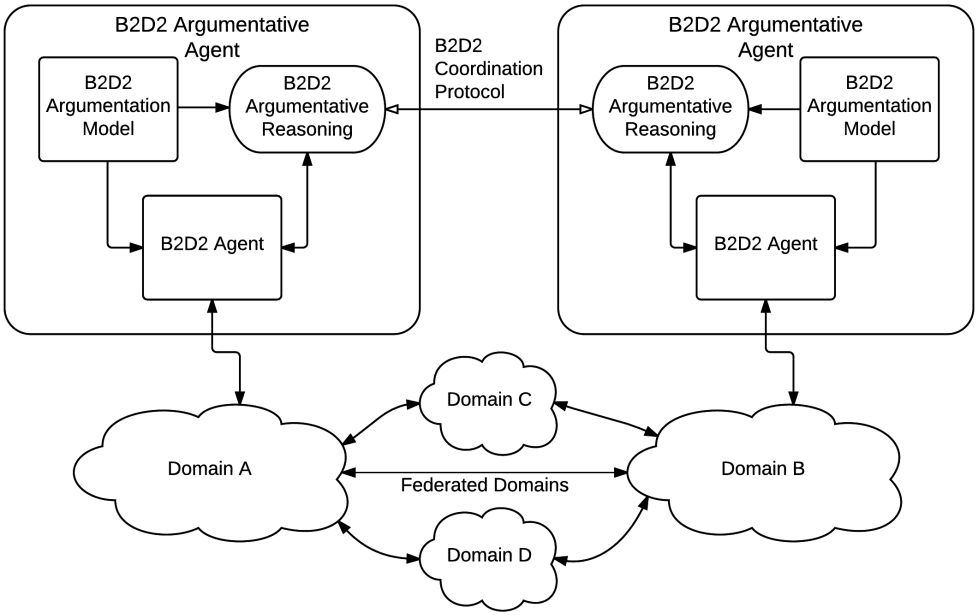

Figure 4.1: Overview of B2D2 Argumentative Agents in Federated Domains.

constituents, the *Argumentation Phase* starts, as exposed in Section 4.4.2. Finally, when a *manager* agent decides that the argumentation is finished, the arguments are analysed to extract a final conclusion during the *Conclusion Phase*, as shown in Section 4.4.3.

Figure 4.2: Phases of the B2D2 Coordination Protocol.

4.4.1 Coalition Formation Phase

This phase starts when an *Argumentative* agent (*Initiator agent*) initiates a new process to diagnose an anomaly or a symptom detected in the supervised network elements. This *Argumentative* agent sends a *Coalition Formation Request* message to the *Manager* agent. This message includes data to identify the problem domain. Then, the *Manager* agent broadcasts the message to the rest of the *Argumentative* agents as a *Coalition Invitation* message. When an agent receives that invitation, it decides whether to join the coalition or not. If it decides to join, it must respond to the invitation. Otherwise, it ignores it. This

decision is based on the local private knowledge of the agent. In other words, if the agent can offer any relevant information about the problem domain, it would accept the invitation. Otherwise, it would not.

A period of time is specified as deadline to response the invitation to avoid deadlocks while the *Manager* agent is waiting responses from all *Argumentative* agents. At this way, the *Coalition Invitation* message can be broadcasted. After the deadline, the *Manager* agent establishes the coalition with all agents that accepted the invitation and the *Initiator agent*. Finally, it broadcasts a *Coalition Established* message with the complete list of agents that joined the coalition. With this last message, the *Coalition Formation Phase* is finished. Figure 4.3 shows a diagram of this phase.

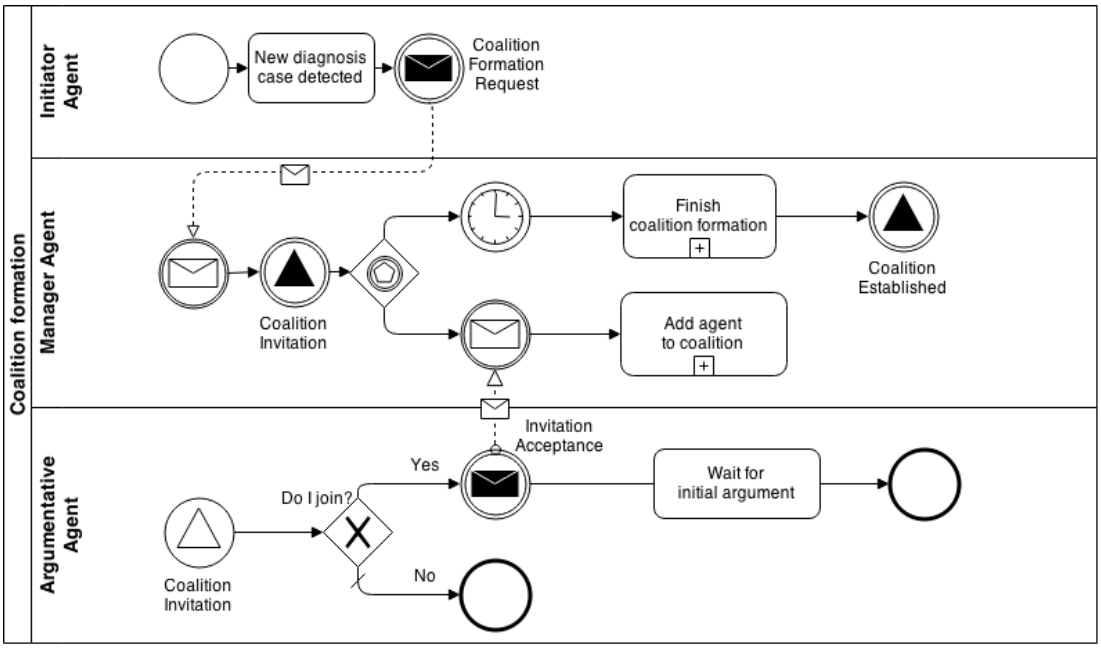

Figure 4.3: Coalition Formation Phase.

4.4.2 Argumentation Phase

This phase starts when the *Coalition Established* message, sent in the previous phase, is received by the *Initiator Agent* who broadcasts the **initial argument** to the coalition. After this step, this agent acts as any other *Argumentative* agent. Later, every agent in the coalition receives that *initial argument* and analyses it to find any attack relation following the reasoning process exposed in Section 4.5.2.3. After an argument is processed, the options of an *argumentative* agent are the followings: (i) If a *discovery* or a *clarification* relation is found, the agent generates a new updated argument. (ii) Alternatively, if a *contrariness*

relation is found, it looks for new information from the environment to add it to the argumentation process. (iii) Finally, if the received argument *support* the agent beliefs, it can **wait** until other argument is received.

During this Argumentation Phase, an agent can receive a message that contains an argument while it is processing or generating other one received previously. In that case, the agent should analyse the arguments in reception order. Afterwards, it would generate only one argument as response when all incoming arguments have been processed. This strategy reduces the number of arguments generated during the argumentation dialogue. This makes the argumentation process requires less messaging and, consequently, less computational resources.

As shown in Figure 4.4, agents broadcasts any generated argument to all coalition members, including the *Manager* agent. At this way, we ensure that any agent receive all arguments to attack or support any of them during the argumentation dialogue. Every time the *Manager* agent receives an argument, it restart a timer that will be used to know when the argumentation is finished. When all agents keep in silence for a time longer than the **silence timeout**, the *Manager* decides this phase is done and starts the *Conclusion Phase*. In other words, the Argumentation Phase continues until every agent has proposed its conclusions to the diagnosis case under consideration and it does not receive any new argument that makes an agent to change those conclusions. It is important to remark that *Argumentative* agents must be able to process and generate arguments in a time lower than the **silence timeout**, to ensure the *Manager* agent does not finish this phase prematurely. Notice that the end of the Argumentation Phase can be delayed in time depending on the number of agents and their configuration (see the notion of similarity threshold and its impact in the permissibility of the agents in Section 4.3.2). To avoid this phase would be too time consuming, the *Manager* agent can be configured to allow the coalition member agents to argue during a time, and then the *Conclusion Phase* starts in any case, even if the Argumentation Phase is not finished. Anyway, the *Manager* agent sends a message to the coalition members to notify that the argumentation process has finished including the conclusions of the argumentation process.

4.4.3 Conclusion Phase

After the Argumentation Phase is done, the *Manager* agent must process arguments which would have not been discarded due to attacks and extract a final conclusion, as shown in Figure 4.5. We name that set of arguments as **candidate arguments set**. Based on the attack types of the argumentation framework proposed in Section 4.3, only one type

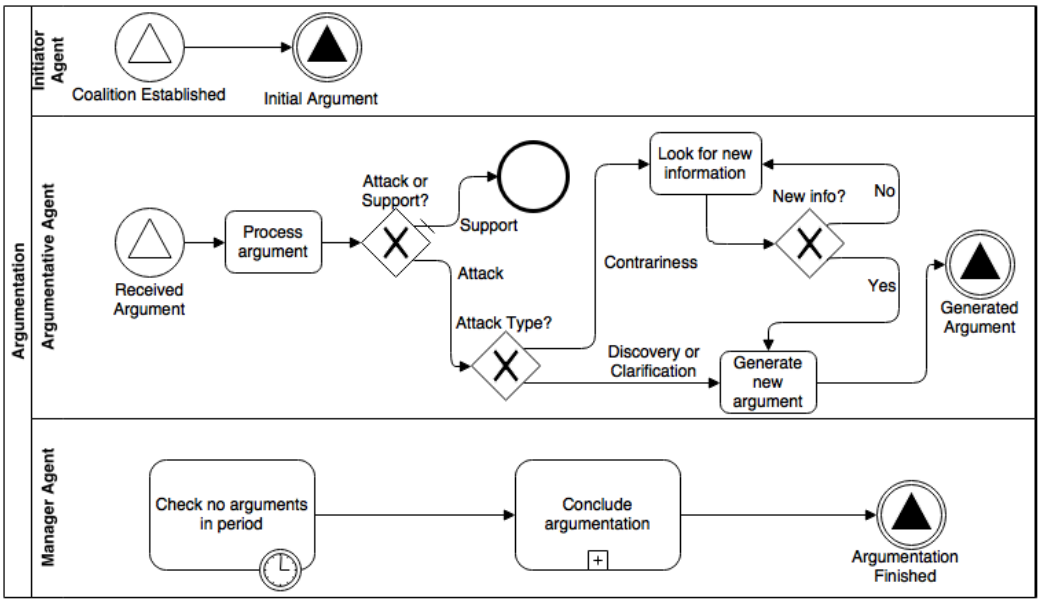

Figure 4.4: Argumentation Phase.

of attack relation can be among arguments of this set: the *contrariness* relation. In other words, every argument of the set contains a proposal about a possible fault root cause. If some *contrariness* is found between two arguments, we say that a **conflict** is found. If some *conflict* is found while analysing the *candidate arguments set*, different criteria can be applied to resolve those conflicts and select a final conclusion. Contrarily, if no conflict is found, the conclusion all agents agree on is selected as the **final conclusion**. After conflicts have been resolved, a final conclusion is obtained as the result of the argumentation dialogue. We propose two different conflict resolution strategies for this final phase.

- *Most Popular Conclusion*: This strategy picks as final conclusion the most popular conclusions in the *candidate arguments set*.
- *Most Confident Conclusion*: This strategy picks as final conclusion the conclusions with highest confidence in the *candidate arguments set*.
- *Weighted Conclusion*: This strategy calculate the average confidence value among all arguments with the same conclusion and picks as final conclusion, the one with highest average confidence value.

Finally, the *Manager* agent sends the conclusions of the distributed diagnosis to all *Argumentative Agents* to notify them end that the argumentation is finished.

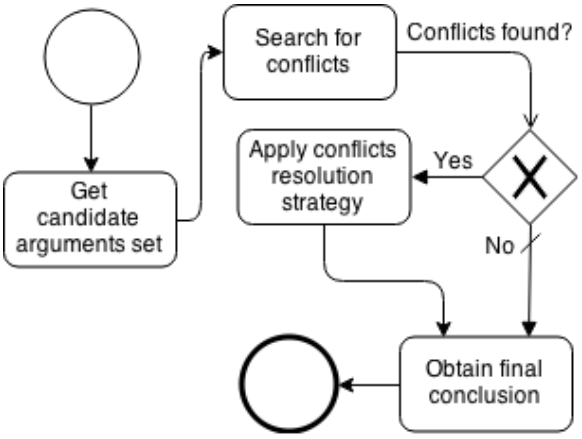

Figure 4.5: Conclusion Phase.

4.5 B2D2 Argumentative Agent Architecture

This section proposes an extension of the B2D2 Agent Architecture presented in the previous chapter. The aim of this extension is to include the argumentative capability in the agent architecture. That capability is used to perform a distributed hypothesis discrimination process cooperatively among agents, as exposed in the coordination protocol presented in Section 4.4. To include the proposed coordination mechanism in the knowledge models of the agent, an argumentation model is presented in Section 4.5.1. Later, the formalisation of the argumentative capability for a B2D2 Argumentative Agent is presented in Section 4.5.2.

4.5.1 Argumentation Model

The model proposed in this section covers the concepts required to enable the execution of a distributed fault diagnosis process using the proposed argumentation framework. This Argumentation Model has been designed to enable the proposed B2D2 Agent Architecture to carry out a distributed diagnosis following the steps defined in the proposed coordination protocol. The developed model is formalised as an ontology which extends the proposed B2D2 Diagnosis Ontology presented in the previous chapter. The main classes of the ontology are shown in Figure 4.6. The most important concept of the model[2] is *argument*, which is composed by three different types of *statements*: *evidences*, *assumptions* and *proposals*. While, *evidences* are associated with *observations* gathered from *network objects*, both *assumptions* and *proposals* are inferred using the *Causal Model* of B2D2 *Agents*. Those *agents*

[2]Notice that all the *italic* concepts refer to classes of the proposed Argumentation Model to simplify the explanations.

conform *coalitions* to discuss trough an *argumentation* process to perform a *distributed diagnosis*. Notice that some of these concepts are equivalent to some concepts in the B2D2 Diagnosis Ontology, such as *network object*, but other ones extend or are related with others, such as *distributed diagnosis* which is a tpye of *diagnosis* or *evidence* which is a type of statement directly related with the concept of *observation*. For further details, please refer to the complete argumentation model[3].

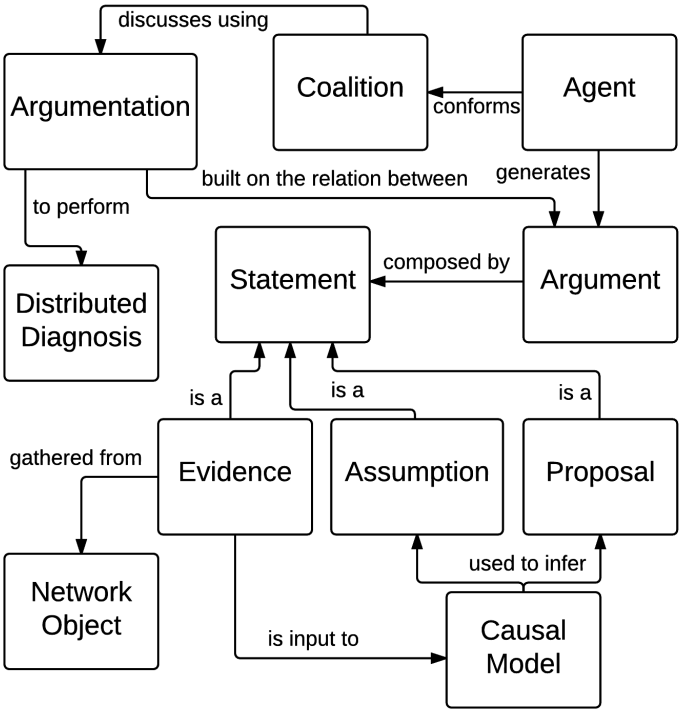

Figure 4.6: Main classes of the Argumentation Model.

To illustrate the use of the proposed argumentation model, Table 4.5 shows a simplified example which includes a set of individuals of the ontology[4] as explained below. In the example, we can find an *argument* (lines 10-18) which is composed by three sets of statements: three *evidences* (lines 12-14), three *assumptions* (lines 15-17) and one *proposal* (line 18). To perform the argumentation process, the *argumentative agent* which generated that *argument* (line 11) is included in a *coalition* (lines 20-23) conformed by three *argumentative agents*. That *coalition* is associated with a specific *argumentation* (lines 25-27) process which is managed by a *manager agent* (line 26). Moreover, one of the *proposals* which compose

[3]B2D2 Argumentation Ontology Specification: `http://www.gsi.dit.upm.es/ontologies/b2d2/argumentation`

[4]Notice that all the italic concepts refer to classes of the proposed Argumentation Model to simplify the explanations.

the *argument* is shown in the example too (lines 29-31). That *proposal* has an associated *probability distribution* (lines 33-35) for a specific *variable* of the *Causal Model* (line 34) which represents a possible fault root cause, as exposed in Section 4.3.

```
1  @prefix pr-owl:
2          <http://www.pr-owl.org/pr-owl.owl#>
3  @prefix b2d2-diag:
4          <http://www.gsi.dit.upm.es/ontologies/b2d2/diagnosis/ns#> .
5  @prefix b2d2-arg:
6          <http://www.gsi.dit.upm.es/ontologies/b2d2/argumentation/ns#> .
7  @prefix b2d2-argex:
8          <http://www.gsi.dit.upm.es/ontologies/b2d2/arg-example/ns#> .
9
10 b2d2-argex:argument-003 a b2d2-arg:Argument ;
11                 b2d2-arg:isGeneratedBy b2d2-argex:agent-001 ;
12                 b2d2-arg:hasEvidenceStatement b2d2-argex:evidence-005 ,
13                     b2d2-argex:evidence-006 ,
14                     b2d2-argex:evidence-007 ;
15                 b2d2-arg:hasAssumptionStatement b2d2-argex:assumption-001 ,
16                     b2d2-argex:assumption-002 ,
17                     b2d2-argex:assumption-003 ;
18                 b2d2-arg:hasProposalStatement b2d2-argex:proposal-003 .
19
20 b2d2-argex:coalition-001 a b2d2-arg:Coalition ;
21                 b2d2-arg:hasArgumentativeAgent b2d2-argex:agent-001 ,
22                     b2d2-argex:agent-002 ,
23                     b2d2-argex:agent-003 .
24
25 b2d2-argex:argumentation-001 a b2d2-arg:Argumentation ;
26                 b2d2-arg:managedBy b2d2-argex:agent-002 ;
27                 b2d2-arg:hasCoalition b2d2-argex:coalition-001 .
28
29 b2d2-argex:proposal-003 a b2d2-arg:Proposal,
30                                 b2d2-arg:Statement ;
31                 b2d2-arg:hasProbDist b2d2-argex:probdist-proposal-003 .
32
33 b2d2-argex:probdist-proposal-003 a pr-owl:PR-OWLTable ;
34                 pr-owl:isProbDistOf b2d2-argex:fault-root-cause-variable-002 ;
35                 pr-owl:hasProbAssign b2d2-argex:probassign-005 .
```

Table 4.5: Application example for the Argumentation Model.

In conclusion, this section proposes an *Argumentation Model* which covers the concepts required to perform a distributed autonomic fault diagnosis using the argumentation framework proposed in Section 4.3. It is formalised as an OWL ontology which extends the *Diagnosis Model* presented in the previous chapter to enable the argumentative capability in a B2D2 Agent.

4.5.2 Argumentative Capability

The application of the proposed coordination mechanisms by a B2D2 Agent requires an extension of the agent architecture presented in Chapter 3. That extension to enable the argumentation capability is presented in this section to formalise the B2D2 Argumentative Agent Architecture. This extension covers all tasks mentioned in the coordination protocol exposed in Section 4.4, which makes a B2D2 Argumentative Agent fully compatible with that protocol. Those tasks are formalised in the following subsections using the AgentSpeak language (Rao, 1996) as follows. Firstly, Section 4.5.2.1 exposes how an argumentative agent can start a distributed diagnosis process extending the plans of the B2D2 Agent exposed in the previous chapter. Section 4.5.2.2 explains how an agent performs the coalition formation task. Section 4.5.2.3 presents the evaluation of arguments generated during the argumentation phase to perform the distributed hypothesis discrimination task. Finally, Section 4.5.2.4 shows the task to get the final conclusion of the diagnosis.

4.5.2.1 Initiating argumentation processes

The detection of a symptom triggers the diagnosis process exposed in the previous chapter for a non-argumentative B2D2 Agent. After the generation of a hypothesis set, the B2D2 Agent actives a goal to look for new evidences to discriminate the most probable cause of fault among its hypothesis. At this point, the argumentative capability can be included in the reasoning cycle of the agent. As shown in Table 4.6, if an agent detects it is not able to collect some evidences due to technical restrictions (line 4), such as they are allocated in other domains, it starts a distributed diagnosis process as *Initiator* agent, as exposed in the coordination protocol explained in Section 4.4. Then, the *Initiator* agent looks for a *Manager* agent (line 6) and later, sends a *Coalition Formation Request* message (line 7) and waits until the coalition is established (line 8).

```
1 /* Initiating a Distributed Hypothesis Discrimination Task - Plans */
2 // Start argumentation if required
3 +!lookForEvidences(actions,networkElements) :
4         restrictedAccessToDomain(domain,networkElements,evidences)
5         <-
6         findArgumentationManager(actors);
7         send(manager,coalitionRequest);
8         !wait(coalitionEstablished).
```

Table 4.6: Initiating an Argumentation for Hypothesis Discrimination Task.

4.5.2.2 Conforming coalitions

To enable the capabilities required to play the *Manager* role, an additional initial goal must be added to the goal base of the agent, as shown in Table 4.7 in line 2. That makes when the *Manager* agent receives a *Coalition Formation Request* (line 8), it checks available actors (line 10) and broadcast a *Coalition Invitation* message (line 11). Then, the *Manager* agent establishes the deadline to this coalition formation phase (line 12) and waits responses (line 13). It starts receiving *Invitation Acceptance* messages (line 16) and add those argumentative agents to the coalition (line 18) until the deadline is over (line 20). Then, the coalition is conformed (line 22) and the *Coalition Established* message is broadcasted (23). At this point, the *Manager* agent starts to check if the argumentation process is finished (line 24), as exposed in Section 4.4.2.

Similarly, a new initial goal for the *Argumentative* agent must be included in the goal base to join coalitions (line 27), as shown in Table 4.7. This goal represents the desire to join coalitions to diagnose faults in a cross-domain scenarios. The agent listen incoming *Coalition Invitation* messages (line 28) and, based on its domain knowledge, it decides to join the coalition (lines 32-36) or to ignore the invitation (lines 37-40).

4.5.2.3 Processing arguments

After the coalition is established and all its constituents have been properly notified, the *Initiator* agent sends the initial argument and the reminder *Argumentative* agents waits until they receive that argument. For explanation purpose, we are going to label the initial argument generated by the *Initiator* agent as arg_0. To build that argument, the agent feeds its Causal model with a set of evidences, i.e. variables the agent does not know certainly

```
1  /* Initial goal */
2  !listenRequests(actors,domains).
3  !listenInvitations(coalition,domains).
4
5  /* Conforming coalitions - Plans */
6  // Manager listens for coalition formation requests
7  +!listenRequests(actors,domains) :
8          incoming(coalitionRequest)
9          <-
10         check(actors,domains);
11         broadcast(coalitionInvitation);
12         set(deadline);
13         !waitResponses(coalitionInvitation)
14 // Wait until deadline is reached
15 +!waitResponses(coalitionInvitation) :
16         incoming(inviationAcceptance) & not expired(deadline)
17         <-
18         addToCoalition(agent).
19 +!waitResponses(coalitionInvitation) :
20         expired(deadline)
21         <-
22         conform(coalition);
23         broadcast(coalitionEstablished);
24         !checkArgumentationIsFinished(coalition).
25
26 // Argumentative agent listens for coalition invitations
27 +!listenInvitations(coalition,domains) :
28         incoming(coalitionInvitation)
29         <-
30         !decideIfJoin(coalition,domains).
31 // Decice if join the coalition or not
32 +!decideIfJoin(coalition,domains) :
33         includesmyDomain(domains);
34         <-
35         join(coalition);
36         !wait(coalitionEstablished).
37 +!decideIfJoin(coalition,domains) :
38         not includesmyDomain(domains);
39         <-
40         ignore(coalitionRequest).
```

Table 4.7: Conforming coalitions for an Argumentation process.

while observes the network. After, it infers a set of assumptions, i.e. variables the agent unknown, and a set of possible conclusions, i.e. proposals of possible fault root causes. Inferring these data with the Bayesian inference process, the agent builds the argument generating statements for the variables V of the problem domain, as exposed in Section 4.3. Then, the agent broadcasts the argument to the coalition and waits any response. After this point, the *Initiator* agents acts as any other *Argumentative* agent in the coalition. This process is formalised in Table 4.8 for *Initiator* agent (lines 3-8) and the reminder *Argumentative* agents (lines 9-12).

```
1 /* Initiating the Argumentation Phase - Plans */
2 // Sends the initial argument
3 +!wait(coalitionEstablished) :
4         incoming(coalitionEstablishedNotification) & myself(initiator)
5         <-
6         generate(argument);
7         broadcast(argument,coalition);
8         !wait(incomingArguments).
9 +!wait(coalitionEstablished) :
10        incoming(coalitionEstablishedNotification) & not myself(initiator)
11        <-
12        !wait(incomingArguments).
```

Table 4.8: Initiating the argumentation phase.

The reasoning process to evaluate arguments is formalised in Table 4.9. When an incoming argument, arg_1, is received by an *Argumentative* agent (line 4), it is compared with the argument that the agent would generate at that moment, arg' (line 6). This is because the last argument generated by the agent, arg_0, could be outdated as new evidences could be detected or agent's beliefs could have changed while it was waiting. Thus, the agent builds arg' and look for any relation between arg_1 and arg' as explained below.

- If a *discovery* relation is detected, the *Causal Model* of the agent is fed again with the updated evidence set including the new evidences found in the incoming argument. Then, a new argument which condenses the current agent's beliefs about the diagnosis case is generated and sent to continue the argumentation (line 14-18).
- If a *clarification* relation is detected, the *Causal Model* is fed again with the updated evidence set including the clarifications as soft-evidences (Pan et al., 2006). Then, a new argument is generated and sent with the updated assumption (line 14-18).

- If a *contrariness* relation is found, the agent has to look for new information to generate the next argument(line 20-25). If no information is found, no argument is generated.

- If none attack relation is found, the received argument arg_1 *supports* the agent's argument arg'. Then, no new argument is generated because both agents agree (lines 10-12).

```
1  /* Reasoning during the Argumentation Phase - Plans */
2  // Sends the initial argument
3  +!wait(incomingArguments) :
4          incoming(argument)
5          <-
6          analyse(argument);
7          !decideNextAction(argumentRelations).
8  // Decide the next action of the agent
9  +!decideNextAction(argumentRelations) :
10         support(argument)
11         <-
12         !wait(incomingArguments).
13 +!decideNextAction(argumentRelations) :
14         discovery(argument) | clarification(argument)
15         <-
16         generate(argument);
17         broadcast(updatedArgument,coalition);
18         !wait(incomingArguments).
19 +!decideNextAction(argumentRelations) :
20         contrariness(argument)
21         <-
22         !lookFor(newInfo);
23         generate(argument);
24         broadcast(updatedArgument,coalition);
25         !wait(incomingArguments).
```

Table 4.9: Processing arguments.

The information gathering process when a *contrariness* (line 16) is found is explained below. The agent starts looking for new evidences, i.e. certain knowledge observed from the network. If new evidences are found, the *Causal Model* is fed again with the updated evidence set and a new argument is generated. If not; the agent checks if any of the assumption contained in arg' is *preferred* to the assumption about the same variable in arg_1. If it is,

the agent generates a new argument with the *preferred* assumption as a proposal. Thus, when the other agent receives the generated argument, it will find a *clarification* relation and, at that point both agents will agree about the status of that variable. Otherwise, if no new evidence is found and no assumption can be clarified, the argumentative agent which is processing those arguments does not respond with any argument because both agents agree about the evidences and the assumptions. In other words, they agree in the ground of the reasoning process, but they disagree about the conclusion. Thus, the argumentative agent continues looking for new observations that could change the conflictive state of the argumentation until the argumentation finishes.

Notice that an agent must look for attack relations in the exposed order: *discovery*, *clarification* and *contrariness*. This is because a *discovery* found between two arguments. That gives rise to a new argument and any *clarification* or *contrariness* relation can be disconsidered because the previous argument is outdated and replaced by the new one. The same situation happens with the *clarification* relation. If a *clarification* is found, any *contrariness* relation is no here longer valuable because the argument is discarded and replaced by a new one.

4.5.2.4 Concluding argumentation processes

This control task of the *Manager* agent is formalised in Table 4.10. Since the initial argument is sent by the *Initiator* agent, the *Manager* agent is supervising the argumentation process (lines 3-7) to decide to finish it if the silence timeout is exceeded, as explained in Section 4.4. Finally, when it decides the argumentation phase finishes (line 8-9), diagnosis conclusions are extracted (line 11) solving any possible conflict among arguments (line 12). Then, the coalition constituents are notified (line 13) and the *Manager* agent returns to listen for coalition formation request (line 14). After the *Argumentation Finished* notification is received, argumentative agents finish the argumentation phase and continue monitoring their network domains to detect any future fault (lines 17-22).

```
1  /* Concluding the Argumentation process - Plans */
2  // Manager agents supervise the argumentation process.
3  +!checkArgumentationIsFinished(coalition) :
4          incoming(argument) & not expired(silenceTimeout)
5          <-
6          reset(silenceTimeout);
7          !checkArgumentationIsFinished(coalition).
8  +!checkArgumentationIsFinished(coalition) :
9          expired(silenceTimeout)
10         <-
11         getCondidateSet(arguments);
12         resolveConflicts(candidates);
13         broadcast(conclusion,coalition);
14         !listenRequests(actors,domains).
15
16 // Return to monitor the network
17 +!wait(incomingArguments) :
18         incoming(conclusion)
19         <-
20         finish(diagnosis);
21         !listenInvitations(coalition,domains).
22         !monitor(networkElements).
```

Table 4.10: Concluding the distributed Hypothesis Discrimination Task.

4.6 Case Study

This section presents a case study to illustrate the operation of a B2D2 ARgumentative Multi-Agent System (BARMAS) in a federated scenario. In this case study, the scenario is composed of several federated networks that offer an enterprise service similar to the one offered by Telefónica O2 Czech Republic, shown in the evaluation of the non-argumentative B2D2 Agent in the previous chapter. An example of a distributed fault diagnosis process is used to illustrate the application of a BARMAS using the B2D2 Coordination Protocol. Finally, the proposed argumentation framework has been evaluated measuring the correctness of the conclusion obtained after the argumentation process. This evaluation process has been performed in a simulation environment in which the conditions of federated domains were reproduced to assess the validity of the proposed argumentation framework as coordination mechanism.

The rest of this section is structured as follows. Firstly, Section 4.6.1 exposes the federated network scenario of this case study where an cross-domain enterprise service is running. Section 4.6.2 presents the argumentative agents involved in the case study. Section 4.6.3 exposes a simplified distributed fault diagnosis example using the proposed coordination protocol. Finally, Section 4.6.4 shows the results of the evaluation process.

4.6.1 Federated Network Scenario

This section exposes a scenario where several telecommunication operators are offering a cross-domain service for international companies. The service allows geographically distributed entities of a company to be connected as if they were physically in the same network (i.e. a VPN service). In this federated scenario, every operator company manages its own network. Under a non-autonomic approach, human operators of every company involved in this cross-domain service should cooperate to handle any possible fault which would happen in the mentioned service. Even though we are considering an autonomic approach, we find the same situation: several agents have to cooperate to carry out fault diagnosis tasks. Initially, we could consider that this multi-agent approach is not required, that a single fault management system could perform a diagnosis process following the B2D2 Agent Architecture. But, considering the complexity of the Future Internet and other non-technical constrains, such as data privacy or business interests, that is impracticable. Therefore, we consider a federated scenario where B2D2 Argumentative Agents are responsible of specific domains and cooperate among them to perform cross-domain diagnoses. Figure 4.7 shows an exemplary agent system deployment in different regions of several European countries.

Every agent (presented as blue dots in the figure) is responsible to diagnose potential faults in its own network domain (i.e. in its geographical region).

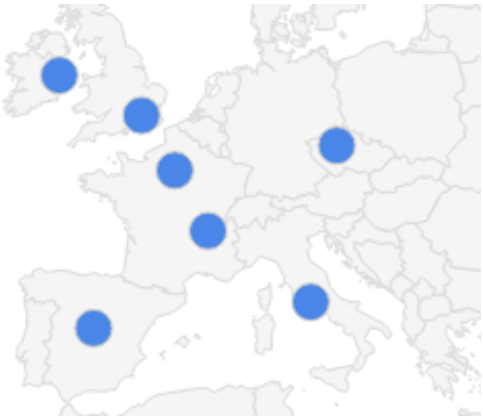

Figure 4.7: Agent deployment in motivational scenario.

For exemplification purpose, we are going to consider a simplified version of this enterprise service. Basically, the service under consideration allows geographically distributed entities of a company to be connected as if they were physically in the same network. To get this feature, a set of complex management tasks must be performed. But we are going to consider a simplified service assuming that only a set of dynamic translations of Internet Protocol (IP) addresses must be done and some registers must be updated with the proper information. Then, we will omit the required low-level configuration tasks. In this simplified scenario, we consider only two offices of a company connected by the described service, one of them in Prague, Czech Republic, and the other one in Madrid, Spain, and that connection is routed through Lyon, France.

4.6.2 Deployment of Argumentative Agents

Following the protocol proposed in Section 4.4, three different *Argumentative Agents* are deployed in the OSS of their respective cities and any of them can adopt the *Manager* role if it is required. As each agent can interact with other agents in other diagnosis processes of that service, such as Rome-Paris, Madrid-Berlin, etc., every agent has its own private background knowledge based on their own previous experience. In other words, every agent has its own *Causal Model* to reason under uncertainty based on their experience of past diagnosis cases. We can name those agents as: *Agent M* (in Madrid), *Agent P* (in Prague) and *Agent L* (in Lyon), as depicted in Figure 4.8. These agents are monitoring their networks and the interactions among them when the VPN service is running. In this simplified scenario, we consider a translation service running in a server in Lyon (*Agent L* domain)

and two registration services running in Prague (*Agent P* domain) and in Madrid (*Agent M* domain). The translation service is the core of this scenario. It is a global IP translation service for many connections of different entities. In contrast, the registration services are two local lists (for Prague and Madrid, respectively) that contain all IP addresses allowed to use the VPN service.

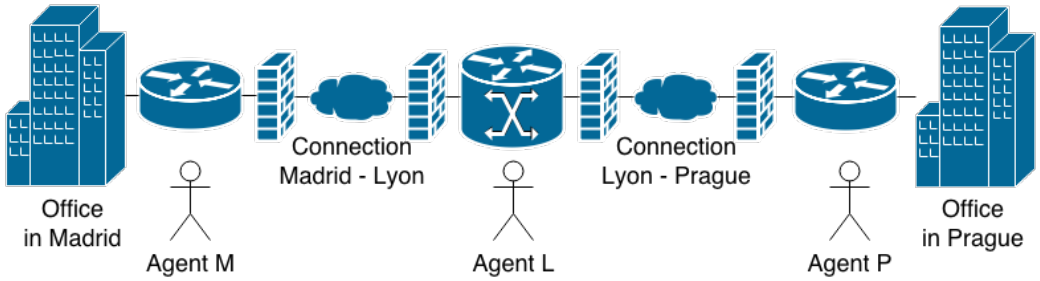

Figure 4.8: Simplified overview of the example of the federated network scenario.

4.6.3 Distributed Diagnosis Example

This section presents a worked example of how a set of three B2D2 Argumentative Agents performs a distributed fault diagnosis process. For this example, we consider the set of variables $\mathcal{V}$ which defines the problem domain and their respective possible states are the ones shown in Table 4.11[5]. We consider that all agents have a *similarity threshold* value equals to 0.2 and they calculate it using the Hellinger distance (see Section 4.3.2). For further explanations of this concepts, please see Section 4.3.

The distributed diagnosis process starts when an anomaly is detected by *Agent P* in the connection between those offices (Prague-Madrid). That anomaly is an unknown source IP address attempting to connect with a server in Prague. Then, the **Coalition Formation Phase** starts. It generates the *Coalition Formation Request* message, but no *Manager* agent is known. So, *Agent P* adopts the role of *Manager* agent and sends a *Coalition Invitation* message. After the coalition formation period, two agents (*Agent M* and *Agent L*) have accepted the invitation. Then, *Agent P* sends the *Coalition Established* message to *Agent L* and *Agent M*. Finally, the *argumentation coalition* is established with three constituents and the protocol continues to the next phase.

At the begging of the **Argumentation Phase**, *Agent P* generates and broadcasts the initial argument. That initial argument contains the information shown in Argument 1. It has three *evidences* that represent: the source IP address is unknown ({ SA:U }), the

[5]Notice that all arguments exposed below for this worked example use the contracted nomenclature exposed in Table 4.11 to facilitate the reading.

Variable	**States**
SourceMachineUp (SU)	True (T) False (F)
DestinationMachineUp (DU)	True (T) False (F)
SourceIPAddress (SA)	Known (K) Unknown (U)
DestinationIPAddress (DA)	Known (K) Unknown (U)
AllowedIPListsRecentlyUpdated (AR)	True (T) False (F)
TranslationIPListRecentlyUpdated (TR)	True (T) False (F)
FaultRootCause (RC)	AllowedIPListsOutDated (A) DuplicatedIPInTranslationTable (D) WrongTranslation (W)

Table 4.11: Variables of the Problem Domain for the worked example.

destination IP address is known ({ DA:K }) and the destination machine is up and ready to offer its services ({ DU:T }). While those three variables are known with certainty, other set of variables are uncertain and admissible to discuss among all agents. That set is composed by the *assumptions* that represent the uncertainty of the belief of *Agent P* as a probability distribution. Those probability distribution are inferred using the *Causal Model* of *Agent P*, based on its background knowledge. The output of the inference process offers different probabilities for each variable: if the source machine is up or is down ({ SU:(T=0.7/F=0.3) }), if the list that contains all IP addresses allowed to use the service has changed recently ({ AR:(T=0.85/F=0.15) }) or if the translation service used to route has changed recently ({ TR:(T=0.4/F=0.6) }). Thus, based on the available *evidences* and those *assumptions*, *Agent P* proposes the most probable fault root cause is the list of allowed IP addresses is outdated and that *proposal* is added to the argument as a coherent statement ({ RC:(A=0.7/D=0.05/W=0.25)}).

Argument 1 *Sender: Agent P*

$E_{arg_1} \rightarrow$ `{SA=U:DU=T:DA=K}`

$A_{arg_1} \rightarrow$ `{AR=(T=0.85/F=0.15):SU=(T=0.7/F=0.3):TR=(T=0.4/F=0.6)}`

$P_{arg_1} \rightarrow$ `{RC=(A=0.7/D=0.05/W=0.25)}`

That initial argument is received by the rest of the constituents of the coalition (*Agent L* and *Agent M*). Then, *Agent M* processes that argument getting the *evidences* and comparing its own assumptions with the assumptions sent by *Agent P* in the initial argument. As *Agent M* knows a new evidence useful for this diagnosis case, it increases the *evidence set* with a new piece of information: the list of allowed IP addresses has not been updated recently ({ AR:F }). Then, *Agent M* generates a new argument (Argument 2) with an updated *evidence set*, its own assumptions in an updated *assumption set* and with its own new proposal of the fault root cause in the *proposal set*.

Argument 2 *Sender: Agent M*

$E_{arg_2} \rightarrow$ `{SA=U:DU=T:DA=K:AR=F}`

$A_{arg_2} \rightarrow$ `{SU=(T=0.6/F=0.4):TR=(T=0.55/F=0.45)}`

$P_{arg_2} \rightarrow$ `{RC=(A=0.05/D=0.5/W=0.45)}`

At this point, *Agent L* has received two arguments. It processes them and adds another new evidence: the source machine is up ({ SU:T }). Then, *Agent L* tries to get information about the status of variable *TR*, but that information is unreachable because the server is overloaded and it is not possible to get that information without stopping the service causing

a decrease of Quality of Service (QoS). Hence, that information is not available at diagnosis time and will be handled as an *assumption* during the argumentation. Thus, the updated *evidence set*, the assumption and the proposal of fault root cause of *Agent L* are sent in the Argument 3.

Argument 3 *Sender: Agent L*
$E_{arg_3} \rightarrow$ {SA=U:DU=T:DA=K:AR=F:SU=T}
$A_{arg_3} \rightarrow$ {TR=(T=0.45/F=0.55)}
$P_{arg_3} \rightarrow$ {RC=(A=0.05/D=0.7/W=0.25)}

When *Agent P* receives Arguments 2 and 3, and *Agent M* receives Argument 3, they process them and detect **discovery** attacks between those arguments. So, they accept the new evidences and generate two new arguments: Argument 4 and Argument 5, that contain the beliefs of *Agent P* and *Agent M* respectively. At this point, the *evidence sets* of Argument 3, 4 and 5 contain all available certain information about the diagnosis case exposed in this worked example. Thus, as all agents have sent their beliefs based on the same *evidence set*, they discuss now about their assumptions to get the most reliable proposal about the fault root cause.

Argument 4 *Sender: Agent P*
$E_{arg_4} \rightarrow$ {SA=U:DU=T:DA=K:AR=F:SU=T}
$A_{arg_4} \rightarrow$ {TR=(T=0.75/F=0.25)}
$P_{arg_4} \rightarrow$ {RC=(A=0.45/D=0.3/W=0.25)}

Argument 5 *Sender: Agent M*
$E_{arg_5} \rightarrow$ {SA=U:DU=T:DA=K:AR=F:SU=T}
$A_{arg_5} \rightarrow$ {TR=(T=0.85/F=0.15)}
$P_{arg_5} \rightarrow$ {RC=(A=0.05/D=0.8/W=0.15)}

At this point, we summarise the status of the argumentation as follows. Arguments 1 and 2 have been discarded and replaced by Arguments 4 and 5, respectively. Thus, Arguments 3, 4 and 5 represent the beliefs about the most probable fault root cause of agent L, P and M, respectively.

Using the *similarity* and *preferability* concepts defined in Sections 4.3.2.1 and 4.3.2.2 respectively, agents detect the statement about the variable TR in the assumption set of the Argument 5 ($st_{TR} \in \mathcal{A}_{arg5}$, simplified as $a_{5_{TR}}$) is *similar* to the one in Argument 4 ($a_{4_{TR}}$)

[6] and not *similar* to the one in Argument 3 ($a_{3_{TR}}$) [7]. At this point, *Agent M* holds the most *preferred* statement about *TR*. Thus, it generates a new argument (Argument 6) with a proposal for the probability distribution of the variable *TR*.

Argument 6 *Sender: Agent M*
$E_{arg_6} \rightarrow$`{SA=U:DU=T:DA=K:AR=F:SU=T}`
$A_{arg_6} \rightarrow \{\emptyset\}$
$P_{arg_6} \rightarrow$`{TR=(T=0.85/F=0.15)}`

When *Agent P* receives Argument 6, it agrees with the proposal and waits for any other argument. Thus, we say that Argument 6 supports Argument 4.

In contrast, when *Agent L* receives this argument, it finds that Argument 6 is a *clarification* for Argument 3. Thus, *Agent L* adds the received belief as input to the Bayesian inference process as soft-evidence (Pan et al., 2006), discards Argument 3 and sends a new argument with a new proposal inferred based on the updated beliefs (Argument 7). After this, as Argument 6 has achieved its commitment and it does not contain any proposal about a possible fault root cause, it is discarded too.

Argument 7 *Sender: Agent L*
$E_{arg_7} \rightarrow$`{SA=U:DU=T:DA=K:AR=F:SU=T}`
$A_{arg_7} \rightarrow$`{TR=(T=0.85/F=0.15)}`
$P_{arg_7} \rightarrow$`{RC=(A=0.05/D=0.9/W=0.05)}`

Finally, all available evidences have been discovered and all agents agree about the possible assumption (only variable *TR* in the example). Only *support* relations (between Arguments 5 and 7) and *contrariness* relations (between Arguments 4 and 5, and between Arguments 4 and 7) exist between arguments. Hence, all agents keep in silence because they do not receive any information that make them to change their beliefs.

After a time in silence longer than the **silence timeout**, *Agent P*, as *Manager Agent*, finishes the **Argumentation Phase** sending a notification to the coalition constituents and starts the **Conclusion Phase**.

As exposed in Section 4.4.3, *Agent P* filters the set of arguments to get the **candidate arguments set**, that, in this example, is composed by Arguments 4, 5 and 7. Summarising, there are three different proposals:

[6] $\Delta(a_{4_{TR}}, a_{5_{TR}}) = 0.08 < th = 0.2 \Rightarrow a_{4_{TR}}$ and $a_{5_{TR}}$ are *similar*.
[7] $\Delta(a_{3_{TR}}, a_{5_{TR}}) = 0.3 > th = 0.2 \Rightarrow a_{3_{TR}}$ and $a_{5_{TR}}$ are not *similar*.

- *Agent P* proposes $A = 0.45/D = 0.3/W = 0.25$ in Argument 4.
- *Agent M* proposes $A = 0.05/D = 0.8/W = 0.15$ in Argument 5.
- *Agent L* proposes $A = 0.05/D = 0.9/W = 0.05$ in Argument 7.

So, there is a conflict between *Agent P* and the team formed by *Agent M* and *Agent L*. At this point, several strategies can be applied to resolve the conflict, as proposed in Section 4.4.3. For example, let say that the resolution conflict strategy applied is that *the most reliable proposal* is picked as final conclusion. Then, the argumentation concludes when *Agent P* picks that the most reliable fault root cause is $D = 0.9$ (proposed by *Agent L* in Argument 7) that means there is a duplicated IP address in the translation table hosted in the *Agent L* domain. The **argumentation finished** message is sent to all messages in the coalition and the distributed fault diagnosis finishes.

4.6.4 Evaluation

This section presents a report of the experiments performed to assess the validity of the proposed argumentation framework as distributed hypothesis discrimination mechanism. The reminder tasks required to carry out the complete diagnosis process have been evaluated in the previous chapter. Therefore, this evaluation process is focus only on the distributed hypothesis discrimination task to get diagnosis conclusions using the proposed argumentative approach. Focusing only in this task, we consider the discrimination as a classification task. This is because, if we omit the reminder tasks validated in the previous chapter, we have that the discrimination process has the aim of identifying what fault root cause among a set of possible causes is generating the observed symptoms, which is essentially a classification task. Therefore, for evaluation purposes, we consider the proposed argumentation framework as a distributed multi-class classification technique and it is compared with other traditional classification techniques.

The rest of this section is structured as follows. Firstly, Section 4.6.4.1 presents the experimentation framework developed to carry out this evaluation process. Section 4.6.4.2 summarises the datasets used during the experimentation process. Finally, Section 4.6.4.3 shows the results of the experiments execution and discusses them.

4.6.4.1 Experimentation Framework

In order to provide an empirical assessment of the application of the proposed argumentative framework in the context of standard classification problems, a series of experiments

were conducted to compare the results of the proposed technique with other traditional classification techniques. The traditional techniques considered were decision trees, (such as J48, LADTree, or PART); support vector machines, (such as SMO); simple probabilistic classifiers (such as NBTree) and probabilistic graphs (such as BayesSearch). Most of the considered techniques are available in WEKA library[8]. In addition, SMILE library[9] was used because it provides algorithms not available in WEKA.

Contrary to the mentioned traditional centralised approaches, the proposed argumentative solution requires more than a data mining library to be executed. We have developed an experimentation framework that offers an environment to execute agents under federated domains conditions, such as access restrictions situations and different background knowledge for every agent. This framework is available as an open-source tool, named B2D2 ARgumentative Multi-Agent System (BARMAS) framework[10]. It uses SMILE library as Bayesian inference and learning engine to enable Argumentative Agents to reason with their *Causal Models* and MASON simulation framework[11] as agent platform. Notice that the considered traditional classification techniques follow a centralised approach, while the proposed argumentative framework has been designed as a distributed solution. Anyway, it is interesting to compare the results of those techniques for contexts where a distributed approach could be replaced by a centralised one assuming some extra costs, such as the requirement of high computing capacity in one single node, or sacrificing the data privacy because information must be stored in a central node for the whole system. Finally, Table 4.12 summarises the considered techniques and the respective software libraries used to execute them.

The validation were carried out with a cross-validation technique with a 10-Fold configuration. While for the validation of traditional centralised approaches, 90% of data were used for training and 10% for testing in 10 different iterations, for executing BARMAS framework, the training data is divided in many sets as argumentative agents are running in the experiment. Then, each agent has only a portion of the total training dataset to provide different background knowledge for every agent. For instance, in an execution with 2 argumentative agents, each one has only a 45% of the original dataset for training. The other 10% is used for testing. For 3 agents, 30%; for 4 agents, 22.5% and so on and so forth. To learn from training data, each argumentative agent performs a training process using the *BayesSearch* technique to synthesise the agent's background knowledge in a *Causal model*. To reproduce access restriction conditions, the set of variables is divided in many subsets as *Argumentative* agents, $V_1 \cup ... \cup V_n = V \setminus frc$ (excluding the classification target variables

[8]Weka Website: `http://www.cs.waikato.ac.nz/ml/weka/`
[9]SMILE Website: `http://genie.sis.pitt.edu/`
[10]Github repository of BARMAS framework: `https://github.com/gsi-upm/BARMAS`
[11]MASON Website: `cs.gmu.edu/~eclab/projects/mason/`

Classification Technique	**Acronym**	**Software Library**
J48 - implementation of C4.5 algorithm	J48	WEKA
Logical Analysis of Data (LAD) Tree	LADTree	WEKA
Pruning Rule based classification Tree	PART	WEKA
Sequential Minimal Optimization	SMO	WEKA
Naive Bayes Tree	NBTree	WEKA
Bayesian Search	BayesSearch	SMILE
B2D2 ARgumentaive Multi-Agent System	BARMAS	SMILE and MASON

Table 4.12: Summary of considered classification techniques.

frc). Each agent can access only to one of those subsets to reproduce the partial view of the global problem in federated domains.

In addition to the argumentative agents mentioned previously, some extra agents are used in the experiments to reproduce the conditions of the real-life scenario which motivated this work. Firstly, a *Generator* Agent is included in the experiments to generate diagnosis cases. In other words, it simulates the symptom detection task and triggers the distributed classification process. A *Manager* Agent is included to control the argumentation process as explained in Section 4.4. The *Manager* agent is configured to follow *the most certain conclusion* strategy, proposed in Section 4.4.3, as conflict resolution strategy. The *silence timeout* defined in Section 4.4.2, is configured to ensure that every agent can generate at least an argument. An *Evaluator* Agent is included to evaluate the conclusion of the argumentation process. This agent is notified by the *Manager* agent when the argumentation processed is finished to check the correctness of the classification process. Therefore, the following agents execute in all experiments: *Generator*, *Evaluator*, *Manager*, and a set of *Argumentative* agents. Finally, all *Argumentative* Agents are configured with a *threshold* value equals to 0.2 ($th = 0.2$) to measure the similarity between statements, as explained in Sections 4.3.

4.6.4.2 Datasets

A number of public datasets were used for the evaluation as well as the private one extracted from the case study presented in Section 3.5.1, which contains fault diagnosis data of a real-live telecommunication network running for one and a half years. Those datasets are

used to measure the accuracy of the proposed approach for the multi-class classification problem. Public datasets have been collected from UCI[12] and KEEL[13] repositories and have a meaningful difference among their characteristics with respect to number of classes, number of instances and number of attributes. An overview of the complexity of the considered datasets is shown in Table 4.13.

Dataset	**#Instances**	**#Classes**	**#Attributes**
Zoo [14]	101	7	16
Solar Flare [15]	1066	6	11
Marketing [16]	8933	9	13
Nursery [17]	12690	5	9
Mushroom [18]	8124	2	22
Chess [19]	28056	18	6
Network [20]	1183	15	27

Table 4.13: Datasets Summary.

4.6.4.3 Results

To measure the accuracy of the considered classification techniques, we present the Error Rate (ER) values obtained for every dataset under different uncertainty levels. That uncertainty was generated hiding some variables of datasets to reproduce a crucial aspect of the motivational problem, i.e. uncertainty during fault diagnosis of telecommunication networks. In other words, uncertainty was reproduced in the experiments using missing attributes for the classification techniques. Three configurations were used to generate different uncertainty levels: no missing attributes, 25% of missing attributes and 50% of missing

[12]UCI Repository Website: `http://archive.ics.uci.edu/ml/datasets.html`
[13]KEEL Repository Website: `http://sci2s.ugr.es/keel/datasets.php`
[14]Zoo Dataset: `http://sci2s.ugr.es/keel/dataset.php?cod=69`
[15]Solar Flare Dataset: `http://sci2s.ugr.es/keel/dataset.php?cod=98`
[16]Marketing Dataset: `http://sci2s.ugr.es/keel/dataset.php?cod=163`
[17]Nursery Dataset: `http://sci2s.ugr.es/keel/dataset.php?cod=103`
[18]Mushroom Dataset: `http://archive.ics.uci.edu/ml/datasets/Mushroom`
[19]Chess Dataset: `http://archive.ics.uci.edu/ml/datasets/Chess+%28King-Rook+vs.+King%29`
[20]Private dataset with Telefónica O2 Czech Republic rights.

attributes[21]. The following tables show the results of the considered traditional classification techniques (BayesSearch, J48, LADTree, NBTree, PART and SMO) and the proposed argumentative technique (BARMAS) with different numbers of agents involved in the argumentation process (2, 3 and 4). Notice the results of different tables should not be compared between them, because the results with no uncertainty will be, in general, better than the results with a 25% or a 50% of missing values, i.e. with uncertainty.

Table 4.14 shows the results with no missing attributes[22]. That implies *Argumentative* agents make no assumptions during the Argumentation Phase because all variables are known with certainty. By analysing these results, we observed the results of BARMAS are comparable with the rest of the traditional techniques. It would suggest that, generally speaking, the use of BARMAS as a distributed approach produces similar results to other centralised alternatives in non-uncertain situations.

Dataset	**BARMAS**			**Bayes**	**J48**	**LAD**	**NB**	**PART**	**SMO**
	#Agents			**Search**		**Tree**	**Tree**		
	2	**3**	**4**						
Zoo	**0.00**	0.03	**0.00**	0.02	0.02	0.01	0.01	0.02	0.01
Solar Flare	0.29	**0.26**	**0.26**	0.36	**0.26**	0.27	**0.26**	0.29	**0.26**
Marketing	0.71	0.71	0.71	0.71	0.7	0.67	0.68	0.70	**0.66**
Nursery	0.07	0.08	0.09	0.07	**0.01**	0.08	0.03	**0.01**	0.07
Mushroom	0.01	**0.00**	0.01	0.01	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
Chess	0.5	0.62	0.64	0.61	0.42	0.69	**0.39**	0.46	0.56
Network	0.16	0.16	0.16	0.18	0.13	0.14	0.14	0.14	**0.12**

Table 4.14: Error Rate without uncertainty (all available data).

Tables 4.15 and 4.16 show the results adding uncertainty to the experiment, i.e. removing the 25% and the 50% of the attributes of the classification case as unknown information (missing attributes) that agents can discuss.

Closer inspection of Table 4.15 revealed that the difference between columns becomes to be significant in uncertain situations for some datasets. For example, focusing in the

[21] We consider uncertainty levels above 50% are quite improbable in real-life scenarios and would conclude unreliable conclusions.

[22] Notice that values are truncated and values equals to 0.00 do not mean a perfect classification. They represent values between 0.00 and 0.01.

Mushroom row of Table 4.15, we observed that *BARMAS* ($0.01 \sim 0.02$), *BayesSearch* (0.03) and *NBTree*[23] (0.04) have quite low ER compared with other alternatives, such as *J48* (0.26), *LADTree* (0.52) *PART* (0.23) or *SMO* (0.23). Other example can be observed in Zoo row of the same table. In contrast, all compared alternatives present similar results in the rest of the datasets.

Dataset	**BARMAS**			**Bayes**		**LAD**	**NB**		
	#Agents			**Search**	**J48**	**Tree**	**Tree**	**PART**	**SMO**
	2	**3**	**4**						
Zoo	**0.09**	0.12	0.11	0.11	0.43	0.4	0.16	0.43	0.41
Solar Flare	0.58	**0.56**	0.6	0.62	0.68	0.67	0.58	0.68	0.79
Marketing	0.70	0.71	0.71	0.71	0.7	0.73	**0.67**	0.70	0.69
Nursery	0.25	0.26	0.26	0.25	**0.24**	0.27	**0.24**	0.26	0.32
Mushroom	0.02	**0.01**	**0.01**	0.03	0.26	0.52	0.04	0.23	0.23
Chess	0.64	0.73	0.77	0.68	**0.62**	0.86	0.64	0.68	0.75
Network	0.16	0.17	0.16	0.17	**0.14**	**0.14**	0.15	0.15	0.18

Table 4.15: Error Rate with 25% of missing attributes.

From Table 4.16, it is apparent that BARMAS has lower (or at least equal) error rate than the other alternatives. This attests to the accuracy of BARMAS in situations with high uncertainty (50% of missing attributes) is preferable to other alternatives. Furthermore, it offers the flexibility to be applied in distributed environments with private knowledge, as mentioned previously. As data in Table 4.16 shows, the results for Zoo or Mushroom datasets present, again, a significant improvement using Bayesian approaches with differences up to 0.49 for the ER values[24]. Contrarily, comparable values are observed (with equal uncertainty levels) in other datasets, such as Solar Flare, Marketing, Nursery, Chess or Network.

In conclusion, one of the most important consequences of the analysed results is the robustness of the proposed approach against uncertainty that can be observed focusing on the same dataset of Tables 4.14 and 4.16. For example, at one end, the experiment with 4 BARMAS agents presents only a 0.02 difference between situations with no uncertainty ($ER = 0.01$) and 50% of missing attributes ($ER = 0.03$) for Mushroom dataset. In contrast, at the other end, *LADTree* presents a difference equals to 0.52 ($ER = 0.00$ with

[23] All of them are Bayesian approaches.

[24] Between BARMAS with 4 agents (0.03) and LADTree (0.52) for Mushroom dataset, shown in Table 4.16.

Dataset	BARMAS			Bayes Search	J48	LAD Tree	NB Tree	PART	SMO
	#Agents								
	2	3	4						
Zoo	**0.13**	0.18	0.19	0.16	0.51	0.59	0.28	0.53	0.6
Solar Flare	**0.63**	0.64	**0.63**	0.65	0.69	0.69	**0.63**	0.69	0.8
Marketing	**0.72**	**0.72**	**0.72**	**0.72**	**0.72**	0.89	0.73	**0.72**	0.83
Nursery	**0.27**	**0.27**	**0.27**	**0.27**	**0.27**	**0.27**	**0.27**	0.28	0.32
Mushroom	0.04	0.04	**0.03**	0.05	0.32	0.52	0.22	0.41	0.21
Chess	**0.72**	0.76	0.78	0.74	**0.72**	0.87	0.73	0.74	0.78
Network	0.22	0.22	**0.21**	0.24	0.23	0.23	0.28	0.23	0.3

Table 4.16: Error Rate with 50% of missing attributes.

no uncertainty and $ER = 0.52$ with 50% of missing attributes.). Thus, broadly speaking, the experiments results attest that BARMAS provides a suitable mechanism to perform distributed fault diagnosis under uncertainty in federated domains.

4.7 Summary

This chapter has proposed an extension of the B2D2 Agent Architecture to add argumentative capability. That capability enables to perform distributed fault diagnosis in federated domains. The argumentation process grounds on the proposed B2D2 Argumentative Framework. An *Argumentation Model* has been developed extending the knowledge models of the B2D2 Agent Architecture. This model covers concepts required to perform a distributed fault diagnosis applying the proposed coordination protocol. That protocol is composed by three different phases: *Coalition Formation Phase*, *Argumentation Phase* and *Conclusion Phase*, which are carried out by two different agent roles: *Argumentative* and *Manager* agents. A set of *Argumentative* agents is able to argue about the discrimination of a detected anomaly or symptom in the network. Those agents interchange arguments that contain information about a diagnosis case until a *Manager* agent decides the argumentation is finished and extracts the final conclusion, i.e. the fault root cause of the diagnosed problem. The proposed argumentation framework grounds on the idea of probabilistic statements to handle the uncertainty of the diagnosis process. Every statement type contains different type of knowledge: *evidences* contain certain knowledge, *assumptions* contain uncertain knowledge

and *proposals* represent propositions about a specific information, such as a fault root cause.

The proposed B2D2 Argumentative Agent Architecture has been evaluated in a simulation framework. The evaluation has been performed considering a B2D2 ARgumentative Multi-Agent System (BARMAS) as a multi-class classification technique with other traditional classification techniques under different conditions of uncertainty. Moreover, different public and private datasets were used during the experiment execution to provide results for different domains. The results attest the proposed argumentative extension of the B2D2 Agent Architecture provides a suitable mechanism to perform distributed fault diagnosis under uncertainty.

Finally, the proposed B2D2 Argumentative Agent Architecture defines coordination mechanisms to perform autonomic fault diagnosis in federated domains. The argumentative capability adds some interesting features to the basic B2D2 Agent Architecture presented in the previous chapter: (i) it enables to perform distributed hypothesis discrimination keeping coherence with high robustness against uncertainty, (ii) it enables to keep private knowledge among agents in federated domains, (iii) it provides cooperation mechanism to reach agreements obtaining conclusions for cross-domain diagnosis cases and (iv) it enables agents to be deployed dynamically in complex environments as coalitions are conformed in execution time as required.

CHAPTER 5

Conclusions and Future Research

This thesis has presented a number of solutions that contribute to the area of Application of Agent Technology for Fault Diagnosis of Telecommunications Networks. In this chapter, those contributions are summarised and final conclusions are presented. Furthermore, having taken account of the achievements of the thesis, this chapter presents possible future research lines.

5.1 Conclusions

The research presented in this thesis was set out with the goal to propose a solution for the problem of fault diagnosis for the Future Internet following an autonomic approach. Although some autonomic solutions begin to be more popular in the community for some network management tasks, such as monitoring or routing, the lack of autonomic solutions for fault management motivated us to research on that topic. Specifically, this thesis proposes the Application of Agent Technology for Fault Diagnosis of Telecommunications Networks. Throughout the course of the thesis, a number of contributions have been delivered that can be gathered under **three main contribution areas**:

Knowledge Gathering for Autonomic Fault Diagnosis. The thesis proposed the BEAST Methodology which is focused on knowledge gathering from human experts to enable the design of Autonomic Fault Diagnosis solutions. The methodology was evaluated during the development of a MAS for a Fault Diagnosis task of FTTH networks. Based on that work, the final contributions in this area are an agile acceptance testing methodology for multi-agent systems and an open-source tool to support the application of the methodology on software development projects. The final conclusions obtained via evaluation of those contributions are: 1) the BDD approach is suitable for its application in MAS development; 2) the use of BDD templates facilitates the communication between stakeholders and designers or developers, which is usually a gap between both of them; 3) the application of mock agents simplifies MAS development in agile environments.

Agent Architecture for Autonomic Fault Diagnosis. The thesis proposed the B2D2 Agent Architecture based on an expert knowledge model for fault diagnosis of telecommunication networks. The proposed architecture has been validated in a real-life telecommunication network and in a simulation environment. The results of the fault diagnosis system of a enterprise service running for one and a half years were analysed to measure the outcomes of the solution. Furthermore, the thesis has contributed with a knowledge model for fault diagnosis by the means of an ontology and connected this model to other domain models which describes different characteristic of a telecommunication network. The final conclusions obtained via evaluation of the presented contributions are: 1) the utilised technology can reduce the time to diagnose in the real-life network management tasks; 2) the use of different knowledge models can offer complementary views of the fault diagnosis problem; 3) the use of hybrid reasoning techniques can improve the correctness of diagnosis conclusions.

Coordination for Autonomic Fault Diagnosis in Federated Domains. The thesis proposed the B2D2 Argumentation Framework as coordination mechanism among agents. This framework was designed to perform distributed hypothesis discrimination in federated domains. The thesis contributed with a coordination protocol to manage the argumentation process among agents. This argumentative capability was included in the proposed agent architecture. Moreover, an argumentation model has been contributed by the means of an ontology to enable agents to add this capability to their knowledge models. The conclusions reached after the experimentation process are: 1) the argumentative framework is suitable to perform distributed hypothesis discrimination in federated domains; 2) it offers high robustness against uncertainty; 3) the coordination solution ensures the scalability of the proposed agent architecture for complex environments.

The order in which the contributions have been presented was not accidental. Each of the consecutive contributions builds on the outcomes of the previous. The firstly proposed knowledge gathering process using the BEAST Methodology is the basis to feed the knowledge models proposed for the B2D2 Agent Architecture for specific networks. Similarly, that B2D2 Agent Architecture is the basis of the B2D2 Argumentative Agent Architecture which extends the possible application of the autonomic fault diagnosis solution to federated environments where multiple agents have to cooperate to achieve a common goal.

5.2 Future Research

The development of this thesis and its contributions to the state of the art in Application of Agent Technology for Fault Diagnosis of Telecommunications Networks have opened new possibilities for future research. The experiments conducted have proved usefulness of certain solutions or excluded particular approaches. Additionally, the related software prototypes and tools have stimulated the development of new ideas for improving the state of the art. In terms of conclusions for the thesis research, the following lines of future research can be pointed out as follows.

The testing methodology proposed in this thesis defines a phase where the whole multi-agent system is evaluated while running in a testing environment. The methodology considers that environment can be a simulated scenario that reproduces some features of the real scenario where the MAS will be deployed. But, it would be desirable a further analysis of the requirements of those simulated environments. That would be specially interesting for the development of multi-agent systems in complex environments where the development of realistic simulations of the environment could require a lot of effort and time. Therefore,

the use of **simulated environments for testing** could be more deeply explored and analysed to define some techniques or design principles for the development of those simulations based on the requirements of the system. But that does not mean the testing methodology should specify how to design the agents of the solution, because the aim of the methodology is not focused on designing multi-agent systems but in acceptance testing. So, it does not depend of any multi-agent design methodology. Nevertheless, some guidelines could be provided to associate user stories gathered as requirements with the development of a solution following a specific MAS design methodology, such as MAS-CommonKADS (Iglesias et al., 1998), Ingenias (Pavon et al., 2005), Prometheus (Padgham and Winikoff, 2003), or Gaia (Wooldridge et al., 2000). In other words, further research could be done to specify a connection between the *System Behaviour Specification* phase and the *MAS Behaviour Specification* phase through a **specific MAS design methodology**. Moreover, the methodology specifies the metrics collected in testing environments are used to evaluate the behaviour of the global system and redesign the solution if required. But, it would be interesting to provide some guidelines about what impact could have specific types of metrics in the design of the solution in future iterations of the project. So, further research could be done to measure the **impact** of **metrics** in the **design** of the solution.

The thesis proposed the use of the testing methodology to extract knowledge from expert network engineers to feed the domain models required for the autonomic fault diagnosis solution proposed as an agent architecture. Notwithstanding, the definition of a process to extract specific key information from **human experts** would be desirable for facilitating the **knowledge gathering** process. For instance, the knowledge related with the structural model of the network used by the proposed fault diagnosis agent architecture. Regarding with that structural model, the evolution of **standard network description languages** must be followed with special attention. The thesis proposed the use of INDL language based on the standardised Network Markup Language (NML) to instantiate that domain knowledge model, but other alternatives could arise to be *de facto* standards, which could promoted the existence of **knowledge repositories**. Those repositories will be a valuable resource to extend the knowledge of the agent with descriptions of network elements providing specific knowledge for every diagnosis case.

Besides, the agent architecture could be extended to add other interesting autonomic capabilities further than Autonomic Fault Diagnosis. Among all the possibilities that the autonomic networking presents, we could highlight some of them, such as a combination of **self-discovery** and **self-learning** techniques to learn passively the behaviour of the network detecting fault symptoms and exploring how to resolve them. This function could be combined with an interface with human operators to ask them if the detected behaviours

are correct or not and if the planned strategy to solve those faults is suitable or not. Other interesting autonomic capability is the **self-protection** for telecommunication networks. It could provide dynamic and adaptive mechanisms to react to malicious behaviours of systems or deliberate user attacks. This could be an interesting future work because the attack detection and classification task is not really different to the fault diagnosis tasks covered in this thesis. Thus, the knowledge models used by the fault diagnosis agent could be adapted to design a self-protection capability for autonomic management systems.

As exposed in the thesis, the agent architecture has been validated in a real-life telecommunication network. In that case, the agent interacts with some information systems which filtered low-level network information and offered high-level events. However, the analysis of **low-level data** could be interesting for some management tasks, such as real-time monitoring or reconfiguration tasks. For that, the possibilities of the **emerging network architectures**, such as Software-Defined Networking (SDN), offer a set of dynamic, manageable and adaptable capabilities that previously was difficult to implement in the traditional network architectures. This makes the application of self-management systems is an interesting research topic in the area. For that, further research for the application of the proposed autonomic fault diagnosis solutions in a **SDN environment** would be interesting for network operator companies. Furthermore, the agent architecture could be validated in a network scenario which combines low-level network diagnosis with high-level diagnosis interacting with third-party information systems.

The thesis proposed a coordination mechanism based on argumentative techniques. The proposed solution could be extended adding **reputation metrics** in the argumentation framework proposed for distributed diagnosis in federated domains. The concept of reputation could be used to enable a new level in the **argument acceptance** process proposed in the framework. The agent reputation could be a decision factor to accept or reject a specific argument during the argumentation process. That could be a mechanisms to detect agents with a high rate of incorrect diagnosis conclusions and improve their background knowledge to get better diagnosis accuracy. This argumentation framework relies in the idea of agents supervise different domains of the network, but it would be interesting to provide some discovery mechanisms to find what domains are related with specific fault types. Thus, further research on **domains specifications** for federated domains could be followed to enable some **discovery mechanisms** among those domains.

Concluding the presented lines of future work: the thesis has investigated and proposed solutions for Autonomic Fault Diagnosis of Telecommunication Networks applying Agent Technology. As pointed out in the solution outline presented at the beginning of this dis-

sertation, all the contributions are interconnected and dependent on each other. Therefore, aside of answering new questions that the thesis rose, future work should investigate further the impact of thesis contributions on autonomic management solutions, with special interest in its application in emerging network architectures and research on how the proposed solutions could be implanted in combination with other autonomic management solutions to realise the ambitious idea of autonomic networking for the Future Internet.

Bibliography

C. E. Abosi, R. Nejabati, and D. Simeonidou. Design and development of a semantic information modelling framework for a service oriented optical internet. In *Transparent Optical Networks, 2009. ICTON'09. 11th International Conference on*, pages 1–4. IEEE, 2009.

G. Adzic. *Bridging the Communication Gap: Specification by Example and Agile Acceptance Testing*. Neuri Limited, United Kingdom, 2009.

G. Adzic. *Specification by Example: How Successful Teams Deliver the Right Software*. Manning Publications Co., Greenwich, CT, USA, 1st edition, 2011.

N. Agarwal and U. Rathod. Defining success for software projects: An exploratory revelation. *International Journal of Project Management*, 24(4):358 – 370, 2006.

N. Agoulmine. Chapter 1 - Introduction to Autonomic Concepts Applied to Future Self-Managed Networks. In *Autonomic Network Management Principles*, pages 1 – 26. Academic Press, Oxford, 2011.

J. P. Almeida, M. E. Iacob, and P. Eck. Requirements traceability in model-driven development: Applying model and transformation conformance. *Information Systems Frontiers*, 9(4):327–342, 2007.

E. Alonso. Rights and argumentation in open multi-agent systems. *Artificial Intelligence Review*, 21(1):3–24, 2004.

L. Amgoud, N. Maudet, and S. Parsons. Modelling dialogues using argumentation. In *MultiAgent Systems, 2000. Proceedings. Fourth International Conference on*, pages 31–38, 2000.

L. Amgoud. An argumentation-based model for reasoning about coalition structures. In *Argumentation in Multi-Agent Systems*, volume 4049 of *Lecture Notes in Computer Science*, pages 217–228. Springer Berlin Heidelberg, 2006.

L. Amgoud and C. Cayrol. On the acceptability of arguments in preference-based argumentation. In *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*, UAI'98, pages 1–7, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

L. Amgoud and S. Parsons. Agent dialogues with conflicting preferences. In *Intelligent Agents VIII*, volume 2333 of *Lecture Notes in Computer Science*, pages 190–205. Springer Berlin Heidelberg, 2002.

L. Amgoud and H. Prade. Using arguments for making and explaining decisions. *Artificial Intelligence*, 173(3-4):413 – 436, 2009.

L. Amgoud and M. Serrurier. Arguing and explaining classifications. In *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '07, pages 160:1–160:7, New York, NY, USA, 2007. ACM.

L. Amgoud and M. Serrurier. Agents that argue and explain classifications. *Autonomous Agents and Multi-Agent Systems*, 16(2):187–209, 2008.

L. Amgoud, J.-F. Bonnefon, and H. Prade. An argumentation-based approach to multiple criteria decision. In *Symbolic and Quantitative Approaches to Reasoning with Uncertainty*, volume 3571 of *Lecture Notes in Computer Science*, pages 269–280. Springer Berlin Heidelberg, 2005.

L. Amgoud, Y. Dimopoulos, and P. Moraitis. A general framework for argumentation-based negotiation. In *Argumentation in Multi-Agent Systems*, volume 4946 of *Lecture Notes in Computer Science*, pages 1–17. Springer Berlin Heidelberg, 2008.

M. Aulinas, P. Tolchinsky, C. Turon, M. Poch, and U. Cortés. Argumentation-based framework for industrial wastewater discharges management. *Engineering Applications of Artificial Intelligence*, 25(2):317 – 325, 2012.

P. Bedi and P. Vashisth. Empowering recommender systems using trust and argumentation. *Information Sciences*, 279(0):569 – 586, 2014.

M. Behringer, M. Pritikin, S. Bjarnason, A. Clemm, B. Carpenter, S. Jiang, and L. Ciavaglia. Autonomic networking: Definitions and design goals. Technical report, Internet Research Task Force, 2015.

F. L. Bellifemine, G. Caire, and D. Greenwood. *Developing Multi-Agent Systems with JADE*, volume 5 of *Wiley Series in Agent Technology*. Wiley, 2007.

T. J. M. Bench-capon, S. Doutre, and P. E. Dunne. Value-based argumentation frameworks. In *Artificial Intelligence*, pages 444–453, 2002.

T. J. Bench-Capon and P. E. Dunne. Argumentation in artificial intelligence. *Artificial intelligence*, 171(10):619–641, 2007.

R. Benjamins. Problem-solving methods for diagnosis and their role. *International Journal of Expert Systems: Research and Applications*, 8(2):93–120, 1995.

A. Bondarenko, F. Toni, and R. A. Kowalski. An assumption-based framework for non-monotonic reasoning. In *Proceedings of the Second International Workshop on Logic Programming and Non-Monotonic Reasoning*, pages 171–189. MIT Press, 1993.

W. Borutzky. *Bond graph methodology: development and analysis of multidisciplinary dynamic system models*. Springer Science & Business Media, 2010.

G. Bouabene, C. Jelger, C. Tschudin, S. Schmid, A. Keller, and M. May. The autonomic network architecture (ANA). *Selected Areas in Communications, IEEE Journal on*, 28 (1):4–14, 2010.

P. Brandao Neto, A. Rocha, and H. Lopes Cardoso. Risk assessment through argumentation over contractual data. In *Information Systems and Technologies (CISTI), 2013 8th Iberian Conference on*, pages 1–6, 2013.

L. Braubach, A. Pokahr, and W. Lamersdorf. Jadex: A BDI-Agent System Combining Middleware and Reasoning. In *Software Agent-Based Applications, Platforms and Development Kits*, Whitestein Series in Software Agent Technologies and Autonomic Computing, pages 143–168. Birkhäuser Basel, 2005.

P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, and J. Mylopoulos. Tropos: An agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems*, 8:203–236, 2004.

N. Bulling, J. Dix, and C. I. Chesñevar. Modelling Coalitions: ATL + Argumentation. In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 2*, AAMAS '08, pages 681–688. International Foundation for Autonomous Agents and Multiagent Systems, 2008.

T. R. Burchfield, S. Venkatesan, and D. Weiner. Maximizing throughput in zigbee wireless networks through analysis, simulations and implementations. In *Proc. Int. Workshop Localized Algor. Protocols WSNs*, pages 15–29, 2007.

X. Caiquan, W. Chunzhi, M. Qing, and S. Xianbin. An argumentation-based interaction model and its algorithms in multi-agent system. In *International Conference on Intelligent Computation Technology and Automation (ICICTA), 2010*, volume 1, pages 493–496, 2010.

A. Carrera and C. A. Iglesias. A systematic review of argumentation techniques for multi-agent systems research. *Artificial Intelligence Review*, 44(4):509–535, 2015.

V. Cerf. Abstraction, federation, and scalability. *Internet Computing, IEEE*, 17(1):96–c3, 2013.

E. K. Cetinkaya, D. Broyles, A. Dandekar, S. Srinivasan, and J. Sterbenz. Modelling communication network challenges for future internet resilience, survivability, and disruption tolerance: a simulation-based approach. *Telecommunication Systems*, 52(2):751–766, 2013.

W. K. Chai, A. Galis, M. Charalambides, and G. Pavlou. Federation of future internet networks. In *Network Operations and Management Symposium Workshops (NOMS Wksps), 2010 IEEE/IFIP*, pages 209–216, 2010.

M. Charalambides, G. Pavlou, P. Flegkas, N. Wang, and D. Tuncer. Managing the future internet through intelligent in-network substrates. *Network, IEEE*, 25(6):34–40, 2011.

H. K. Chow, W. Siu, C.-K. Chan, and H. C. Chan. An argumentation-oriented multi-agent system for automating the freight planning process. *Expert Systems with Applications*, 40 (10):3858 – 3871, 2013.

CIM Specification Group. Common Information Model. `http://www.dmtf.org/standards/cim`, 2015. Accessed December 15, 2015.

D. Clark, S. Shenker, and A. Falk. GENI Research Plan (Version 4.5). *GENI Research Coordination Working Group and GENI Planning Group. GDD*, pages 06–28, 2007.

N. Clynch and R. Collier. SADAAM: Software agent development-an agile methodology. In *Proceedings of the Workshop of Languages, methodologies, and Development tools for multi-agent systems (LADS07), Durham, UK*, 2007.

R. Coelho, U. Kulesza, A. von Staa, and C. Lucena. Unit testing in multi-agent systems using mock agents and aspects. In *Proceedings of the 2006 international workshop on Software engineering for large-scale multi-agent systems*, SELMAS '06, pages 83–90, New York, NY, USA, 2006. ACM.

D. P. Cox, Y. Al-Nashif, and S. Hariri. Application of autonomic agents for global information grid management and security. *Summer Computer Simulation Conference*, pages 1147–1154, 2007.

S. Das. Symbolic argumentation for decision making under uncertainty. In *Information Fusion, 2005 8th International Conference on*, volume 2, pages 8 pp.–, 2005.

A. S. dAvila Garcez, D. M. Gabbay, and L. C. Lamb. A neural cognitive model of argumentation with application to legal inference and decision making. *Journal of Applied Logic*, 12(2):109 – 127, 2014.

A. Dempster. Upper and lower probabilities induced by a multi-valued mapping. *The Annals of Statistics*, 28:325–339, 1967.

A. Di Pietro, F. Huici, D. Costantini, and S. Niccolini. DECON: Decentralized Coordination for Large-Scale Flow Monitoring. In *INFOCOM IEEE Conference on Computer Communications Workshops , 2010*, pages 1–5, 2010.

O. Dikenelli, R. C. Erdur, and O. Gumus. SEAGENT: a platform for developing semantic web based multi agent systems. In *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, AAMAS '05, pages 1271–1272, New York, NY, USA, 2005. ACM.

M. J. Druzdzel. SMILE: Structural Modeling, Inference, and Learning Engine and GeNIe: a development environment for graphical decision-theoretic models. In *AAAI/IAAI*, pages 902–903, 1999.

P. Duarte, J. Nobre, L. Granville, and L. Rockenbach Tarouco. A P2P-Based self-healing service for network maintenance. In *Integrated Network Management (IM), 2011 IFIP/IEEE International Symposium on*, pages 313–320, 2011.

P. M. Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial intelligence*, 77(2):321–357, 1995.

A. El-Sisi and H. Mousa. Argumentation based negotiation in multiagent system. In *7th International Conference on Computer Engineering Systems (ICCES), 2012.*, pages 261–266, 2012.

D. Evans. The internet of things: How the next evolution of the internet is changing everything. *CISCO white paper*, 1:14, 2011.

E. Evans. *Domain-driven design: tackling complexity in the heart of software*. Addison-Wesley Professional, 2004.

X. Fan, R. Craven, R. Singer, F. Toni, and M. Williams. Assumption-based argumentation for decision-making with preferences: A medical case study. In *Computational Logic in Multi-Agent Systems*, volume 8143 of *Lecture Notes in Computer Science*, pages 374–390. Springer Berlin Heidelberg, 2013.

X. Fan, F. Toni, A. Mocanu, and M. Williams. Dialogical two-agent decision making with assumption-based argumentation. In *Proceedings of the 2014 International Conference on Autonomous Agents and Multi-agent Systems*, AAMAS '14, pages 533–540, Richland, SC, 2014. International Foundation for Autonomous Agents and Multiagent Systems.

S. P. Ferrando and E. Onaindia. Context-aware multi-agent planning in intelligent environments. *Information Sciences*, 227:22 – 42, 2013.

E. T. Feteris. *Fundamentals of legal argumentation: A survey of theories on the justification of judicial decisions*, volume 1. Boom Koninklijke Uitgevers, 1999.

J. FitzGerald and A. Dennis. *Business Data Communications and Networking*. John Wiley and Sons, 2008.

D. Fogli, M. Giacomin, F. Stocco, and F. Vivenzi. Supporting medical discussions through an argumentation-based tool. In *Proceedings of the Biannual Conference of the Italian Chapter of SIGCHI*, CHItaly '13, pages 18:1–18:10, New York, NY, USA, 2013. ACM.

D. Gaertner and F. Toni. Computing arguments and attacks in assumption-based argumentation. *Intelligent Systems, IEEE*, 22(6):24–33, 2007.

D. Gaertner and F. Toni. Preferences and assumption-based argumentation for conflict-free normative agents. In *Argumentation in Multi-Agent Systems*, volume 4946 of *Lecture Notes in Computer Science*, pages 94–113. Springer Berlin Heidelberg, 2008.

A. Galis, H. Abramowicz, M. Brunner, D. Raz, P. Chemouil, J. Butler, C. Polychronopoulos, S. Clayman, H. De Meer, T. Coupaye, et al. Management and service-aware networking architectures (MANA) for future Internet - Position paper: System functions, capabilities and requirements. In *Communications and Networking in China, 2009. ChinaCOM 2009. Fourth International Conference on*, pages 1–13. IEEE, 2009.

I. García-Magariño, A. Gómez-Rodríguez, J. Gómez-Sanz, and J. González-Moreno. Ingenias-SCRUM development process for multi-agent development. In *International Symposium on Distributed Computing and Artificial Intelligence 2008 (DCAI 2008)*, pages 108–117. Springer, 2009.

Z. Ge, J. Guo-rui, and H. Ti-yun. Design of argumentation-based multi-agent negotiation system oriented to e-commerce. In *International Conference on Internet Technology and Applications, 2010.*, pages 1–6, 2010.

M. Ghijsen, J. Van Der Ham, P. Grosso, C. Dumitru, H. Zhu, Z. Zhao, and C. De Laat. A semantic-web approach for modeling computing infrastructures. *Computers & Electrical Engineering*, 39(8):2553–2565, 2013.

J. Gómez-Sanz, J. Botía, E. Serrano, and J. Pavón. Testing and Debugging of MAS Interactions with INGENIAS. In *Agent-Oriented Software Engineering IX*, volume 5386 of *Lecture Notes in Computer Science*, pages 199–212. Springer Berlin / Heidelberg, 2009.

M. A. Grando, D. Glasspool, and A. Boxwala. Argumentation logic for the flexible enactment of goal-based medical guidelines. *Journal of Biomedical Informatics*, 45(5):938 – 949, 2012.

J. Guckenheimer and J. M. Ottino. Foundations for complex systems research in the physical sciences and engineering. In *Report from an NSF Workshop in September*, 2008.

C. Guerra-García, I. Caballero, and M. Piattini. Capturing data quality requirements for web applications by means of dqwebre. *Information Systems Frontiers*, 15(3):433–445, 2013.

P. Hamill. *Unit test frameworks.* O'Reilly, 1 edition, 2004.

A. Hartfelt, A. S. Järvinen, and M. A. S. Vila. Rule-based argumentation. Master's thesis, ITU, 2010.

P. Harvey, C. Chang, and A. Ghose. Support-based distributed search: A new approach for multiagent constraint processing. In *Argumentation in Multi-Agent Systems*, volume 4766 of *Lecture Notes in Computer Science*, pages 91–106. Springer Berlin Heidelberg, 2007.

B. Haugset and G. Hanssen. Automated acceptance testing: A literature review and an industrial case study. In *Agile, 2008. AGILE '08. Conference*, pages 27–38, 2008.

S. Heras, J. Jordán, V. Botti, and V. Julián. Case-based strategies for argumentation dialogues in agent societies. *Information Sciences*, 223(0):1 – 30, 2013a.

S. Heras, J. Jordán, V. Botti, and V. Julián. Argue to agree: A case-based argumentation approach. *International Journal of Approximate Reasoning*, 54(1):82 – 108, 2013b.

Z. Houhamdi. Multi-agent system testing: A survey. *International Journal of Advanced Computer Science and Applications (IJACSA)*, 2(6):135–141, 2011.

L. Hsairi, K. Ghedira, M. Alimi, and A. Benabdelhafid. Resolution of conlicts via argument based netotiation: Extended enterprise case. In *International Conference on Service Systems and Service Management, 2006.*, volume 1, pages 828–833, 2006.

L. Hsairi, K. Ghedira, A. Alimi, and A. Benabdelhafid. Trust and reputation model for R2-IBN framework. In *IEEE International Conference on Systems Man and Cybernetics (SMC), 2010.*, pages 2137–2144, 2010.

S.-L. Huang and C.-Y. Lin. The search for potentially interesting products in an e-marketplace: An agent-to-agent argumentation approach. *Expert Systems with Applications*, 37(6):4468 – 4478, 2010.

C. A. Iglesias, M. Garijo, J. C. González, and J. R. Velasco. Analysis and design of multiagent systems using mas-commonkads. In *Intelligent Agents IV Agent Theories, Architectures, and Languages*, volume 1365 of *Lecture Notes in Computer Science*, pages 313–327. Springer Berlin Heidelberg, 1998.

N. K. Janjua and F. K. Hussain. WebIDSS - Argumentation-enabled Web-based IDSS for reasoning over incomplete and conflicting information. *Knowledge-Based Systems*, 32:9 – 27, 2012.

M. Jarke and K. Lyytinen. Complexity of systems evolution: Requirements engineering perspective. *ACM Trans. Manage. Inf. Syst.*, 5(3):11:1–11:7, 2015.

B. Jennings, S. van der Meer, S. Balasubramaniam, D. Botvich, M. ó Foghlú, W. Donnelly, and J. Strassner. Towards autonomic management of communications networks. *Communications Magazine, IEEE*, 45(10):112–121, 2007.

S. Jiang, B. Carpenter, and M. Behringer. General gap analysis for autonomic networking. Technical report, Internet Research Task Force, 2015.

A. Kakas and P. Moraitis. Argumentation based decision making for autonomous agents. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '03, pages 883–890, New York, NY, USA, 2003. ACM.

A. C. Kakas, P. Mancarella, and P. M. Dung. The acceptability semantics for logic programs. In *ICLP*, volume 94, pages 504–519, 1994.

M. Kamalrudin and S. Sidek. Automatic acceptance test case generation from essential use cases. In *Intelligent Software Methodologies, Tools, and Techniques, 2014. SOMET 2014. 13th International Conference on*. IOS Press, 2014.

P. Katina, C. Keating, and R. Jaradat. System requirements engineering in complex situations. *Requirements Engineering*, 19(1):45–62, 2014.

J. Kephart, J. Kephart, D. Chess, C. Boutilier, R. Das, J. O. Kephart, and W. E. Walsh. An architectural blueprint for autonomic computing. *IEEE internet computing*, 18(21), 2007.

J. Keppens. On extracting arguments from bayesian network representations of evidential reasoning. In *Proceedings of the 13th International Conference on Artificial Intelligence and Law*, ICAIL '11, pages 141–150, New York, NY, USA, 2011. ACM.

B. Kitchenham and S. Charters. Guidelines for performing Systematic Literature Reviews in Software Engineering. Technical Report EBSE 2007-001, Keele University and Durham University Joint Report, 2007.

U. B. Kjaerulff and A. L. Madsen. *Bayesian Networks and Influence Diagrams.* Information Science and Statistics. Springer New York, 2008.

F. Klügl. Measuring complexity of multi-agent simulations–an attempt using metrics. In *Languages, Methodologies and Development Tools for Multi-Agent Systems*, pages 123–138. Springer, 2008.

J. R. Koiter. *Visualizing inference in Bayesian networks.* PhD thesis, Delft University of Technology, 2006.

G. Koslovski, P. B. Primet, and A. Charao. Vxdl: Virtual resources and interconnection networks description language. In *Networks for Grid Applications*, volume 2 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 138–154. Springer Berlin Heidelberg, 2009.

P. Kraaijeveld, M. Druzdzel, A. Onisko, and H. Wasyluk. Genierate: An interactive generator of diagnostic bayesian network models. In *Proc. 16th Int. Workshop Principles Diagnosis*, pages 175–180, 2005.

O. Landsiedel, K. Wehrle, and S. Götz. Accurate prediction of power consumption in sensor networks. In *Proc. of the Second Workshop on Embedded Networked Sensors*, 2005.

C. Laurent et al. Autonomic network engineering for the self-managing Future Internet (AFI); Generic Autonomic Network Architecture (An Architectural Reference Model for Autonomic Networking, Cognitive Networking and Self-Management), 2013.

P. Leitão, J. Barbosa, and D. Trentesaux. Bio-inspired multi-agent systems for reconfigurable manufacturing systems. *Engineering Applications of Artificial Intelligence*, 25(5):934 – 944, 2012.

I. Letia and A. Groza. Justifying argument and explanation in labelled argumentation. In *IEEE International Conference on Intelligent Computer Communication and Processing (ICCP), 2012*, pages 11–18, 2012.

X. Liu, R. Wanchoo, and R. Arvapally. Intelligent computational argumentation for evaluating performance scores in multi-criteria decision making. In *International Symposium on Collaborative Technologies and Systems (CTS), 2010.*, pages 143–152, 2010.

S. Luke, C. Cioffi-Revilla, L. Panait, K. Sullivan, and G. Balan. MASON: A Multiagent Simulation Environment. *Simulation*, 81(7):517–527, 2005.

H. Luo, S. lin Yang, X. jian Hu, and X. xuan Hu. Agent oriented intelligent fault diagnosis system using evidence theory. *Expert Systems with Applications*, 39(3):2524 – 2531, 2012.

P. Maio, N. Silva, and J. Cardoso. Generating arguments for ontology matching. In *22nd International Workshop on Database and Expert Systems Applications (DEXA), 2011*, pages 239–243, 2011.

P. Maio and N. Silva. A three-layer argumentation framework. In *Theorie and Applications of Formal Argumentation*, pages 163–180. Springer, 2012.

P. Maio and N. Silva. An extensible argument-based ontology matching negotiation approach. *Science of Computer Programming*, 95, Part 1(0):3 – 25, 2014. Special Issue on Systems Development by Means of Semantic Technologies.

A. Marnewick, J.-H. Pretorius, and L. Pretorius. A perspective on human factors contributing to quality requirements: A cross-case analysis. In *Industrial Engineering and Engineering Management (IEEM), 2011 IEEE International Conference on*, pages 389–393, 2011.

G. Marreiros, C. Ramos, and J. Neves. Dealing with emotional factors in agent based ubiquitous group decision. In *Lecture Notes in Computer Science*, pages 41–50, 2005.

A. Martín, C. León, J. Luque, and I. Monedero. A framework for development of integrated intelligent knowledge for management of telecommunication networks. *Expert Systems with Applications*, 39(10):9264 – 9274, 2012.

R. Martin and G. Melnik. Tests and requirements, requirements and tests: A möbius strip. *Software, IEEE*, 25(1):54–59, 2008.

M. L. Massie, B. N. Chun, and D. E. Culler. The ganglia distributed monitoring system: design, implementation, and experience. *Parallel Computing*, 30(7):817–840, 2004.

N. Maudet, S. Parsons, and I. Rahwan. Argumentation in multi-agent systems: Context and recent developments. In *Argumentation in Multi-Agent Systems*, pages 1–16. Springer, 2007.

P. McBurney, R. Van Eijk, S. Parsons, and L. Amgoud. A dialogue game protocol for agent purchase negotiations. *Autonomous Agents and Multi-Agent Systems*, 7(3):235–273, 2003.

J. L. Méndez. Bayesian Reasoning Module for BDI Agent Architectures. Application for diagnosis in FTTH networks. Master's thesis, Escuela Técnica Superior de Ingenieros de Telecomunicación - Universidad Politécnica de Madrid, 2011.

B. Mendoza, P. Xu, and L. Song. A multi-agent model for fault diagnosis in petrochemical plants. In *Sensors Applications Symposium (SAS), 2011 IEEE*, pages 203 –208, 2011.

B. Middleton, M. Shwe, D. Heckerman, M. Henrion, E. Horvitz, H. Lehmann, and G. Cooper. Probabilistic diagnosis using a reformulation of the internist-1/qmr knowledge base. *Medicine*, 30:241–255, 1991.

Mockito Project. Mockito Framework. `http://mockito.org`, 2012. Accessed December 15, 2015.

M. E. Monteiro, R. C. Favoreto, F. B. Silva, P. Negri, and P. P. F. Barcelos. An ontology-based approach to improve snmp support for autonomic management. In *Network and Service Management (CNSM), 2014 10th International Conference on*, pages 280–283. IEEE, 2014.

A. Monteserin and A. Amandi. Argumentation-based negotiation planning for autonomous agents. *Decision Support Systems*, 51(3):532 – 548, 2011.

A. d. Moor and M. Aakhus. Argumentation support: from technologies to tools. *Communications of the ACM*, 49(3):93–98, 2006.

P. Moraitis and N. Spanoudakis. Argumentation-based agent interaction in an ambient-intelligence context. *Intelligent Systems, IEEE*, 22(6):84–93, 2007.

M. Morge and P. Beaune. A negotiation support system based on a multi-agent system: Specificity and preference relations on arguments. In *Proceedings of the 2004 ACM Symposium on Applied Computing*, SAC '04, pages 474–478. ACM, 2004.

R. Mugridge and W. Cunningham. *Fit for developing software: framework for integrated tests*. Prentice Hall, 2005.

P. Müller. Future internet design principles. Technical report, European Commission - Information Society and Media, 2012.

J. D. Myers. The background of internist i and qmr. In *Proceedings of ACM conference on History of medical informatics*, pages 195–197. ACM, 1987.

C. D. Nguyen, A. Perini, and P. Tonella. Goal oriented testing for mass. *Int. J. Agent-Oriented Softw. Eng.*, 4(1):79–109, 2010.

C. D. Nguyen. *Testing Techniques for Software Agents*. PhD thesis, International Doctorate School in Information and Communication Technologies, 2009.

C. Nguyen, A. Perini, C. Bernon, J. Pavón, and J. Thangarajah. Testing in multi-agent systems. In *Agent-Oriented Software Engineering X*, volume 6038 of *Lecture Notes in Computer Science*, pages 180–190. Springer Berlin Heidelberg, 2011.

D. Nguyen, A. Perini, and P. Tonella. A Goal-Oriented Software Testing Methodology. In *Agent-Oriented Software Engineering VIII*, volume 4951 of *Lecture Notes in Computer Science*, pages 58–72. Springer Berlin / Heidelberg, 2008.

M. Nikulin. Hellinger distance. *Encyclopaedia of Mathematics. Kluwer Academic Publishers*, 2002.

D. North. Introducing: Behaviour-driven development. `http://dannorth.net/introducing-bdd`, 2007. Accessed December 15, 2015.

D. North. JBehave. A framework for Behaviour Driven Development (BDD). `http://jbehave.org`, 2011. Accessed December 15, 2015.

O. Ntofon, D. Hunter, and D. Simeonidou. Towards semantic modeling framework for future service oriented networked media infrastructures. In *Computer Science and Electronic Engineering Conference (CEEC), 2012 4th*, pages 200–205, 2012.

N. Obeid and A. Moubaiddin. On the role of dialogue and argumentation in collaborative problem solving. In *9th International Conference on Intelligent Systems Design and Applications (ISDA), 2009.*, pages 1202–1208, 2009.

A. Oniśko, M. J. Druzdzel, and H. Wasyluk. Learning bayesian network parameters from small data sets: application of noisy-or gates. *International Journal of Approximate Reasoning*, 27(2):165 – 182, 2001.

L. Padgham and M. Winikoff. Prometheus: A Methodology for Developing Intelligent Agents. In *Agent-Oriented Software Engineering III*, volume 2585 of *Lecture Notes in Computer Science*, pages 174–185. Springer Berlin Heidelberg, 2003.

R. Pan, Y. Peng, and Z. Ding. Belief update in bayesian networks using uncertain evidence. In *Tools with Artificial Intelligence, 2006. ICTAI '06. 18th IEEE International Conference on*, pages 441–444, 2006.

L. Paolino, H. Paggi, F. Alonso, and G. Lopez. Solving incidents in telecommunications using a multi-agent system. In *Intelligence and Security Informatics (ISI), 2011 IEEE International Conference on*, pages 303 –308, 2011.

K. Pashaei, F. Taghiyareh, and K. Badie. A negotiation-based genetic framework for multi-agent credit assignment. In *Multiagent System Technologies*, volume 8732 of *Lecture Notes in Computer Science*, pages 72–89. Springer International Publishing, 2014.

J. Pavon, J. J. Gomez-Sanz, and R. Fuentes. The INGENIAS methodology and tools. *Agent-oriented methodologies*, 9:236–276, 2005.

J. Pearl. *Bayesian networks: A model of self-activated memory for evidential reasoning.* University of California (Los Angeles). Computer Science Department, 1985.

J. Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference.* Morgan Kaufmann, 1988.

J. Pearl. *Causality: models, reasoning and inference.* Cambridge Univ Press, 2000.

C. Perelman and L. Olbrechts-Tyteca. *The New Rhetoric: A Treatise on Argumentation.* University of Notre Dame Press, 1969.

T. Plevyak and V. Sahin. *Next generation telecommunications networks, services, and management*, volume 15. John Wiley & Sons, 2011.

A. Pras, J. Schönwälder, M. Burgess, O. Festor, G. M. Perez, R. Stadler, and B. Stiller. Key research challenges in network management. *Communications Magazine, IEEE*, 45(10): 104–110, 2007.

G. Provan. Abstraction in belief networks: the role of intermediate states in diagnostic reasoning. In *Proceedings of the Eleventh conference on Uncertainty in artificial intelligence*, pages 464–471. Morgan Kaufmann Publishers Inc., 1995.

I. Rahwan, S. D. Ramchurn, N. R. Jennings, P. McBurney, S. Parsons, and L. Sonenberg. Argumentation-based negotiation. *The Knowledge Engineering Review*, 18:343–375, 2003.

A. Rao. AgentSpeak(L): BDI agents speak out in a logical computable language. In *Agents Breaking Away*, volume 1038 of *Lecture Notes in Computer Science*, pages 42–55. Springer Berlin Heidelberg, 1996.

S. T. Redwine, Jr. and W. E. Riddle. Software technology maturation. In *Proceedings of the 8th International Conference on Software Engineering*, ICSE '85, pages 189–200, Los Alamitos, CA, USA, 1985. IEEE Computer Society Press.

J. Rowe, K. Levitt, S. Parsons, E. Sklar, A. Applebaum, and S. Jalal. Argumentation logic to assist in security administration. In *Proceedings of the 2012 workshop on New security paradigms*, NSPW '12, pages 43–52, New York, NY, USA, 2012. ACM.

J. Rubio-Loyola, A. Astorga, J. Serrat, W. Chai, L. Mamatas, A. Galis, S. Clayman, A. Cheniour, L. Lefevre, O. Mornard, A. Fischer, A. Paler, and H. de Meer. Platforms and software systems for an autonomic internet. In *Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE*, pages 1–6, 2010.

J. Sammon, J.W. A nonlinear mapping for data structure analysis. *IEEE Transactions on Computers*, C-18(5):401 – 409, 1969.

M. Sanz-Bobi, M. Castro, and J. Santos. Idsai: A distributed system for intrusion detection based on intelligent agents. In *Internet Monitoring and Protection (ICIMP), 2010 Fifth International Conference on*, pages 1 –6, 2010.

K. Schwaber and J. Sutherland. Scrum Guide. *Scrum Alliance*, 19(6):21, 2009.

G. Shafer. *A mathematical theory of evidence*. Princeton University Press., 1967.

M. A. Shwe, B. Middleton, D. Heckerman, M. Henrion, E. Horvitz, H. Lehmann, and G. Cooper. Probabilistic diagnosis using a reformulation of the internist-1/qmr knowledge base. *Methods of information in Medicine*, 30(4):241–255, 1991.

C. Sierra, N. Jennings, P. Noriega, and S. Parsons. A framework for argumentation-based negotiation. In *Intelligent Agents IV Agent Theories, Architectures, and Languages*, volume 1365 of *Lecture Notes in Computer Science*, pages 177–192. Springer Berlin Heidelberg, 1998.

S. Skiena. Dijkstra's algorithm. *Implementing Discrete Mathematics: Combinatorics and Graph Theory with Mathematica, Reading, MA: Addison-Wesley*, pages 225–227, 1990.

C. Solis and X. Wang. A study of the characteristics of behaviour driven development. In *Software Engineering and Advanced Applications (SEAA), 2011 37th EUROMICRO Conference on*, pages 383–387, 2011.

H. H. Song, L. Qiu, and Y. Zhang. NetQuest: a flexible framework for large-scale network measurement. *IEEE/ACM Transactions on Networking (TON)*, 17(1):106–119, 2009.

S. Staab and R. Studer. *Handbook on ontologies*. Springer Science & Business Media, 2013.

J. Strassner. DEN-ng: achieving business-driven network management. In *Network Operations and Management Symposium, 2002. NOMS 2002. 2002 IEEE/IFIP*, pages 753–766, 2002.

J. Strassner, N. Agoulmine, and E. Lehtihet. FOCALE: A novel autonomic networking architecture. *International Transactions on Systems Science and Applications*, 3(1):67–79, 2007.

R. Subbaraj and N. Venkatraman. A systematic literature review on ontology based context management system. In *Emerging ICT for Bridging the Future - Proceedings of the 49th Annual Convention of the Computer Society of India CSI Volume 2*, volume 338 of *Advances in Intelligent Systems and Computing*, pages 609–619. Springer International Publishing, 2015.

L. Sun, K. Ousmanou, and M. Cross. An ontological modelling of user requirements for personalised information provision. *Information Systems Frontiers*, 12(3):337–356, 2010.

Y. Tang and S. Parsons. Argumentation-based dialogues for deliberation. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '05, pages 552–559. ACM, 2005.

S. Tannai, Y. Goto, Y. Maruyama, T. Itoya, T. Hagiwara, and H. Sawamura. A versatile argumentation system based on the logic of multiple-valued argumentation. In *11th International Conference on Hybrid Intelligent Systems (HIS), 2011*, pages 370–376, 2011.

X. Tao, Y. Miao, Y. Zhang, and Z. Shen. Collaborative medical diagnosis through fuzzy petri net based agent argumentation. In *Fuzzy Systems (FUZZ-IEEE), 2014 IEEE International Conference on*, pages 1197–1204, 2014.

J. Thangarajah, G. Jayatilleke, and L. Padgham. Scenarios for system requirements traceability and testing. In *The 10th International Conference on Autonomous Agents and Multiagent Systems - Volume 1*, AAMAS '11, pages 285–292, Richland, SC, 2011. International Foundation for Autonomous Agents and Multiagent Systems.

R. Thomopoulos, M. Croitoru, and N. Tamani. Decision support for agri-food chains: A reverse engineering argumentation-based approach. *Ecological Informatics*, 2014.

A. Tiryaki, S. Oztuna, O. Dikenelli, and R. Erdur. SUNIT: A Unit Testing Framework for Test Driven Development of Multi-Agent Systems. In *Agent-Oriented Software Engineering VII*, volume 4405 of *Lecture Notes in Computer Science*, pages 156–173. Springer Berlin Heidelberg, 2007.

S. E. Toulmin. *The uses of argument*. Cambridge University Press, 2003.

C. Tschudin and C. Jelger. An autonomic network architecture research project. *Praxis der Informationsverarbeitung und Kommunikation*, 30(1):26–31, 2007.

G. Tselentis and A. Galis. *Towards the Future Internet: Emerging Trends from European Research*. IOS press, 2010.

T. L. van der Weide, F. Dignum, J.-J. C. Meyer, H. Prakken, and G. A. W. Vreeswijk. Multi-criteria argument selection in persuasion dialogues. In *The 10th International Conference on Autonomous Agents and Multiagent Systems - Volume 3*, AAMAS '11, pages 921–928, Richland, SC, 2011. International Foundation for Autonomous Agents and Multiagent Systems.

F. H. van Eemeren, R. F. Grootendorst, and F. S. Henkemans. Fundamentals of argumentation theory: A handbook of historical backgrounds and contemporary applications, 1996.

N. R. Velaga, N. D. Rotstein, N. Oren, J. D. Nelson, T. J. Norman, and S. Wright. Development of an integrated flexible transport systems platform for rural areas using argumentation theory. *Research in Transportation Business and Management*, 3(0):62 – 70, 2012.

R. M. Vicari, C. D. Flores, A. M. Silvestre, L. J. Seixas, M. Ladeira, and H. Coelho. A multi-agent intelligent environment for medical knowledge. *Artificial Intelligence in Medicine*, 27(3):335 – 366, 2003. Software Agents in Health Care.

M. Voortman. Using Cases To Refine Bayesian Networks. Master of science thesis, Delft University of Technology, 2005.

G. Vreeswijk. Argumentation in bayesian belief networks. In *Argumentation in Multi-Agent Systems*, volume 3366 of *Lecture Notes in Computer Science*, pages 111–129. Springer Berlin Heidelberg, 2005.

K. R. Wallace. The substance of rhetoric: Good reasons. *Quarterly Journal of Speech*, 49 (3):239–249, 1963.

D. Walton. Argumentation theory: A very short introduction. In *Argumentation in Artificial Intelligence*, pages 1–22. Springer US, 2009.

B. Wang and G. Luo. Extend argumentation frameworks based on degree of attack. In *Cognitive Informatics (ICCI), 2010 9th IEEE International Conference on*, pages 771–776, 2010.

G. Wang, T. N. Wong, and X. Wang. A negotiation protocol to support agent argumentation and ontology interoperability in mas-based virtual enterprises. In *7th International Conference on Information Technology: New Generations (ITNG), 2010.*, pages 448–453, 2010.

Y. Wang, L. Yao, B. Liu, and J. Xu. An argumentation based feedback system for action planning of service robots. In *Multisensor Fusion and Information Integration for Intelligent Systems (MFI), 2014 International Conference on*, pages 1–7, 2014.

M. Wardeh, F. Coenen, and T. Bench-Capon. Multi-agent based classification using argumentation from experience. *Autonomous Agents and Multi-Agent Systems*, 25(3):447–474, 2012.

M. J. Wooldridge. *Reasoning about rational agents*. Intelligent robotics and autonomous agents. MIT Press, 2000.

M. Wooldridge. *An introduction to multiagent systems*. John Wiley & Sons, 2009.

M. Wooldridge, N. R. Jennings, and D. Kinny. The Gaia methodology for agent-oriented analysis and design. *Autonomous Agents and Multi-Agent Systems*, 3(3):285–312, 2000.

M. Wynne and A. Hellesy. Cucumber. Behaviour driven development with elegance and joy. `https://cucumber.io/`, 2008. Accessed December 15, 2015.

C. Xiong, W. Xiang, and Y. Ouyang. Argumentation in multi-agent system based on JADE. In *3rd International Conference on Intelligent Control and Information Processing (ICICIP), 2012*, pages 88–91, 2012.

D. Xue-jie, C. Jian, H. Ying-lan, J. Guo-rui, and H. Ti-yun. Multi-attribute negotiation model based on internal factors argumentation. In *Management Science and Engineering (ICMSE), 2013 International Conference on*, pages 20–27, 2013.

Y. Ye, H. Lin, G. Chen, and Z. Liu. Argumentation-based negotiation in multi-agent system of drying oven for automobile body. In *IEEE International Conference on Intelligent Computing and Intelligent Systems (ICIS), 2010.*, volume 2, pages 453–456, 2010.

J. Yuan, A. Bao, L. Yao, X. Qi, and F. Liu. Defeasible logic base bdi agent for argumentation. In *IEEE International Conference on Intelligent Computing and Intelligent Systems, 2009.*, volume 1, pages 223–228, 2009.

J. Yuan, L. Yao, Z. Hao, F. Liu, and T. Yuan. Multi-party dialogue games for distributed argumentation system. In *Proceedings of the 2011 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology - Volume 02*, WI-IAT '11, pages 329–332, Washington, DC, USA, 2011. IEEE Computer Society.

W. Zhang, Y. Liang, S. Ji, and Q. Tian. Argumentation agent based fire emergency rescue project making. In *IEEE Symposium on Robotics and Applications (ISRA), 2012*, pages 892–895, 2012.

Z. Zhang, J. Thangarajah, and L. Padgham. Automated testing for intelligent agent systems. In *Agent-Oriented Software Engineering X*, volume 6038 of *Lecture Notes in Computer Science*, pages 66–79. Springer Berlin / Heidelberg, 2011.

List of Figures

List of Tables

Glossary

AAF Assumption-based Argumentation Framework

ABN Argumentation-based Negotiation

AOSE Agent Oriented Software Engineering

ATDD Acceptance Test Driven Development

AUT Agent Under Test

B2D2 BDI for Bayesian Diagnosis

BARMAS B2D2 ARgumentative Multi-Agent System

BDD Behaviour Driven Development

BDI Belief-Desire-Intention

BEAST BEhavioural Agent Simple Testing

BN Bayesian Network

BPMN Business Process Model and Notation

BRAS Broadband Remote Access Server

CDF Cumulative Distribution Function

CIM Common Information Model

CM Causal Model

CPT Conditional Probability Table

DAF Dung's Argumentation Framework

DAG Directed Acyclic Graph

DEN-ng Directory Enabled Networking-next generation

DPN Distributed Perception Network

DSLAM Digital Subscriber Line Access Multiplexer

ETSI European Telecommunications Standards Institute

ER Error Rate

FTTH Fiber To The Home

GANA Generic Autonomic Network Architecture

HAN Home Area Network

IDCP Inter-Domain Controller Protocol

INDL Infrastructure and Network Description Language

IRTF Internet Research Task Force

KPI Key Performance Indicator

LPwNF Logic Programming without Negation as Failure

MADL Media Applications Description Language

MAPE Monitor-Analyze-Plan-Execute

MAS Multi-Agent System

MPLS Multiprotocol Label Switching

MSBN Multiply Sectioned Bayesian Network

MTTD Mean Time to Diagnose

NML Network Markup Language

OLT Optical Line Termination

ONT Optical Network Terminal

OSPF Open Shortest Path First

OSPF-TE Traffic Engineering Extensions to Open Shortest Path First

OSS Operation Support Systems

OWL Ontology Web Language

P2P Peer-to-Peer

PAF Preference-based Argumentation Framework

PLDM Prior/Likelihood Decomposable Model

PPP Point-to-Point Protocol

PR-OWL Probabilistic OWL

PSM Problem-Solving Method

REN Regional Ethernet Network

RFC Request For Comments

SBE Specification by Example

SDN Software-Defined Networking

SDSL Symmetric Digital Subscriber Line

SLA Service Level Agreement

SM Structural Model

SNMP Simple Network Management Protocol

SPIN SPARQL Inference Notation

SRDL Semantic Resource Description Language

TDD Test Driven Developoment

TLAF Three-Layer Argumentation Framework

TTD Time To Diagnose

UML Unified Modelling Language

VAF Value-based Argumentation Framework

VoD Video On Demand

VPN Virtual Private Network

VXDL Virtual private eXecution infrastructure Description Language

WSN Wireless Sensor Network

APPENDIX A

Publications

The results of this thesis have produced a number scientific publications in journals and in conference proceedings. The list of those publications is shown below:

A.1 Journal Articles

- Álvaro Carrera, Carlos A. Iglesias, Javier García-Algarra, and Dušan Kolařík. A real-life application of multi-agent systems for fault diagnosis in the provision of an internet business service. Journal of Network and Computer Applications, 37:146-154, 2014. ISSN 1084-8045. doi: 10.1016/j.jnca.2012.11.004. **Impact Factor** (2014): 2.229 **Q1**.
- Álvaro Carrera, Carlos A. Iglesias, and Mercedes Garijo. Beast methodology: An agile testing methodology for multi-agent systems based on behaviour driven development. Information Systems Frontiers, 16(2):169-182, 2014. ISSN 1387-3326. doi: 10.1007/s10796-013-9438-5. **Impact Factor** (2014): 1.077 **Q2**.
- Álvaro Carrera and Carlos A. Iglesias. A systematic review of argumentation techniques for multi-agent systems research. Artificial Intelligence Review, 44(4):509-535, 2015b. ISSN 0269-2821. doi: 10.1007/s10462-015-9435-9. **Impact Factor** (2014): 2.111 **Q2**.

- J. García-Algarra, J. González-Ordás, P. Arozarena, R. Afonso, and Á. Carrera. A probabilistic approach to g-pon self healing. Revista Iberoamericana de Automática e Informática Industrial RIAI, 11(1):80-85, 2014. ISSN 1697-7912. doi: 10.1016/ j.riai.2013.11.005. **Impact Factor** (2014): 0.118 **Q4**.

A.2 Conference Proceedings

- Álvaro Carrera and Carlos A. Iglesias. Towards fault diagnosis based on agent technology for wireless sensor networks. In Future Generation Communication Technology (FGCT), 2015 Fourth International Conference on, pages 68-73, July 2015. ISBN 978-1-4799-8266-0. doi: 10.1109/ FGCT. 2015.7300248.

- Álvaro Carrera, Jorge J. Solitario, and Carlos A. Iglesias. Behaviour driven development for multi-agent systems. In Proceedings of The Third International Workshop on Infrastructures and Tools for Multiagent Systems - ITMAS 2012, pages 107-120, June 2012. ISBN 978-84-8363-850-7.

- Álvaro Carrera and Carlos A. Iglesias. Improving diagnosis agents with hybrid hypotheses confirmation reasoning techniques. In Massimo Cossentino, Michael Kaisers, Karl Tuyls, and Gerhard Weiss, editors, Multi-Agent Systems, volume 7541 of Lecture Notes in Computer Science, pages 48-62. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-34798-6. doi: 10.1007/978-3-642-34799-3 4.

- Álvaro Carrera and Carlos A Iglesias. B2DI - A Bayesian BDI Agent Model with Causal Belief Updating based on MSBN. In Proceedings of the 4th International Conference on Agents and Artificial Intelligence, pages 343-346, 2012. ISBN 978-989-8425-95-9. doi: 10.5220/0003746003430346.

- Álvaro Carrera, Javier Gonzalez-Ordás, Javier García-Algarra, Pablo Arozarena, and Mercedes Garijo. A multi-agent system with distributed bayesian reasoning for network fault diagnosis. In Yves Demazeau, Michal Pěchoucěk, JuanM. Corchado, and Javier Bajo Pérez, editors, Advances on Practical Applications of Agents and Multiagent Systems, volume 88 of Advances in Intelligent and Soft Computing, pages 113-118. Springer Berlin Heidelberg, 2011. ISBN 978-3-642-19874-8. doi: 10.1007/978-3-642-19875-5 15.

- Álvaro Carrera and Carlos A. Iglesias. Multi-agent architecture for heterogeneous reasoning under uncertainty combining msbn and ontologies in distributed network diagnosis. In Proceedings of the 2011 IEEE/WIC/ACM International Conferences on

Web Intelligence and Intelligent Agent Technology - Volume 02, WI-IAT2011, pages 159-162, Washington, DC, USA, 2011. IEEE Computer Society. ISBN 978-0-7695-4513-4. doi: 10.1109/WI-IAT. 2011. 106.

- Javier García-Algarra, Pablo Arozarena, Sergio García-Gómez, Álvaro Carrera-Barroso, and Raquel Toribio-Sardón. A lightweight approach to distributed network diagnosis under uncertainty. In Santi Caballé, Fatos Xhafa, and Ajith Abraham, editors, Intelligent Networking, Collaborative Systems and Applications, volume 329 of Studies in Computational Intelligence, pages 95-116. Springer Berlin Heidelberg, 2011. ISBN 978-3-642-16792-8. doi: 10.1007/978-3-642-16793-5 5.

- Pablo Arozarena, Raquel Toribio, and Álvaro Carrera. Distributed fault diagnosis using bayesian reasoning in Magneto. In Reliable Distributed Systems Workshops (SRDSW), 2011 30th IEEE Symposium on, pages 96-101, Oct 2011. ISBN 978-0-7695-4451-9. doi: 10.1109/SRDSW.2011.20.

- Andrés Sedano-Frade, Javier González-Ordás, Pablo Arozarena-Llopis, Sergio García-Gómez, and Álvaro Carrera-Barroso. Distributed Bayesian diagnosis for telecommunication networks. In Yves Demazeau, Frank Dignum, Juan M. Corchado, and Javier Bajo Pérez, editors, Advances in Practical Applications of Agents and Multiagent Systems, volume 70 of Advances in Intelligent and Soft Computing, pages 231-240. Springer Berlin Heidelberg, 2010. ISBN 978-3-642-12383-2. doi: 10.1007/978-3-642-12384-9 28.

APPENDIX B

Developed Tools and Ontologies

A number of different open-source resources have been developed during the course of this thesis. They have been used in the validation of the contributions presented in the thesis and have the aim of fostering further research works.

B.1 Open-source Tools

A set of **open-source tools** are available in public source code repositories hosted by Github web platform. A brief description of these tools are provided below.

BEAST Tool Support tool for the application of the BEAST Methodology.

This tool offers a set of facilities for the testing process of a MAS following the proposed BEAST Methodology. It automates the translation process of both types of stories: User Stories and Agent Stories, and generates test case skeletons for agent unit testing. Moreover, it includes a set of mock agents ready to be used in any test case. It is fully compatible with JADE 4.0 and JADEX 2.0.

Available at: `http://github.com/gsi-upm/BeastTool`.

SHANKS Framework Simulation framework for heterogeneous and autonomous networks based on the MASON framework.

This framework offers a wrapper of the MASON framework focusing on the simulation of MAS in telecommunication networks. It provides a set of capabilities to facilitate the implementation of agents in the simulation framework. Moreover, it offers integration with other agent frameworks, such as JASON framework, or Bayesian inference libraries, such as SMILE or UnBBayes.

Available at: `http://github.com/gsi-upm/Shanks`.

WSN SHANKS Module Extension Module for Wireless Sensor Networks.

This package extends the basic SHANKS framework to simulate a WSN scenario. It has been used to execute a B2D2 Agent used for the evaluation process of the B2D2 Agent Architecture. It emulates MicaZ devices in a motion detection application. It follows the ZigBee standard to build the sink tree topology of the network. The B2D2 Agent in running in the ZigBee Coordination node.

Available at: `http://github.com/gsi-upm/shanks-wsn-module`.

BARMAS Framework Experimentation framework for distributed argumentation process.

Built on the top of SHANKS, this framework is used to validate the B2D2 Argumentative Framework for distributed hypothesis discrimination. It reproduces federated domain conditions to execute argumentative agents. Those conditions are key for the motivational problem, such access restriction for cross-domain information or private background knowledge for every agent.

Available at: `https://github.com/gsi-upm/BARMAS`.

B.2 Ontologies

Moreover, a set of the different **ontologies** have been developed as knowledge models for the proposed Fault Diagnosis Agents. Those ontologies are listed below.

B2D2 Diagnosis Ontology Diagnosis Model used in the B2D2 Agent Architecture.

It is designed to describe the concepts involved in a Fault Diagnosis process, such as *observation*, *symptom* or *hypothesis*. Moreover, it supports the usage of probabilistic concepts

for handling the uncertainty of the diagnosis process.

Available at: `http://www.gsi.dit.upm.es/ontologies/b2d2/diagnosis`.

B2D2 Argumentation Ontology Argumentation Model used in the B2D2 Argumentative Agent Architecture.

It is designed to describe the concepts involved in a Distributed Fault Diagnosis process based on the proposed B2D2 Argumentation Framework. It includes concepts such as *statement*, *argument* or *coalition*. Moreover, it extends the B2D2 Diagnosis Ontology to enable the argumentative capability in a B2D2 Agent.

Available at: `http://www.gsi.dit.upm.es/ontologies/b2d2/argumentation`.

B2D2 WSN Ontology Diagnosis Model of a B2D2 Agent for Fault Diagnosis of a WSN.

This ontology is a extension of the B2D2 Diagnosis Ontology which includes an Structural Model used in the WSN SHANKS Module to validate the B2D2 Agent Architecture. It includes concepts such as *ZigBeeSensorNode*, *ZigBeeBaseStation*, *ZigBeeRouter*, *SensorLink* or *MicaZ* device.

Available at: `http://www.gsi.dit.upm.es/ontologies/b2d2/wsn`.