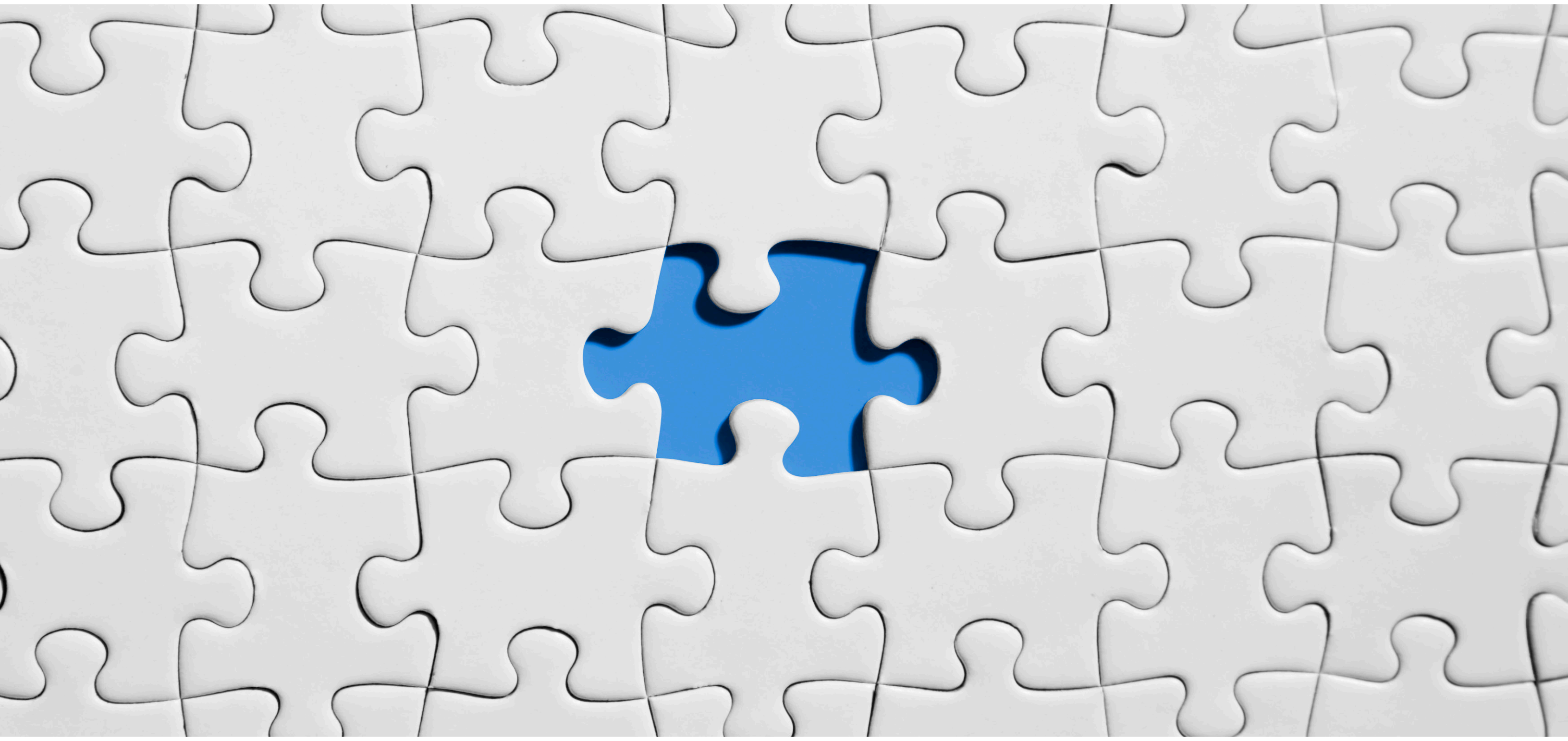
DISTRIBUTED INTELLIGENT SECURITY AGENTS FOR IOT



Sandra Mamdouh Sidhom

Sales Engineer Analyst

Dell Technologies

Sandra.sidhom@dell.com

The Dell Technologies Proven Professional Certification program validates a wide range of skills and competencies across multiple technologies and products.

From Associate, entry-level courses to Expert-level, experience-based exams, all professionals in or looking to begin a career in IT benefit from industry-leading training and certification paths from one of the world's most trusted technology partners.

Proven Professional certifications include:

- Cloud
- Converged/Hyperconverged Infrastructure
- Data Protection
- Data Science
- Networking
- Security
- Servers
- Storage
- Enterprise Architect

Courses are offered to meet different learning styles and schedules, including self-paced On Demand, remote-based Virtual Instructor-Led and in-person Classrooms.

Whether you are an experienced IT professional or just getting started, Dell Technologies Proven Professional certifications are designed to clearly signal proficiency to colleagues and employers.

Learn more at www.dell.com/certification

Table of Contents

1 Introduction

Internet of Things (IoT) is gaining traction as an up to date technology encouraging communication between devices. Data is transferred between devices over a sensor network, which is one of the main elements in the IoT environment. Those sensors are typically placed in open environments, vulnerable to various kinds of threats and attacks launched against the network of devices. Consequently, the network components behave in an unusual manner and manipulated data is exchanged between devices (de Souza et al., 2017). Therefore, attention should be given to detecting abnormal behavior in IoT systems. There are various approaches to deal with this issue. This article examines the more intelligent-based approach.

The development of multi-agent systems is gaining more attention, facilitated by various agent platforms. One widely used agent paradigm is intelligent agents. The strength of this paradigm lies in the abilities of this type of agent, which include mainly reasoning as well as communication capabilities. Their communication ability enables them to be deployed to form a network of distributed systems, in which agents can coordinate with each other. This coordination is supported by their autonomous nature enabling them to observe the surrounding environment and actively react to any occurring changes (Wooldridge, 2009). In order to take a rational decision, after receiving a triggering event, intelligent agents utilize their reasoning power to identify the suitable goal and deliberate over how it will be achieved (Piunti et al., 2008).

This article assesses the capabilities of distributed intelligent agents to detect anomalies in IoT. The main objective is to investigate the existing paradigms of distributed intelligent agents as well as the security threats taking place in the IoT domain, then examine whether it is feasible to utilize the distributed intelligent agents to detect abnormal behaviors in the IoT environment.

2 Anomaly detection

2.1 General overview of security threats

A secure computer system can be defined as a system that functions as intended. In the domain of computer systems' security, the expectation of the actual behavior of the system to be equal to its intended one is called "trust". It influences the confidence in how the system is supposed to function. The system's expected behavior should act as guidelines for the goals that the system should meet. Moreover, a system is considered secure if it conforms to the following concepts: confidentiality, integrity and availability: confidentiality ensures access of data to authorized users only; integrity is realized when information is modified by legitimate entities and it prevents any malicious trials to edit the data; availability guarantees that the required resources are accessible to authorized users anytime (Kumar, 1995).

Having defined the security of a computer system, focus turns to security threats and attacks which face the computing environments. Witzke (2016) pinpoints some of these threats. Password attacks, for example are becoming possible nowadays. Attackers utilize the power of hardware, software and processors to crack passwords and target encrypted password files. Once an attacker can download this file from the victim's system, brute force attacks can be launched on the encrypted data until the correct password of the system is reached and is accessed. In addition, network-based attacks such as Transmission Control Protocol SYN flooding attacks, ping flood attacks, etc. also threaten the security of computer networks. However, they are out of this paper's scope.

Other security threats are viruses and worms. On the one hand a virus is malicious code that harms the system by altering software code. It accesses the system through contaminated files or e-mail attachments. Meanwhile, worms are malicious code that copies itself over the network without modifying files on the system or appending itself to any applications. Nonetheless, they are harmful once the software is executed. Hence, malicious code is viewed as one of the significant threats that should be considered in the security domain (Whitman, 2003). Attacks can also be launched against wireless networks by sending a flood of packets aiming to interfere in communication between networks and affecting the protocol processing capability of the target system. Another type of attack is software engineering attacks which evolve around the idea of fooling the target system to believe that the attacker is a legitimate user facing problems with accessing the system. As a result, and after many trials, the hacker can access the system (Witzke, 2016).

Furthermore, Stallings and Brown (2012) classify security threats based on the CIA security concepts mentioned above. They group security attacks based on their consequences. There are four types of threat consequences and each threatens confidentiality, integrity or availability. As its name indicates, the unauthorized disclosure threatens the confidentiality of the system while deception is a risk against the integrity of the system and its data. Similarly, disruption is a threat to integrity as well as availability. Additionally, usurpation involves misuse and unauthorized control of resources, jeopardizing the integrity of the system.

2.2 Overview of anomaly detection

Many different security technologies are built upon the idea of recognizing threats and potentially malicious activities and preventing or stopping them from harming the system or the network. One of the most common approaches of recognition is signature matching, also called misuse detection. The signature-based detection method is widely used in anti-virus software and intrusion detection systems. It employs a database of recognized attacks by comparing activities to the stored attacks' signatures. If the detector finds an activity like one of the known malicious attacks, it raises an alarm for suspicious behavior. The problem, however, lies in their inability of matching patterns to unknown signatures which were not detected before. Therefore, if an unknown abnormal behavior occurs, this attack will be completely ignored by the detector (Ashfaq et al., 2011).

Anomaly detection technique has been developed to overcome the challenge imposed by signature-based detection approaches on the security of the systems and networks. Applied in several areas including computer network intrusion, gene expression analysis and financial fraud detection, anomaly detection is a promising mechanism to detect unusual behavior in the environment. It is a behavior-based method that looks for normal and abnormal usage patterns. In other words, the anomaly detection system builds patterns of normal activities and continuously observes its characteristics and actual behavior. Unlike the system-based technique, the system gives a warning once an activity which is different than its learnt normal pattern is performed and can detect new types of attacks that did not take place before (Ahmed and Mahmood, 2015).

According to Sekar et al. (2002) the process of anomaly detection comprises two phases: training phase and detection phase. The training phase refers to a learning approach in which patterns are observed and a knowledge base of the normal behavior is created based on machine learning techniques. The detection phase represents the comparison process resulting in detecting malicious activities like those explained above. Anomaly detection aims to recognize anomalous and, especially, intrusive activities. The following part presents anomaly detection techniques from a network intrusion detection perspective.

Anomaly detection techniques are classified into three categories as follows:

1) Statistical-based method:
   It analyzes the behavior of the system over a period and stores the findings in a profile which is afterwards compared to the ongoing activity on the system. The comparison is based on an anomaly score indicating how much the actual event is different than the measures of the stored profile. The score is calculated using a function which considers the measures of the stored profile. Eventually, the anomaly score must be above a specific threshold for the system to consider the processed event as an intrusion (Patcha and Park, 2007).

2) Machine learning techniques:
   They are based on the process of building a data model in order to categorize patterns in a given dataset. This model can enhance its performance by learning from old data patterns. Therefore, the model can apply any required changes when it is provided with new information. Machine learning techniques are classified into several schemes such as Bayesian Networks, Neural Networks, Fuzzy logic, Genetic Algorithms, Clustering and Outlier Detection, which are out of this paper's scope. However, detailed information on their mechanisms and advantages can be found in Garcia-Teodoro et al. (2009); Omar et al. (2013) and Kaur et al. (2013).

3) Knowledge-based anomaly detection technique:
   It gathers knowledge about the occurring attacks against a system or a network. This knowledge is then exploited to raise awareness of and alerts concerning coming attacks. Three more techniques fall under this approach; State Transition Analysis, Expert System and Signature Analysis. Further details are provided about the three techniques by Raut and Singh (2014) and Garcia-Teodoro et al. (2009).

Lazarevic et al. (2003) add another anomaly detection technique that is based on data mining. Techniques for detecting anomalies support building models based on the normal data of the system from a given training dataset. Subsequently, the model finds out if new testing data is considered normal or anomalous. This intrusion detection algorithm is based on supervised anomaly detection, where the model is trained to mark data as normal or abnormal behavior of the network. On the other hand, the unsupervised method does not require data to train from. Nonetheless, both methods result in many false alarms as any unusual action on the system is considered abnormal and a threat. Due to its various techniques for detecting threats and malicious activities, anomaly detection has a lot of interest recently in several domains that will be explored next.

2.3 Applications of Anomaly Detection

This section points at how anomaly detection is applied in each application. Furthermore, it reviews the nature of the data processed and some of the techniques for detecting anomalies related to each domain are highlighted.

2.3.1 Intrusion Detection

The anomaly detection mechanism is very effective when it comes to detecting intrusions. Therefore, an intrusion detection system can be one of the most promising applications for this detection approach. Intrusion detection views behavior of an intruder to differ from that of normal users to a quantifiable extent. Based on the discussion above, anomaly detection is applied in this domain and is considered one of its main aspects according to Lazarevic et al. (2003).

It can be expected that the difference between the behaviors of both parties would overlap. Thus, care needs to be taken when interpreting what defines an intrusive behavior. Too loose of an interpretation of intrusive actions would catch more intruders but would also result in false positives where a legitimate user is mistakenly identified as an intruder. On the other hand, a tight interpretation of intrusive actions would result in false negatives, where intruders are passed off as legitimate users, so there is a certain level of compromise that needs to be considered (Kemmerer and Vigna, 2002; Liao et al., 2013; Stallings and Brown, 2012).

In general, an intrusion detection system continuously examines the events occurring in its surrounding in order to verify if the source of this event is a legitimate action or an attack against the system or the network. In that light, an intrusion detection system is viewed by Debar et al. (1999) as a detector that decides whether the activities taking place in the environment of the system to be secured indicate any intrusive actions. The detector hence takes the following information from the system as Figure 1 shows:

- long-term information of the intrusion detection techniques used by the system such as a database storing records of the previous attacks
- information about the system's configuration and its state
- audit information including the actions that take place on the system

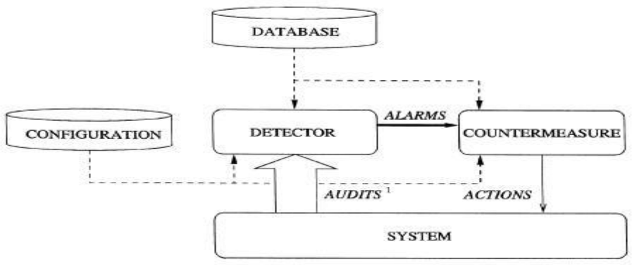


Figure 1: An intrusion-detection system (Debar et al., 1999:806)

The literature examines two main categories of intrusion detection systems; network-based and host-based. They are classified according to their deployment scheme and utilize the anomaly detection techniques discussed in section 2.2. Network-based systems monitor a network segment to capture external attacks against the network. They work on sampling all the packets that pass through the network's traffic in real-time. Host-based systems just monitor a single system while running on its operating system and ensuring its security policy is met. This kind of system looks at logs and activity occurring on the system, for instance the utilization of resources and interaction between programs and tries to find anomalies indicating internal or external suspicious activities (Bhuyan et al., 2014).

2.3.2 Fraud Detection

A further area of interest in which anomaly detection is widely used is fraud detection. It is defined as examining the activities of systems and users in commercial organizations such as credit card or insurance companies to capture anomalous behavior resulting from an illegitimate access or utilization of the organizational resources. Fraudulent actions are detected by capturing anomalies in data (Kou et al., 2004). Today, fraud detection leverages statistical analysis, data mining and machine learning practices. These techniques and how they are applied to detect fraud are out of the scope of this paper yet are explained thoroughly by Bolton and Hand (2002); Phua et al. (2010); Srivastava et al. (2008) and Bolton and Hand (2001).

This anomaly detection approach is based on the concept of keeping records of the normal behavior of the user and checking for abnormalities in the actual behavior. Similarly, this technique is applied by applications of fraud detection including credit card fraud. In the process of detecting fraud in credit card companies, information about the user ID, time between transactions and money spent is recorded while the fraud is an unusual product purchased or high amount spent or a transaction occurring at an abnormal location. That is why techniques for detecting anomalies have two different forms; *a by-owner* approach based on comparing new transactions to usage history of the cardholder and *by-operation*, responsible for finding abnormal patterns related to the geographic location of the transaction (Chandola et al., 2009).

2.3.3 Sensor Networks

Recently, sensor networks have received considerable attention in the computing environment. A sensor network is a deployment of several devices equipped with sensors that perform a collaborative measurement process. The drawback of this kind of network lies in its limited resources, i.e. processor speed, memory, power and radio range. Another issue is the unattended nature of the sensor network that exposes it to multiple security attacks (Bhuse and Gupta, 2006). Also, regarding the network's constraints, Xie et al. (2011) have added its restricted capability of computation and communication. Both resource constraints as well as the unguarded deployment of the sensors make it susceptible to possible attacks and malicious activities (Rajasegarar et al., 2007).

Identification of abnormalities that deviate from the normal pattern in sensed data in sensor networks draws the attention to the need for data analysis. This can be done by analyzing the data monitored and measured by the sensors or the traffic patterns passing through the network (Rajasegarar et al., 2006). The source of abnormal behavior here does not only consist of malicious attacks but can also be noise and errors occurring on the network. In all cases, detecting anomalies ensures the reliability and quality of the measured data and the protected operating of the network (Zhang et al., 2010).

Chandola et al. (2009) divide anomaly detection in sensor networks to two types: sensor fault detection and intrusion detection. It is stated that anomalies in this domain indicate faults in the sensors or intrusive and unusual activities in the network that need to be analyzed further. The data coming from the network and processed by the anomaly detection techniques can be in many forms, for example binary, discrete or continuous. Regardless of the data type they usually include some missing values or noise patterns coming from the sensors' environment. Hence, it becomes a challenging task for the anomaly detection mechanism to identify the actual anomalies and discard this type of unnecessary data.

An added challenge for anomaly detection in the sensor environment is highlighted by Rajasegarar et al. (2008). Limitations and constraints in sensor networks pinpoint the necessity of conserving energy and power of resources. That it is why the adopted approaches to detect anomalies in the network should take this aspect into consideration in order to decrease the energy consumption of the sensors without affecting the lifetime of the network. An example is provided from the machinery sector to explain how anomalies are detected in sensor networks. First, the vibrations of the machines are measured in terms of temperature, pressure, frequency and amplitude. For example, once an unusual value or pattern appears indicating overheating, an alert is given to consider this case.

Sensor networks are a key component in the IoT domain. In an IoT environment, devices transmit huge amount of data from their sensors to enable machine-to-machine communication. Like sensor networks, an IoT network is vulnerable to data tampering and many other threats that will be discussed in the next section.

2.4 Anomalies in IoT

In this section an overview of IoT is given covering its definition, architecture, characteristics, architecture and some of its adopted technologies. It is followed by a discussion of several aspects of the security of IoT as a motivation to the need for anomaly detection to secure the IoT environment. How the anomaly detection approach is managed and implemented in IoT is also discussed.

2.4.1 Overview of IoT

IoT is simply a technology that offers person-to-computer and machine-to-machine communication. It permits devices to communicate by the use of sensors such as Wireless Sensor Network (WSN), nanotechnology and miniaturization in a way that will change our daily routines (Kelly et al., 2013). Moreover, IoT is reshaping our lives by providing opportunities to many best seller applications (Xia et al., 2012; Azzara et al., 2013). Further, IoT is considered the enabler that connected our world (Physical world) with the internet (Cyber Space). It offers interconnectivity and integration between the two mediums. IoT is also viewed as the trend of future networking and it's believed that IoT will cause a third wave IT revolution (Ma, 2011). Furthermore, it is seen that a collaboration between cloud computing and IoT will help address several problems such as transportation issues like congestion and vehicle safety (He et al, 2014). Finally, IoT can be defined as the impact of using technology on the various aspects of our daily lives along with the impact on the technology users' behaviors. This effect will be greatly observable in domestic and working fields (Atzori et al., 2010).

The IoT architecture is divided into four layers as shown in Figure 2. The **perception layer** is composed of the physical components such as data sensors like Radio Frequency Identification (RFIDs), barcodes and any sensor networks of such kind. The main purpose of this layer is to Identify objects uniquely and handling the data collected from the real world via sensors. The **network layer** is responsible for transferring the information collected from the perception layer to any information processing system using communication networks such as Internet or mobile networks. The **middleware layer** contains the information processing systems, where automated actions are taken based on the outcome of the data processed. It acts as a middleware to link the system with the database to store the data gathered. The **application layer** manages the practical applications of IoT to fulfill the needs of the users in different industries, i.e. Smart Home, Smart Environment, Smart Transportation and Smart Hospitals (Farooq et al., 2015).

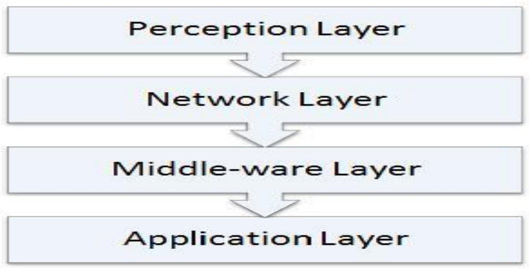


Figure 2: Architecture of IoT (Farooq et al., 2015:1)

Regarding the scope of the paper, prominent technologies used in the IoT environment are RFID and WSN. Both components are part of the perception layer. RFID is based on the idea of attaching tags or smart chips to objects to make them identifiable. The scanning process in the RFID system takes place as follows: the RFID reader sends radio signals to the tag. These signals are received by the antenna and are used to power on the chip. After the chip is powered on the tag it can interrogate with the reader to exchange data that is then sent and saved in a database system linked with the RFID reader. The RFID tags involved in this process can be active or passive. Active tags have an extra internal battery with a finite lifetime from which they get their power and transmit signals through their antenna. Passive tags do not include an internal power source. Rather, they get their power through the signals sent by the interacting RFID reader (Amendola et al., 2014; Gubbi et al., 2013; Kelm et al., 2013).

The transmission of data in an IoT environment can take place over a sensor network, which is a different set of technology that needs to be taken into consideration. WSN, a low cost and easy-to-establish network without any maintenance overhead is utilized in sensing applications. It is defined as several nodes connected which consist of one or more sensors, a processing unit and a transceiver. The sensors enable the collection of data in the form of measured values from the surrounding environment, while the processor handles the intelligent computation of the gathered measurements. The transceiver enables wireless communication between all the nodes over the network which avoids the need to acquire a wired infrastructure. However, the sensor nodes of the WSN are characterized by limited processing capabilities, restricted energy resources and storage as well as short communication ranges and low bandwidth (O'Reilly et al., 2014).

In regard to the characteristics of IoT, Alghuried (2017) stated some distinctive features which take IoT to another level in the Internet world.

- **Ubiquitous sensing**: IoT offers sensing capabilities that not only match human sensing abilities but surpass them. These capabilities are varied and intense enough to merge the gap between human and machine resulting in combining the cyber and physical world to solve human problems. There are two types of sensing objects; an environmental one which is responsible for grasping the surrounding environment and give alarms to humans if hazardous conditions take place; and an emotional sensing object whose main function is to grab the human emotions and give a reaction back to them.
- **Network of networks**: IoT is a heterogeneous network where different types of networks such as GSM, CDMA, WCDMA and IP networks are covered by it.

- **Intelligent processing**: IoT enable humans to focus on qualitative thinking and taking ultimate decisions instead of spending too much time exploring the information. This would likely happen as, while IoT objects are designed to be as intelligent as humans, they are much quicker than humans in processing massive amounts of sensing data.

2.4.2 Security of IoT

Despite the additional aspects introduced by the IoT domain which are not included in general purpose or traditional computing paradigms, IoT devices face similar security problems typically encountered by general-purpose devices. However, IoT systems add additional capabilities which expose them to a new set of threat vectors. The nature of IoT, in terms of unprotected sensors and the need to connect several devices together to transmit a huge amount of data, makes this environment vulnerable to various types of attacks which will be discussed later. In fact, the ultimate security goal of IoT technology is to ensure the basic requirements for data confidentiality, integrity and availability mentioned in a previous section. They form the basis for the security policy of any information system and are also a model for establishing security mechanisms. Farooq et al. (2015) clarify how each concept is guaranteed in IoT security.

2.4.2.1 Security goals of IoT

Data confidentiality addresses how to keep the personal information of users secured against external disclosures or unauthorized access. It can be achieved through data encryption as ciphered text is only accessed by authorization credentials. Another technique is two-step verification which requires two dependent entities to be authenticated in order to provide access to the data. For instance, it ensures that the data of the sensor nodes in an IoT-based network is not disclosed to other nodes and the data collected by RFID tags is only provided to authorized readers. As for data integrity, it prevents tampering of data while they are exchanged between IoT devices so that it reaches the authorized users with its actual form and structure without unpermitted modifications. Integrity is provided by mechanisms for error detection in data such as checksum and cyclic redundancy check. On the other side, data availability uses the redundancy and failover backup methods to guarantee availability and reliability of data to users through duplicating the system components. It also requires firewalls to handle the consequences of a denial-of-service (DoS) attack to make data available to users even during catastrophic situations. So, its main goal is to provide users with access to their data anytime (Farooq et al., 2015).

2.4.2.2 Security requirements of IoT

There are several security requirements, depicted in Figure 3, to consider in order to achieve the previously discussed security goals of IoT. One of the requirements that ensure confidentiality, integrity and availability is user identification. It serves as an identity verification practice for avoiding unauthorized access to IoT devices. From a similar perspective, identity management manages the access of resources inside an IoT system by providing credentials and rights to permitted users when user intervention is necessary. For example, a remotely home monitored health system can be used to handle some health issues of patients avoiding the need of physical treatment. This is a non-traditional medicine approach called “telemedicine” and is enabled by the use of wireless sensor networks, which transmit medical information of patients to the supervisory doctors. Due to the sensitive nature of this information, the principles of user identification and identity management are required in that case in order to identify the users accessing the system as well as secure the transmission of information (Babar et al., 2011; Luo et al., 2010).

However, if the securing process failed and the system is attacked, the tamper resistance mechanisms should defend the IoT system against malicious activities and unauthorized manipulation of data. The remaining concerns address the security of different elements of the IoT system including running applications and the digital content of the system. In addition, the network connection should only be established if the device is secured. More to this point, secure data communication between IoT devices and storage requirements address the confidentiality and integrity of the exchanged data and the users' private information, respectively (Babar et al., 2011; Luo et al., 2010).

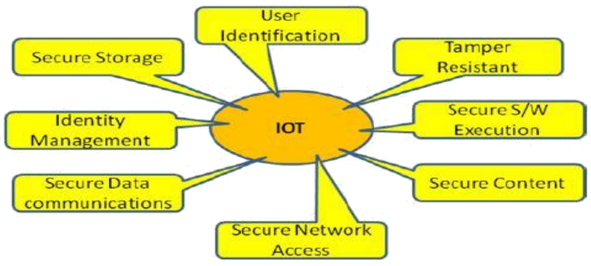


Figure 3: Security requirements for IoT (Babar et al., 2011)

2.4.2.3 Security threats and attacks of IoT

Based on the previous discussion these security requirements should be considered accurately in the IoT to combat the impact of security threats. From the point of view of Zhang et al. (2015), security threats to IoT can be classified to two classes regardless of the type of devices connected over the network. The first class is relevant to the security of the traditional network of non-IoT devices and it threatens its confidentiality, integrity and availability. Nonetheless, the impact and difficulty of these threats in IoT is more intense because of the heterogeneous nature of the devices. Thus, IoT security concerns should address the communication between heterogeneous objects.

These heterogeneous devices are deployed on a communicating network, in which information and communication systems are integrated as an embedded component of the IoT environment. Consequently, large amounts of data need to be collected, stored and transmitted between IoT devices. This challenge is handled by sensors and acting components integrated in the IoT environment, which were not included in the traditional computing systems mentioned above. Sensitive data continuously collected by IoT devices at several locations ranging from room temperature at home to heartbeats require privacy protection as they might be disclosed to malicious attackers, which could lead to invasion of the users' home privacy or health status. Hence, security models must enhance the restricted security capabilities of the resource-constrained IoT devices to manage the collection and disclosure of enormous amounts of critical information to avoid the unauthorized modification of data or the misbehavior of devices (Zhang et al. (2015); Gubbi et al., 2013).

Hodo et al. (2016), claim that IoT is a promising paradigm in several applications such as medical fields, logistics, automobiles and smart cities. Yet there are many threats in IoT that should be considered. One of them is **DoS attack**, a cyber-attack where the perpetrator tries to hinder the availability of the network or system resources to the permitted users by overloading them with useless traffic. Another threat is **malware**. Attackers run disruptive software by exploiting weaknesses of the IoT devices platform aiming to destroy the IoT architecture and accessing confidential data. This kind of attack may lead to data breach. It is another malicious activity harming the IoT environment and users as their personal and confidential information is gathered by intruders. Moreover, the design of the IoT devices connected over the network does not account for the network security mechanisms. Consequently, the IoT environment is difficult to defend against many threats.

In most cases the existence of threats in a network increases the probability of malicious attacks. Babar et al. (2011) study the potential attacks compromising the security of the IoT environment. They are classified into six categories:

1) **Physical attacks** are the most difficult type as they target the destruction of hardware which is an expensive operation.
2) **Software attacks** are the most dangerous as they expose the whole system by making use of its design and implementation weaknesses.
3) **Side channel attacks** which attackers also seek to launch, and in which they collect information generated by encryption devices in order to recognize the encryption or decryption key.
4) **Cryptanalysis attacks** are used to identify the plaintext, if attackers succeeded to get the key.
5) **Network attacks** allow attackers to take advantage of the unprotected nature of the sensor nodes which discloses the data passing through the network to the public. Accordingly, data can be accessed by network intruders.
6) **Environmental attacks** can have an adverse effect on the IoT environment.

2.4.3 Anomaly detection in IoT

Previously, the issues and attacks encountered in the IoT environment were presented in detail. Attention is now turned to the consequences of these threats and the mechanisms used to identify them. Despite the success proved by IoT devices in various applications, the existing threats lead to abnormal behavior of the systems and network. For that reason, the need for stronger resistance in this domain is increased to secure the IoT environment. However, it is not an easy task as IoT devices and sensors operate in an unreliable environment. When sensors transmit tampered data, it is an indication of unusual activity occurring on the network that needs to be analyzed. The abnormal pattern in data is usually introduced in the findings of a data analysis process which is presented in Figure 4 (de Souza et al., 2017).

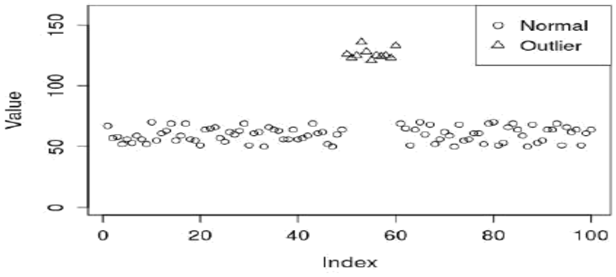


Figure 4: Outliers detected and highlighted in a dataset (de Souza et al., 2017:487)

Referring to Alghuried (2017), the heterogeneity of IoT devices and constraints imposed by the nature of the hardware prevent implementation of host-based security measures. Thus, the need to adopt anomaly detection techniques to secure the IoT devices is emphasized. Nonetheless, security mechanisms require energy and numerous resources especially when they are deployed in devices with restricted computational power, battery capacity and storage, which is the case of IoT devices (Babar et al., 2011). The source of this limitation is the characteristics of the wireless sensor network environment including the inability of sensor nodes to flexibly react to changes. Therefore, anomaly detection approaches implemented in sensor nodes should be lightweight and simple to consume less computational power and energy (O’Reilly et al., 2014).

While the previous part discussed where anomalies can be detected in IoT systems, this subsection focuses on the time and conditions of their detection. Desnitsky et al. (2015) state that the deviations in the data, produced by the sensors resulting from malicious actions, are detected while the system is running. Detection is performed according to a group of predefined criteria and restrictions – referred to as, expert knowledge. Expert knowledge applies specific constraints during the system's running time, considering the previous readings of the sensors, the current state of the data in terms of time, source, type, destination and location as well as the structure and the operating logic of the monitored system.

This thesis tries to contribute to the security of IoT by developing an approach to capture anomalies in an IoT network. The following chapter evaluates the capabilities and draws on the strengths of the intelligent agents to that problem domain.

3 Distributed Systems

3.1 Challenges of Distributed Systems

A distributed system is defined as a system where communication takes place between the components of hardware or software found in a network of computers. This communication is organized through sending and receiving messages (Coulouris et al., 2012). Another representation for a distributed system is given by Tanenbaum and Steen (2007). They view it as a group of separate components or computers, but users see it as one consistent system. This definition indicates that those independent components must function together. While this collaboration depends on the development of the distributed system itself, it is kept out of the users' sight.

According to Braubach and Pokahr (2012), developing distributed applications is becoming an ongoing necessity. It resulted from using smart devices extensively and trying to connect them over several networks. However, developers encounter many challenges when designing and implementing these systems. Therefore, they need to decide on a bundle of technologies and technical approaches to be used in order to handle and solve these challenges. In addition to adopting the applicable technical solution, they need to consider a software paradigm for it, which describes the software from a real-world perspective by identifying the components of the software and how they communicate.

Before exploring some of the existing systems paradigms, several challenges encountered during their development and faced by software engineers will be discussed. Unlike distributed systems, developing an individual application requires focusing only on its behavioral aspect. It is concerned with developing the software in a way that ensures the intended design, flexibility and sustainability. On the other hand, developing distributed systems causes many issues illustrated in Figure 5 such as concurrency and distribution as well as non-functional problems. Pokahr et al. (2010) view concurrency to provide systems with more computational power. For example, it takes place through using multi-core processors and other facilities to enable parallel processing.

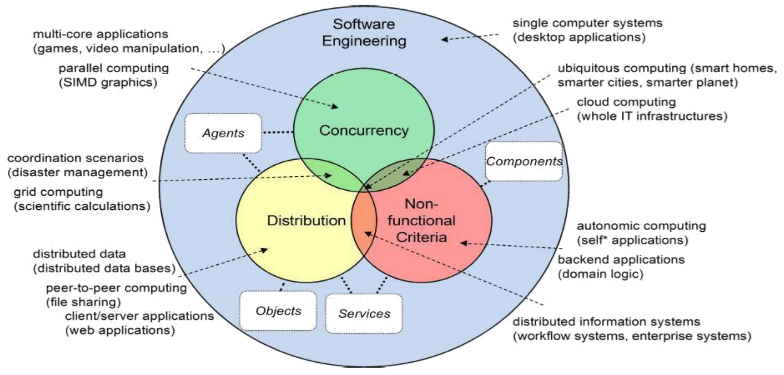

Software Engineering
multi-core applications (games, video manipulation, ...)
parallel computing (SIMD graphics)
single computer systems (desktop applications)
ubiquitous computing (smart homes, smarter cities, smarter planet)
cloud computing (whole IT infrastructures)
Agents
Concurrency
Components
coordination scenarios (disaster management)
grid computing (scientific calculations)
Distribution
Non-functional Criteria
autonomic computing (self* applications)
backend applications (domain logic)
distributed data (distributed data bases)
peer-to-peer computing (file sharing)
client/server applications (web applications)
Objects
Services
distributed information systems (workflow systems, enterprise systems)

Figure 5: Applications and paradigms for distributed systems (Braubach and Pokahr, 2012:101)

However, it has many problems such as divergence, deadlock and mutual exclusion. Divergence is defined as a loop that never ends. Concurrency enables several executable threats to run at the same time concurrently, while deadlock is defined as a state, in which a system waits as its required resource is occupied by another system. Therefore, each process is only resumed when the resource it needs becomes accessible and not used by another process. Similarly, mutual exclusion occurs once two processes are not allowed to exploit the same resource at the same time (Agha, 1985).

Furthermore, distribution is divided into two aspects: inherent distribution and distribution transparency. Inherent distribution means that all the entities of the system such as users, other systems and information are naturally spread. In other words, diverse users generate and organize various kinds of information which are managed independently by different programs. Consequently, this distribution must be transparent. People and other applications should see the system as one independent entity regardless of the underlying communication processes and resources of its components which are physically distributed (Nadiminti et al., 2006).

Coulouris et al. (2012) have discussed the non-functional issues of distributed systems mentioned above. It mainly revolves around the quality of the service. Quality is mostly influenced by the reliability, security and performance of the system. They determine whether the system can deliver services timely and as intended. Another non-functional problem for developing distributed systems is scalability. A system can be scalable in three different ways. It can either: increase the number of its users and resources; be a geographical scalable system in which users and resources are located at different places; ensure scalability on an administrative level by operating across several separate departments or entities (Tanenbaum and Steen, 2007).

From a similar perspective, a distributed system should be scalable and capable of coping with many resources and users simultaneously to enhance performance and availability. Fault tolerance is mentioned as an additional non-functional issue of distributed systems. The wide-spread use of distributed systems and parallel processing may result in network and hardware failures that cause fault delays in communication and computation processes (Mishra and Tripathi, 2014).

Also, openness and performance improvement are important aspects that should be considered. Improving performance of the systems as well as incurring low operating costs is a very challenging task. While openness of a distributed system identifies whether the system can be further developed and implemented, the extent to which a system can add and share more services with other applications indicates its level of openness. It is only enabled by publishing the interfaces to make the shared resources accessible and through agreeing on a communication technique for all systems (Mishra and Tripathi, 2014). The next section explores some solutions used for developing distributed systems that address these challenges.

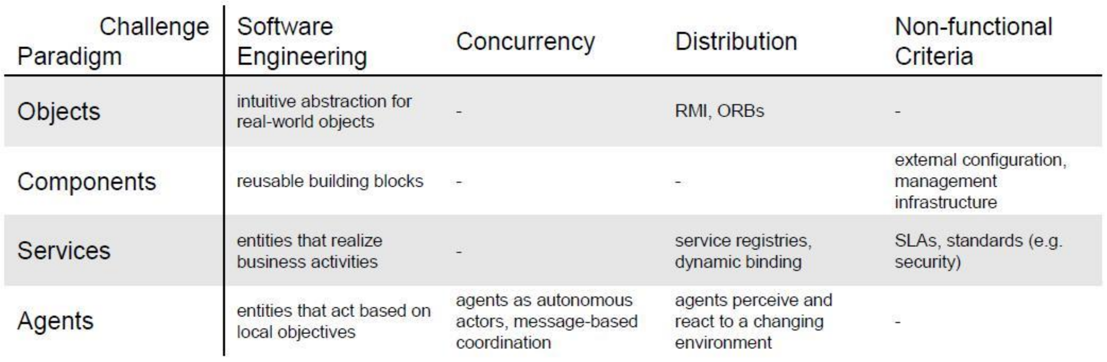
3.2 Distributed systems paradigms

This section discusses several software paradigms with their limitations to build distributed applications, focusing on the concept of intelligent agents, their characteristics and why they are useful for developing distributed systems. Also explained is how the agents' paradigm is combined with other technologies and concepts to form a new software paradigm, "Jadex active components", which contributes to the challenges mentioned in section 3.1.

3.2.1 Review of different paradigms

Braubach and Pokahr (2012) suggest paradigms for developing distributed applications. These include object, component, service and agent orientation. Each paradigm can tackle one or two challenges mentioned in the previous section as shown in Table 1. Object Orientation is used to build applications simulating the actual world. It includes Remote Method Invocation to coordinate with distributed systems. The component orientation is an extension for the object-oriented model. They are independent entities that supply a well-defined functionality and also manage non-functional features. Another paradigm is service-oriented. It tends to combine the business and technical side by using services to perform tasks required for the business process.

Agents are self-managing and independent programs that can operate with multiple systems and behave in a smart manner.



Challenge / Paradigm	Software Engineering	Concurrency	Distribution	Non-functional Criteria
Objects	intuitive abstraction for real-world objects	-	RMI, ORBs	-
Components	reusable building blocks	-	-	external configuration, management infrastructure
Services	entities that realize business activities	-	service registries, dynamic binding	SLAs, standards (e.g. security)
Agents	entities that act based on local objectives	agents as autonomous actors, message-based coordination	agents perceive and react to a changing environment	-

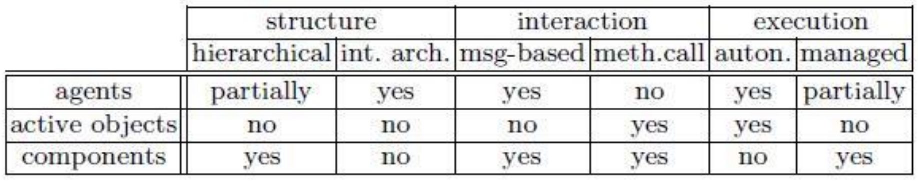
Table 1: Contributions of paradigms (Braubach and Pokahr, 2012:102)

As seen in Table 1, the four paradigms are not able to handle concurrency, distribution and non-functional problems together. Nonetheless, agents are able to tackle the two main functional challenges, indicating that the agent paradigm is very promising for developing distributed systems. This thesis is built around the idea of “BDI agents”, which consist out of an underlying intelligent distributed agents’ network. The agent-based software paradigm and the idea of intelligent agents followed by the “BDI” model will be thoroughly explained in section 3.2.4.

3.2.2 Requirements of software paradigms

Pokahr et al. (2010) cover from a different standpoint some important requirements of the software paradigm in order to create distributed systems. They should comply with the software engineering principles such as composition or decomposition and reusability. A clear behavior should be demonstrated by the software components. The paradigm must provide an interaction and communication functionality, for example, through sending and receiving messages or exploiting services. This interaction as well as all the entities’ actions should be done independently. In addition, it should manage non-functional challenges such as scalability.

Each requirement of the software paradigm is matched from a technical point of view to a specific characteristic listed in Figure 6 including structure, interaction and execution. The first criteria related to the software engineering principles is targeted by the structure property as it covers the mechanism inside the component itself and how it is constructed, whether it is composed of or decomposed into several entities. The structure also concentrates on the second requirement about the behavior of the software entity. From its name, the third criteria are addressed by the interaction property communicating through messages and invoking methods. As for the execution feature, it focuses on the fourth and fifth requirement. They state how the components are implemented and run in their environment. The runtime phase of the components must be done independently since an existing infrastructure must have control on the components in order to comply with the main non-functional aspects.



	structure		interaction		execution	
	hierarchical	int. arch.	msg-based	meth.call	auton.	managed
agents	partially	yes	yes	no	yes	partially
active objects	no	no	no	yes	yes	no
components	yes	no	yes	yes	no	yes

Figure 6: Technical properties of paradigm entities (Pokahr et al., 2010:104)

3.2.3 Evaluation of software paradigms

The following software paradigms including active objects and components will be evaluated based on the requirements and technical properties presented in section 3.2.2. The third paradigm – the software agents' paradigm – will be discussed in detail in section 3.2.4.

3.2.3.1 Active objects paradigm

An active object is a mechanism for developing concurrent systems. It integrates object-oriented programming principles using its own methods, concepts and one individual active thread. This thread for each object decreases the complicated concurrency and distribution issue (Clarke et al., 2008). It can reduce the coupling between the method caller and callee through asynchronous communication. The method calls are managed by a scheduler over each object's thread. Therefore, it can achieve more concurrency in a distributed environment (Johnsen and Owe, 2007). These functionalities show that the active object can address the interaction and part of the execution property through managing the method call process in an independent way by decoupling the caller and callee. Nevertheless, it has limitations regarding the execution and structure of the object, so it is not considered a complete paradigm for distributed applications.

3.2.3.2 Component-based paradigm

The second paradigm is component-based software. It is created by combining several subcomponents following specific component architecture. Each component has an interface that serves communicating with other components (Cai et al., 2000). It is defined by Washizaki et al. (2003) as an entity composed of many others with interfaces. Interfaces enable users to take advantage of the software without understanding its technical implementation. The interaction is enabled via messages and method calls. Regarding communication between the user and the component, it only exists when the user requires a specific task from the component. Thus, they are passive entities that do not function independently. The component software includes a management infrastructure that aids in its implementation process in order to build flexible and scalable executable applications. Like the active objects, the component-based concept does not concentrate on how the component should be internally structured but unlike the objects paradigm it is considered with the composition of the entities.

3.2.4 Discussion of intelligent agents' paradigm

This section discusses the intelligent agents' paradigms in detail. It starts by explaining the aspects of an agent system. The "Jadex Active Components" is presented as a framework for developing distributed agents. Intelligent agents, their characteristics and other relevant concepts are examined. Finally, the BDI model is presented as a model for developing intelligent agents.

3.2.4.1 Conceptual idea of agent systems

Another successful paradigm for distributed computing is agent systems. It also represents the concurrency aspect in its model. Agents are situated and reactive meaning they observe the environment around them and react to any changes. In addition, they are proactive and independent as each agent is able to manage its own behavior and goals following its intentions and not according to human actions, as opposed to component-based software. Consequently, they are eligible to run in a distributed concurrent environment. They are also socially capable of interacting with other agents (Yu, 2001). Being autonomous and interactive allows an agent to coordinate with other agents and form a multi-agent system. They act using messages on behalf of the users to simulate their ordinary communication process. Figure 7 depicts the interaction between an agent and its environment as an on-going process. Referring to the independency of systems, an object can manage its state and methods and identify who can access them. However, it is not able to control neither its behavior nor its execution after they are invoked. On the contrary, the interaction between agents in a multi-agent system is performed differently. It is not based on method calls but is represented in form of providing and requesting services. It gives agents the chance to decide on providing the service or not so they have a full control over the behavior of the system (Wooldridge, 2009).

According to the previous evaluation and Table 1 findings, the discussed software paradigms are very powerful, but they still have weaknesses that should be considered. For example, none can handle concurrency, distribution and non-functional problems together. This hinders the possibility of creating applications that overlap all or just two of the areas (cf. Figure 5). As a result, the properties and capabilities of the paradigms were consolidated to a new agents' software paradigm – "Jadex Active Components".

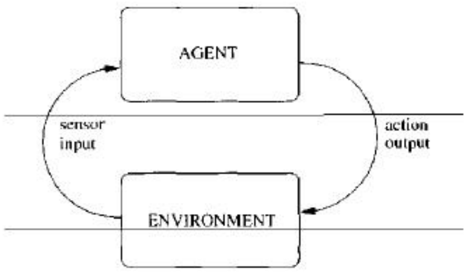


Figure 7: An agent in its environment (Wooldridge, 2009:16)

3.2.4.2 Jadex Active Components

Building distributed systems is not a simple process. Many complexities such as heterogeneous software and hardware as well as communication difficulties should be considered. The existing software paradigms can address some of these issues, yet they still have weaknesses. "Jadex Active Components" is an approach that combines the properties of the component's software, the active objects and the service-oriented paradigm with agents to strengthen the mechanism of building distributed applications (Jander et al., 2015).

Jadex establishes a new paradigm for distributed systems. The agent's platform used in the Jadex environment is organized as follows: the programming language used is Java complemented with a specific programming model and framework provided as an execution infrastructure. The programming model is an addition to an architecture called Software Component Architecture (SCA). SCA components consolidate Service Oriented Architecture (SOA) with an object-oriented paradigm. These components interact through services. The environment is rapidly changing and therefore needs technologies that can cope with it. Therefore, the active components have been added to the SCA in order to change them from passive to independent active entities, which is beneficial for them.

In this context the active component is an independent agent. It interacts with its surroundings via sensors and reacts based on its observations as depicted in Figure 8. On the other side, the SCA component is passive. It simply provides to and makes use of services from other components. Moreover, SCA components can be decomposed and linked together through required and provided services. Thus, they support composition and service-oriented communication of the active components (Braubach and Pokahr , 2012).

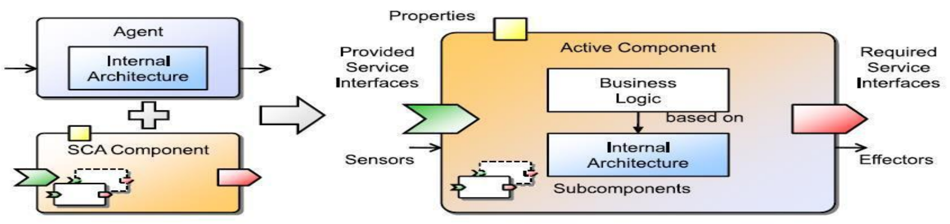


Figure 8: Active component structure (Braubach and Pokahr , 2012:103)

After presenting the structure of the active component, its behavior will be introduced. The behavior of the active component is composed of the mechanism of the component and the service. Additionally, this service has an interface and implementation. The component behaves according to its internal or underlying architecture. In other words, it either reacts to a specific event or takes the initiative to be proactive and perform an intended task (Pokahr et al., 2013a).

Active components function independently as agents without human intervention and hence can reject providing a specific service, unlike objects. Despite that, active components are usually programmed to offer the required service. However, if, for instance, another service request exists then the execution will be terminated, and the active component will decide on the action that should be performed. This process demands asynchronous service interfaces controlled with futures. The future is defined as a placeholder that the provider sends to the requester as soon as it gets the service call. The provider also places the value of the result in it when the execution is completed. During the execution the caller can function normally. Afterwards, it is informed of the available result through a callback (Jander et al., 2015).

The Jadex paradigm requires an execution infrastructure in addition to the programming model. Braubach and Pokahr (2012) present Jadex platform as the infrastructure for loading and executing active components. Furthermore, it enables components to interact and discover other components existing in remote platforms. These functionalities are facilitated through the component container. It includes the basic modules of the platform, namely the component management and messaging module.

The component management serves for starting and stopping the active components. After it creates an instance of the component, it allows its provided services to be found and invoked. On the other side, the way for publishing the required services depends on how the component is configured. As for the messaging module, it identifies each component with an ID to make it able to interact by sending and receiving messages. All the available modules are viewed as the provided services of Jadex platform. Hence, it is treated as an active component and has all its functionalities. This means that the platform’s services are also controlled by asynchronous call processing which addresses the concurrency problem.

Referring to Braubach and Pokahr (2012) and as far as this paper is concerned the implementation of Jadex platform targets developing distributed systems. This requires communication between components dispersed among remote platforms. The awareness approach provided by Jadex platform supports this aspect and works as follows. It provides different mechanisms to discover remote platforms yet all of them are based on the same idea. Figure 9 shows that once a remote platform is detected, a proxy component is created for it on the local platform by the awareness management module. The proxy component stands for its corresponding remote platform. If the scope of the required service contains remote platforms, the proxy components contact a remote management system to start searching for the required service on its remote platforms.

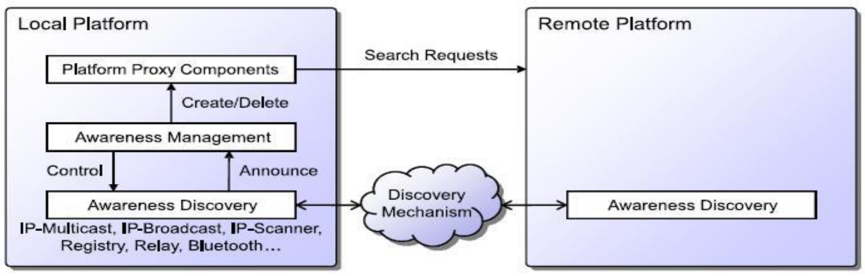
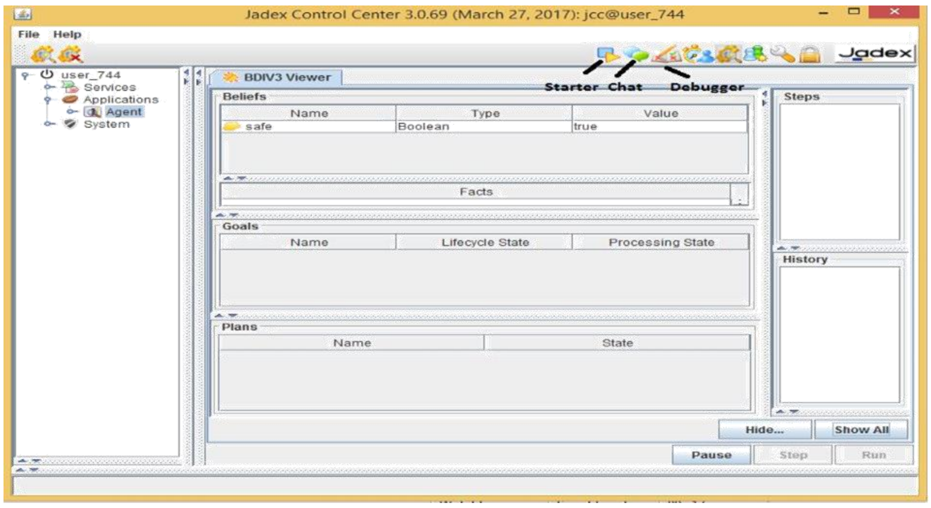


Figure 9: Jadex platform awareness (Braubach and Pokahr, 2012:112)

An additional property provided by Jadex is a bundle of tools composed of two types: tools for modeling and runtime tools. Writing the code of the agents is done using either a Java or an XML development environment. The Jadex Control Center (JCC) brings together Jadex runtime tools shown in Figure 10. They control the components through a running infrastructure. Each tool has its corresponding plug-in. The *Starter* plug-in lists the existing components and controls their status and execution, while the *Chat* plug-in is responsible for the messaging process and the interaction between components. The *Debugger* aids the developer in observing the execution process step by step and inserting breakpoints to identify errors (Pokahr et al., 2010)



Name	Type	Value
safe	Boolean	true

Name	Lifecycle State	Processing State

Name	State

Figure 10: Runtime tools

Having discussed Jadex as a framework for creating agents, which provides them with a communication platform, the next section will discuss building intelligence into the agents. These agents have mental states as they are based on a model called “Belief-Desire-Intention”. This model supports the human behavior including the humans’ desires and motivations. A review on the BDI approach is given in section 3.2.4.4.

3.2.4.3 Intelligent agents

Based on this analysis the agent’s paradigm has proved its potential in distributed networks. Experts claim that its interactivity, self-government and motivation to accomplish its goals are a necessity to software developers (Luck, 2005). It is essential therefore to look at it in detail.

As computer systems interact independently over many types of networks, they have required new development technologies and techniques that cope with these changes. Consequently, software agents, defined as small and independent programs, have been suggested as new development paradigms for distributed systems. Additionally, they are easy to handle and use in building complicated systems (Minar et al., 1999). In distributed systems they serve as a model for human behavior. They act as autonomous, proactive and reactive programs (Howden et al., 2001).

Intelligent agents can take rational decisions by behaving proactively as well as reactively at the same time. A proactive agent chooses a specific goal and insists on achieving it by selecting the suitable required actions. Meanwhile, behavior of the reactive agent is altered based on any occurring changes, which indicates its flexibility. Intelligent agents can commit to their decisions and show flexibility when changes take place in their surrounding environment. However, when taking a decision, intelligent agents must balance between the reactive and proactive principle (Winikoff et al., 2001).

According to Martin et al. (2016), the advent of intelligent agents facilitates the management of rapidly changing environments by providing intelligent interaction between agents. Intelligent interacting agents can serve the users' needs through simulating the behavior carried out by humans in the real world. From the same perspective, intelligent agents are not only able to interact with each other but also with the users in order to accomplish their predefined goals.

Due to the agents' capability of modeling human behavior, they can communicate with users and accomplish their goals. Users seek support services that match their needs and preferences, are flexible and can interact. They need an automated reasoning technology. However, these requirements are difficult to provide due to the rapidly changing, complicated environment and time limitations. Therefore, developers aim to combine the human behavior concepts with the automated reasoning techniques to build intelligent agents' systems (Nguyen and Jain, 2009).

These systems can achieve the users' goals in an interactive, effective and efficient way. The new system paradigms not only perform computational tasks, but also enable goals delegation and interactive support in a conversational manner. This capability proves that these systems can handle the users' tasks and do not require human intervention. They pay attention to their environment instead and react if anything regarding their goal's changes (Nguyen and Jain, 2009).

After discussing the ability of the intelligent agents to interact with users and accomplish their goals, attention should be drawn to the development of agent-based systems. To build an application following this paradigm, one needs an agent's platform as mentioned in section 3.2.4.2 to support this. Referring to Winikoff (2005), it is a technology used to build agent systems, divided into two components; agent-oriented programming language and a framework. The programming language provides the concepts and syntax for creating the agents while the framework is necessary for allowing communication between agents through sending and receiving messages.

An agent platform should comply with the following features. First, it must be familiar to developers. It must be similar or an addition to the already existing languages in order to be easily understood and used. It must enable the agents' integration and interaction with other software as well as with other services, databases and graphical user interfaces. The platform should be scalable and strong to support large systems and any number of agents reliably and efficiently. Most importantly, the platform should include development and debugging tools and must be supported by documents.

3.2.4.4 BDI Agents

Several platforms have been introduced in the past years to develop multi-agent systems. However, they are not capable of covering the three main requirements of an agent's platform;

openness, middleware and reasoning capabilities. Openness is concerned with linking heterogeneous systems over the network. Middleware addresses interoperability and services infrastructure. Reasoning capabilities support the decision capabilities of agents through a goal-oriented approach. The difference between the two previous properties is a motivation for adopting the BDI model (Pokahr et al., 2005). This section starts with the concept of BDI and a description for the BDI foundation. Then it presents the BDI architecture, design and implementation. Finally, the execution model of the BDI reasoning engine is briefly explained.

Jadex programming model is viewed from the inter-agent and intra-agent levels. The inter-agent level handles the communication between agents, while the intra-agent level is concerned with the internal architecture and implementation of the individual agent. This concept is targeted by the BDI reasoning engine supported by the goal deliberation and means-end reasoning. The BDI is a model for developing intelligent agents and hence, is based on human behavior and actions backed up by philosophical and psychological theories. It analyzes human behavior as an intentional stance by introducing the system design as a group of components intending to attain specific goals by applying how people think (Pokahr et al., 2013b).

Through following the motivational or intentional stance, BDI agents assume that human intentions and desires are the only reason or explanation for human attitudes and actions. This type of agent behaves independently according to its desired goal by examining its environment via sensors and acting in response to any changes (Castanedo et al., 2006). The BDI model concludes these actions rationally from three mental attitudes: beliefs, desires and intentions. The agent's decision of acting is facilitated by a practical reasoning mechanism. First, the agent selects several goals it aims to accomplish through a goal deliberation process. Then the means-end reasoning deals with how they will be attained (Piunti et al., 2008).

BDI agents are defined by their *beliefs*, *goals* and *plans*. In the following representation, design and implementation of the three concepts will be presented. According to Braubach et al. (2004) *beliefs* are informational attitudes. They consist of information about the agent's environment and its own status. Beliefs stand for the agent's perception of the real world. The perceptual concept includes only facts placed in a belief base that make up the agent's knowledge. It serves as a source for plans and goals to retrieve valuable information about the agent. This information is in the form of a single Java object or a set of objects (Korecko, 2014).

The belief base of the agent is object-oriented to enable flexibility of manipulating classes. The beliefs are accessed with their names in order to be able to change, remove or add them. The belief base is an active storage because it serves as conditions for executing plans or creating goals. Hence, during the execution the reasoning engine takes them as an input and based on their changes or satisfaction, the agent may generate an internal event that can terminate a plan, drop or create a new goal (Braubach et al., 2005).

Pokahr et al. (2005) gives an overview about *goals*, the second mental attitude of the BDI agent. Goals are the motivational attitudes of the agent that represent its desires. Agents always seek to perform actions until the *goal* is achieved. If it is not the case, the adopted goal should be discarded. Like *beliefs*, *goals* are stored in a goal base to be reachable for the reasoning entities of the agent and for the plans to be aware of the agent's desires. The agent's *goals* are managed by a goal life cycle depicted in Figure 11, responsible for monitoring the status of the goal whether it is an option, active or suspended. The process runs as follows: once the *goal* is adopted, it is included as an option in the goal base. Afterwards, the settings for the *goal* deliberation determine its current state. The *beliefs* of the agent identify the validity of the *goal* by a context condition. Therefore, the goal execution is paused if it is invalid.

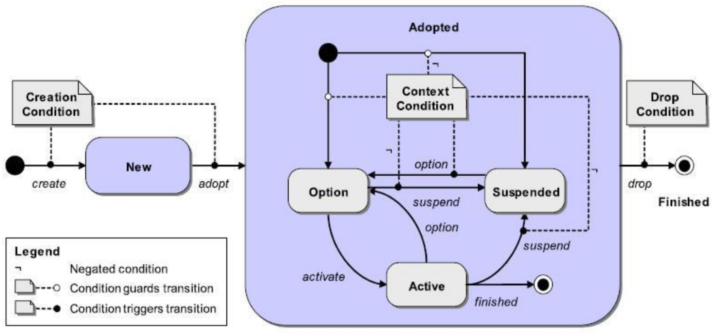


Figure 11: Goal lifecycle (Pokahr et al., 2005:5)

There are four types of *goals*: achieve, perform, maintain and query *goal*. The achieve *goal* is a desired world state that is considered reached when a specific condition is met without considering the execution of plans. The perform *goal* is accomplished when a specific plan is done regardless of its output. A maintain *goal* is similar to an achieve *goal* but it ensures preserving the achieved state by continuously executing plans. Hence, it is never dropped until it is forced by the programmer himself or by a condition. The query *goal's* desired target is related to the internal status of the agent. So, like the achieve goal it succeeds when the target is presented either through a plan or not (Pokahr et al., 2013b).

Another BDI concept is *plans,* which stand for the agent's deliberative or behavioral attitudes. They are ways to attain *goals* and take actions when changes occur such as receiving a message or adopting a new goal. The conditions for selecting a *plan* are included in the *plan* head. They set the criteria for executing the *plan* and identify the events and *goals* a *plan* handles, while the body of the *plan* includes the step by step actions performed by the *plan*. An agent has several distinct *plans* implemented as Java classes and can be executed by other agents. Agents intend to select and execute *plans* to accomplish goals or sub-goals which create a hierarchy of *plans. Plans* are chosen during a meta-level reasoning deliberation process in which the desired *goals* and their corresponding suitable *plans* are identified. This process is repeated until the *plan* is executed correctly without delays or errors (Braubach et al., 2005).

The deliberation process mentioned previously has several inputs. These inputs embrace the messages coming into and out of the agent, its internal events and the new adopted goals. They influence its reaction to any internal changes that occur. This deliberation procedure determines whether an event will be added as a new *plan* to the existing *plan* library or to the active *plans*. It runs like a cycle as running a *plan* as shown in Figure 12 can alter the belief base, make the agent interact with other agents via messages or also result in building new goals and internal events. The agent is mostly viewed as a black box with it inputs and outputs. Nonetheless, its properties should be identified to the system in order to construct an agent and start it. First, the agent's *beliefs*, *goals* and *plans* make up its state. The three concepts are mainly implemented using Java classes which encompass different Jadex libraries and BDI features. Therefore, only the BDI fundamentals of each agent impact its behavior while its reaction and deliberation technique are not different among the other agents (Braubach et al., 2003).

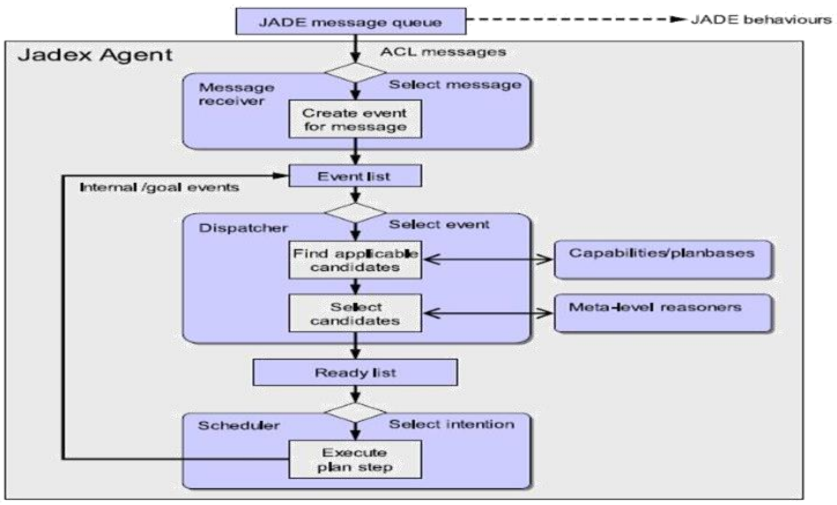


Figure 12: BDI agent architecture (Braubach et al., 2003:78)

After reviewing the BDI architecture, in the following part the different components used in the BDI execution model will be explained. Piunti et al. (2008), claim that the model differentiates between processing a *goal* and executing a *plan*. Goals stand for the world desires that should be accomplished. *Plans* describe the actions for achieving them.

The three components involved in the process are message receiver, dispatcher and scheduler depicted in Figure 13. They all aid the *plan* selection as it is considered an essential step in the reasoning process. Each component has a separate functionality. The message receiver gets new messages, considers them as new events and adds them to the event list. The dispatcher takes the event list as an input and selects the possible *plan* for the events. Afterwards, it chooses the suitable *plans* for execution, matches each *plan* to an event or *goal* and adds it to the ready list. Eventually, the scheduler performs the execution of the *plans* (Braubach et al., 2004)

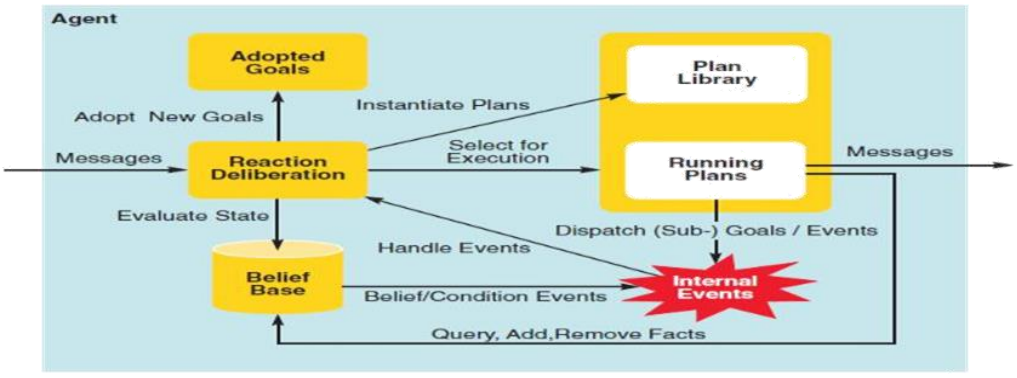


Figure 13: BDI execution model (Braubach et al., 2004:5)

Based on this discussion and from a developer point of view, the BDI model is very simple to understand yet it provides all the functionalities needed by software engineers to develop an agent-based system using the idea of objects. First, the programming model is deploying Java as a programming language which is a familiar and easy language. Moreover, a developer can use any development environment to create BDI agents, another advantage of the model, in addition to the list of functionalities regarding developing distributed agents capable of communicating and cooperating (Pokahr et al., 2005).

This thesis aims to evaluate the capabilities of distributed intelligent agents to detect anomalies in IoT. Anomaly detection mechanisms require distribution and a level of intelligence to be able to flexibly adapt to any changes in the IoT environment. Therefore, a suitable approach to be applied to this problem would be the BDI model, which will be examined later in the methodology chapter.

3.3 Research Gap

Based on previous literature review it can be stated that there exists a technology providing reasoning abilities as well as advent of a small computing device with powerful capabilities. Moreover, several studies have proposed several anomaly detection approaches. However, within the scope of the thesis no research was found studying the feasibility of utilizing reasoning capabilities of intelligent agents to detect anomalies in IoT systems. Furthermore, there is a lack of research in how intelligent agents can transfer raw data transmitted by sensors to useful information to make appropriate decisions regarding IoT security.

There are open questions around an optimal architecture for building intelligent agents while gaining the most advantages. In that light, a distributed agents' approach allows different architectures including centralized, localized and collaborative reasoning. In the centralized reasoning approach the sensor data are channeled to a central service, where it is processed. Meanwhile, the local intelligence mechanism avoids transferring data over the network and performs the reasoning process as close as possible to the agents. Like the local approach, the collaborative reasoning approach takes place locally, but it also makes it possible for the agents to interact between them. Therefore, further investigation should be done to identify the consequences of adopting either one of those architectural decisions. This thesis covers the conceptual idea of the three architectural approaches in the following chapter. However, it tries to contribute in the centralized reasoning domain by implementing a distributed sensor network of agents based on the centralized approach.

4 Methodology

This chapter will provide a discussion for the chosen design and technologies used for this project. The project addresses the security of IoT domain by investigating anomaly detection in IoT utilizing distributed intelligent agents. The area of study of this project is the new existing computing paradigm that extends traditional computing, such as server computing or office computing which do not support the concept of environmental interaction. The new computing paradigm used in the IoT domain brings interaction between sensors and the IoT devices. Therefore, one needs to understand computing systems in a slightly different way taking sensors, which transmit data over the IoT network, into consideration. The new computing paradigm raises several issues regarding security of the IoT environment as its scope is different and adds aspects not covered by the traditional environment. This thesis studies an approach to tackle this security problem through recognizing threats from the transmitted data, which will be conceptually presented next.

4.1 Reasons for utilizing “Jadex” framework and the “BDI” model

The project proposes the exploitation of the distributed intelligent agents’ technology, discussed in section 3.2.4, to demonstrate the agents’ reasoning capabilities for the purpose of anomaly detection. The BDI model, which was examined in section 3.2.4.4, is utilized by Jadex as a computational model to provide agents with reasoning capabilities. In addition, Jadex provides a framework for the creation of those goal-oriented agents. There are several reasons behind choosing Jadex as a paradigm for developing distributed agents. First, the Jadex tool supports the development of multi-agent systems by utilizing the strong structure and features provided by agent-based systems. This is done through adoption of object-oriented concepts combined with Java, as a programming language, which makes the development process smoother as it is a familiar programming environment. The Jadex framework facilitates development of agents on a distributed level by providing essential capabilities for the agents to communicate. Jadex-based agents can communicate regardless of their different platforms. Due to the platform awareness feature offered by the Jadex platform, an agent on a local platform can search for other agents on remote platforms to exchange services with them.

The Jadex platform supports cognitive agents by exploiting the BDI model. In addition to the benefits of the Jadex framework, the BDI architecture also has its own characteristics that make it a suitable architecture for developing intelligent agents. One of the reasons for selecting the BDI model as the basis for this project is that it realizes the creation of rational agents based on the concept of mental attitudes. The internal structure and capabilities of the BDI agents are determined by the idea of mental attitudes which makes it easier to program, understand and deal with the agent-based system as it resembles how humans think and takes decisions on an abstract level. Additionally, the BDI agents are aware of their surrounding and the objectives they seek to achieve in a rational and goal-oriented manner. This rational thinking leads the BDI agents to use its reasoning capabilities by weighing up the available options before performing an action. Therefore, they behave based on their knowledge about their environment and the goals they are programmed to achieve.

4.2 Limitations of sensor networks

Despite the various uses sensors possess in different areas, they have many limitations that should be accounted for. One is restricted resources in terms of memory, processing speed and radio range through which they communicate with each other. As a result, they are not capable of performing complex computations. In general, based on their internal structure and abilities, sensors do a passive job without reacting to changes occurring in their environment. The reason for that is that they are not aware of whether the measured values indicate a normal or abnormal behavior.

An area deserving of attention is the role of anomaly detection in recognizing the existence of anomalous behavior in a network or system. It could be due to a defective sensor that measures functions incorrectly or resulting from an intrusive activity. Most importantly, recognized deviations should be analyzed further to preserve the quality and reliability of the data as well as the functioning of the network. However, in both cases, the capabilities of a sensor network do not enable it to detect abnormal patterns passing through the network traffic or to carry out analytical operations to identify the cause of these anomalies and react to it as the sensor network lacks intelligence.

4.3 Sensor network of distributed intelligent BDI agents

Distributed agents play a big role in the detection and analysis mechanism through utilizing their communication and reasoning capabilities as previously discussed. To clarify the conceptual idea of the project and investigate whether the anomaly detection approach can be executed as close to the source of data as possible, an explanation for several aspects regarding the development of a distributed agents' network will be provided. Then an example will be given for the architecture and basic functionality of a network, consisting of distributed intelligent agents with sensing and reasoning capabilities. To simplify the coming conceptual idea, a simplified use case focusing on the agents' implementation will be discussed in the next chapter.

4.3.1 Components of the BDI agent

A BDI agent can differentiate between normal and abnormal behavior based on its beliefs in terms of predefined conditions or fixed values for specific measurements set by the owners. The beliefs of the BDI agent represent the environment characteristics which are updated after performing any action as well as what the agent believes about the world. This information is mostly acquired by the agent through observing its environment, is identified according to the current state of its world and is built up over time. As for the desires, they stand for what the agent seeks to achieve at the end, which is the motivation behind its actions that impacts its belief. An agent can have multiple desires, from which it should select only one according to its current belief as they might be conflicting. For reaching its goal, the agent selects the suitable plan among several options. The following real-life examples will illustrate more the concept.

4.3.2 Architectural paradigms of intelligent distributed agents

Intelligent analysis in the sensor network can take place centrally but it can be also done locally. Local analysis is facilitated by the emergence of inexpensive, computationally powerful computing devices joining all the required capabilities and technologies together. That is why these devices can run an intelligent agents' implementation with attached sensors transmitting data to these agents. The local approach makes use of this computational power to enable running the anomaly detection and reasoning process closer to the sensor to be able to perform the initial steps of reasoning and analysis closer to the device. Consequently, local decisions can be taken avoiding the need for centralized analysis service.

In that sense, it is also feasible to take these decisions to a further step by deploying specific actors on the agents to be able to act on the environment and control the surrounding devices. Utilization of intelligent agents provides two different paradigms for the reasoning process. Each agent can function independently and make quick and independent decisions accordingly or it can also exchange information with other agents to eventually reach a decision together.

Despite the advantages of the local approach, some conclusions cannot be reached by a single agent. Therefore, the combination of agents is needed as it provides a broader picture that makes many aspects, which are not identified by an individual agent, clearer. Most of these conceptual ideas will be applied to the following example illustrating also the reasoning capabilities of a BDI agent.

4.3.1 Example for a household sensor network

As shown in Figure 14, there are five distributed BDI agents connected to the same network. Each agent's software is deployed on a small device with a specific type of sensor attached to it based on the functionality of the agent. These sensors are deployed on different kinds of IoT devices placed in a household. This section presents four different scenarios for detecting anomalies in an IoT environment

4.3.1.1 Scenario 1

Imagine the owners of the house are travelling for few weeks and an agent with a motion sensor is placed inside the house next to their door to recognize any movements there. During their travel period the *motion agent* is informed of their absence, for example for 2 weeks, so its belief is that during this period no one is in the house. The desire of the agent is to keep the intruders away from the house and accordingly generates an instant alarm as soon as it receives a motion signal to notify the owners of someone in the house. The agent sends a signal to a switch, responsible for immediately opening the door of the guarding dogs' house to terrify the intruders. All actions done by the agent in case of the intrusion event represent its intention.

4.3.1.2 Scenario 2

A sensor for measuring humidity of the plants can be used to ensure they are watered each time the humidity level in the soil gets below a predefined value to protect them against drying which is the agent's desire. For example, the agent believes that the soil humidity should not exceed a specific limit. Accordingly, the agent's system works as follows: the *humidity agent* collects the humidity measurements every 6 hours to check if the level is acceptable or an unusual humidity value is captured, while at the same time its belief is updated with each new value. The plan of the agent depends on the measured value. If the humidity falls below the normal value, the agent intends to send a signal to a switch that turns on an automated watering machine. Afterwards, during the watering process, the agent measures the humidity continuously until it reaches the normal level and then switches to another action which is sending a signal to the switch to turn the machine off.

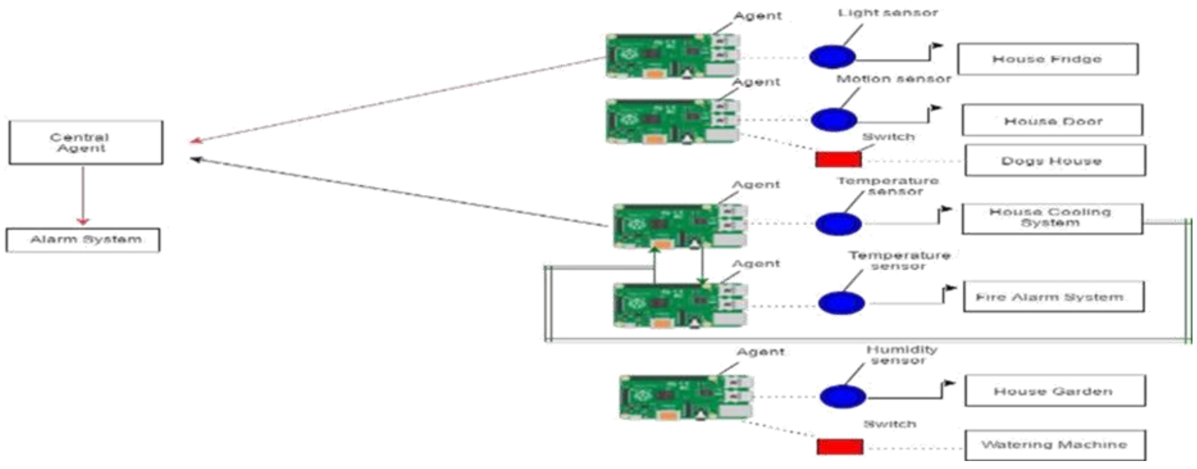


Figure 14: Sensor network of distributed intelligent agent

5 Implementation

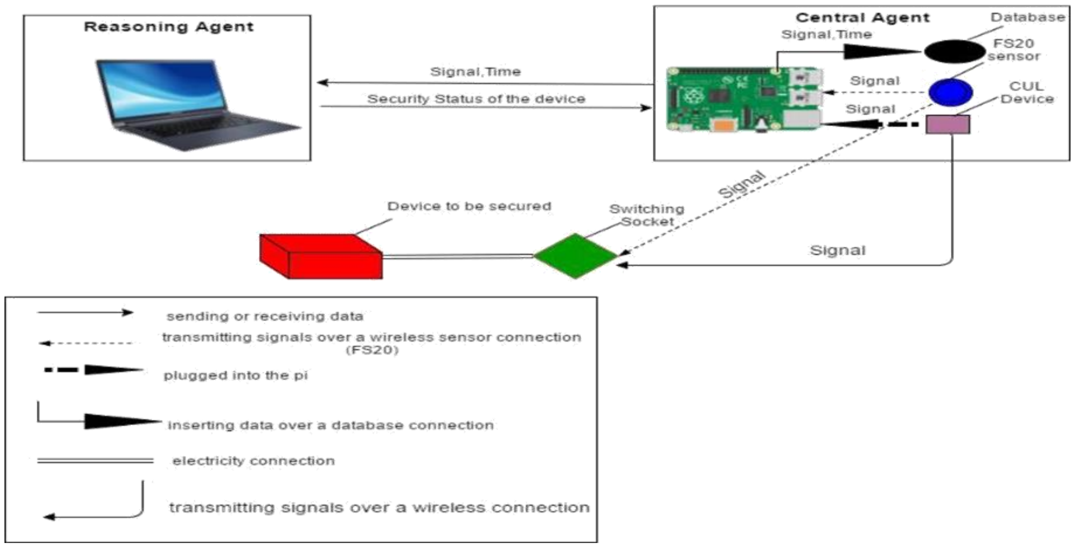
5.1 Use Case Description

This chapter provides a simplified example to limit the scope of the above described network of intelligent distributed agents. It illustrates the conceptual use of the BDI agents with their intelligent reasoning and sensing power. In the implementation of the project, the architecture of the agents is done as simply as possible to be more understandable, by programming each agent to be specialized in performing one simple operation. The reason behind this approach is to avoid building a huge and complicated program in which all actions are carried out together. This is shown with the application of two BDI agents, one connected to a sensor collecting and transmitting some data to the other agent, which performs immediate reasoning operations on a single event and acts as a central agent.

The actual implementation of the project is like the different possible agents' architectures discussed in the methodology. It comprises two BDI agents, the first connected to a sensor transmitting signals responsible for turning on and off a device. These signals are collected by the agent, which checks when the signal was received and then stores both information in a database. Afterwards, it sends them further to the central agent, which is the one responsible for examining whether the device to be secured was opened or closed during the specified time range by the authorized user and accordingly reports on the security status of the device to inform the user of taking the suitable actions.

5.2 Development environment

In order to build a sensing agent that communicates with a simple reasoning immediate acting agent, various hardware and software components were utilized, illustrated by the diagram in Figure 15 showing the connections between the components. This section presents the required elements for the two agents. Both agents are developed based on the BDI model discussed earlier. However, there are several differences between them regarding the implementation, programs and hardware used to execute each program, which are mentioned briefly next.



Reasoning Agent
Central Agent
Database
Signal,Time
FS20 sensor
Signal
CUL Device
Signal
Signal,Time
Security Status of the device
Signal
Device to be secured
Switching Socket
Signal
sending or receiving data
transmitting signals over a wireless sensor connection (FS20)
plugged into the pi
inserting data over a database connection
electricity connection
transmitting signals over a wireless connection

Figure 15: Components Diagram

5.2.1 Sensing BDI agent

Looking at each agent separately, the agent software of the sensing agent is deployed on a raspberry pi, a small computing device, to which a “CUL device” is attached making it capable to send and receive signals. Those received signals are initially sent by an “FS20 sensor” to either switch on or off a device. This device, which is to be secured, is plugged into an on/off radio switching socket that receives any transmitted signal and switches on or switches off the device based on it as shown in Figure 15 and 16.

5.2.1.1 Description of the sensor

One of the components utilized in the environment of the sensing agent is a sensor, operating with the FS20 protocol, a wireless protocol that enables communication between devices. It provides easy access to the transmitted data over a wireless channel and is widely used for home automation purposes. The FS20 device employed here is considered a transmitter of on/off signals as it is used manually to emit a signal to the switching socket, connected to the device, and to the CUL device as well. The transmission process works as follows: the FS20 device generates and transmits signals when triggered by button pressing. In theory this FS20 device works also as a light sensor that transmits signals once it detects light. However, this functionality is not used here due to the scope of the use case.

5.2.1.2 Transfer of data through the CUL device

Adding to the Jadex software utilized to build the agent, another technology is used to implement the functionality of the agent. It is called the serial communication and it is enabled by the usage of the CUL device which facilitates the transfer of data between the agent and the radio switching socket as well as the sensor as shown in Figure 18. The signal received by the agent is depicted by a normal shaped arrow. The signal is transmitted by the sensor using the FS20 protocol to the device through the switching socket and read by the CUL device, which forwards it further to the agent to be sent to the central agent and saved with its time in the database. The sent signal by the agent is illustrated by a dotted arrow and the process runs as follows: the agent sends a signal, which is transferred by the CUL device and received by the switching socket over a wireless connection to open or close the device. Once the sensing agent sends a signal, the signal as well as its time are also transmitted to the central agent and saved in the database. This serial connection is established and implemented using Java as a programming language via the “RXTX library”.

5.2.1.3 Inserting the data into a database

The other functionality of the sensing agent, which is saving the received or transmitted signals with their timings, is enabled by use of the “LAMP stack” (Linux, Apache, MySQL and PHP). It is open source software to set up and run web servers and connects the sensing agent to the database. To run a database on the Pi, one would not need a “LAMP stack”. Nonetheless, this is done to simplify the database administration for convenience reasons. Additionally, the utilization of the database has a long-term purpose as the stored data can be processed in the future. In fact, when the data is received by the central agent and the reasoning process is performed, the transmitted data cannot be accessed anymore. Therefore, the data must be stored at a secure place so it can be retrieved when necessary as shown in Figure 18. For example, the central agent can access the data collected in a month to evaluate the overall security status of the device during this month.

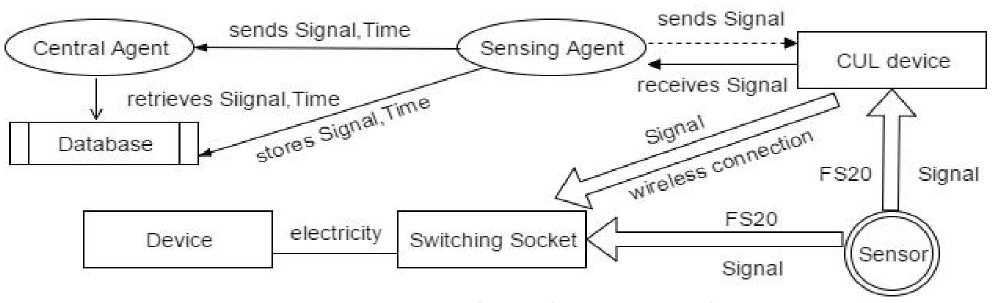


Figure 16: Agent sending and receiving signals

5.2.2 Central BDI Agent

The central reasoning agent set up is simpler than the sensing agent as the software runs on a laptop without using any extra hardware. Once the signal is received by the central agent it identifies whether or not the time of its transmission is within the acceptable range. Consequently, it sends a message to the sensing agent to inform it of the security of the device, whether it is safe or exposed to any type of security threats.

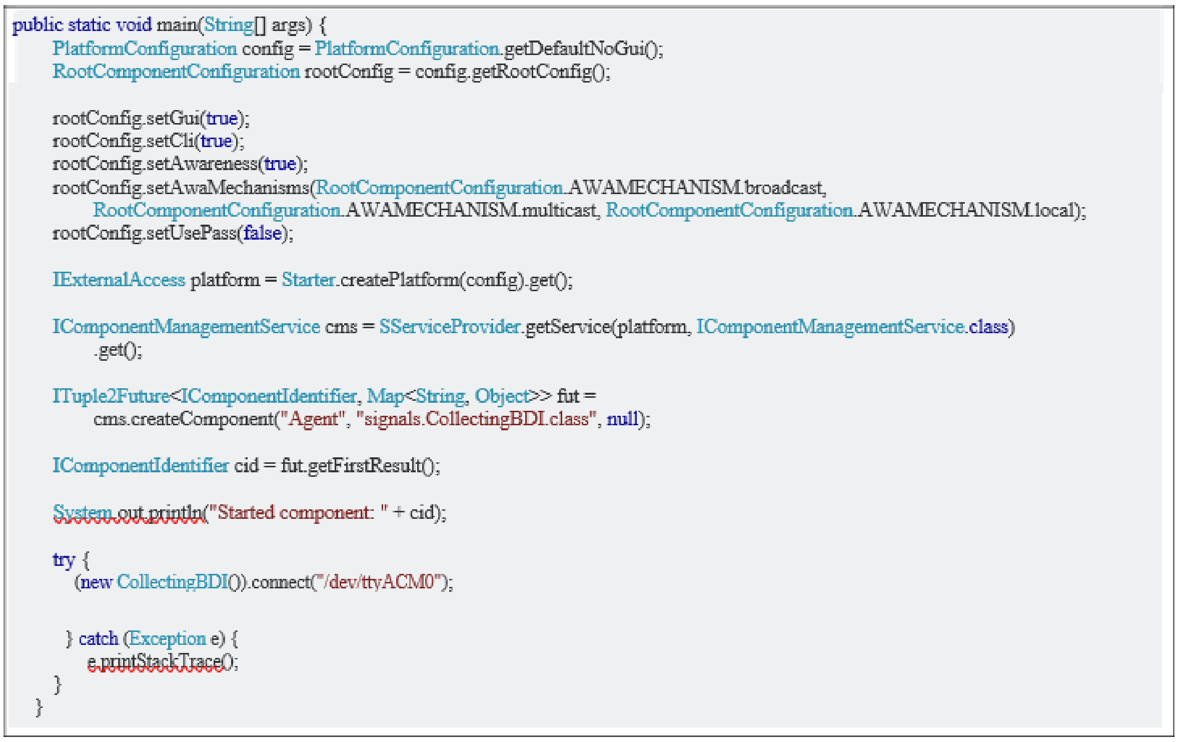
5.3 Project Implementation

This section details implementation of both agents and how they communicate with each other.

5.3.1 Initialization of both agents' platform

As described in the previous section, there is a communication between both agents, represented by two java classes: *ReasoningBDI and CollectingBDI*. Although they are distributed and running on two different machines, there is a connection between them facilitated by the Platform Awareness mechanism provided by Jadex. It allows an agent to search for another remote agent and detect that it exists to be able to provide to it or require from it services. According to this mechanism, to execute an agent a running platform is needed, which is started in the main ( ) method of each agent shown in Figure 17 and Figure 20. The main ( ) method is structured as follows:

- PlatformConfiguration and RootComponentConfiguration objects are created to adjust the platform to the user's needs including launching a user interface for the agent, the "Jadex Control Center", which represents the main access point for the available runtime tools, shown in Figure 18 and Figure 21, and enabling the awareness mechanism to search for remote agents once the agent is started.
- The configuration object is passed to createPlatform( ) method to start the platform.
- The IExternalAccess object belonging to the platform is used to retrieve the platform's ComponentManagementService, responsible for starting the agent itself as it takes as an input the name of the agent's class such as ReasoningBDI or CollectingBDI.
- Identifier of the started agent is retrieved and printed on the console depicted in Figure 19 and Figure 22.
- In the main ( ) method of the CollectingBDI agent there is an additional piece of code that connects the agent to the CUL device once it is started.



```java
public static void main(String[] args) {
    PlatformConfiguration config = PlatformConfiguration.getDefaultNoGui();
    RootComponentConfiguration rootConfig = config.getRootConfig();

    rootConfig.setGui(true);
    rootConfig.setCli(true);
    rootConfig.setAwareness(true);
    rootConfig.setAwaMechanisms(RootComponentConfiguration.AWAMECHANISM.broadcast,
        RootComponentConfiguration.AWAMECHANISM.multicast, RootComponentConfiguration.AWAMECHANISM.local);
    rootConfig.setUsePass(false);

    IExternalAccess platform = Starter.createPlatform(config).get();

    IComponentManagementService cms = SServiceProvider.getService(platform, IComponentManagementService.class)
        .get();

    ITuple2Future<IComponentIdentifier, Map<String, Object>> fut =
        cms.createComponent("Agent", "signals.CollectingBDI.class", null);

    IComponentIdentifier cid = fut.getFirstResult();

    System.out.println("Started component: " + cid);

    try {
      (new CollectingBDI()).connect("/dev/ttyACM0");


     } catch (Exception e) {
        e.printStackTrace();
    }
  }
```

Figure 17: Main method of the CollectingBDI

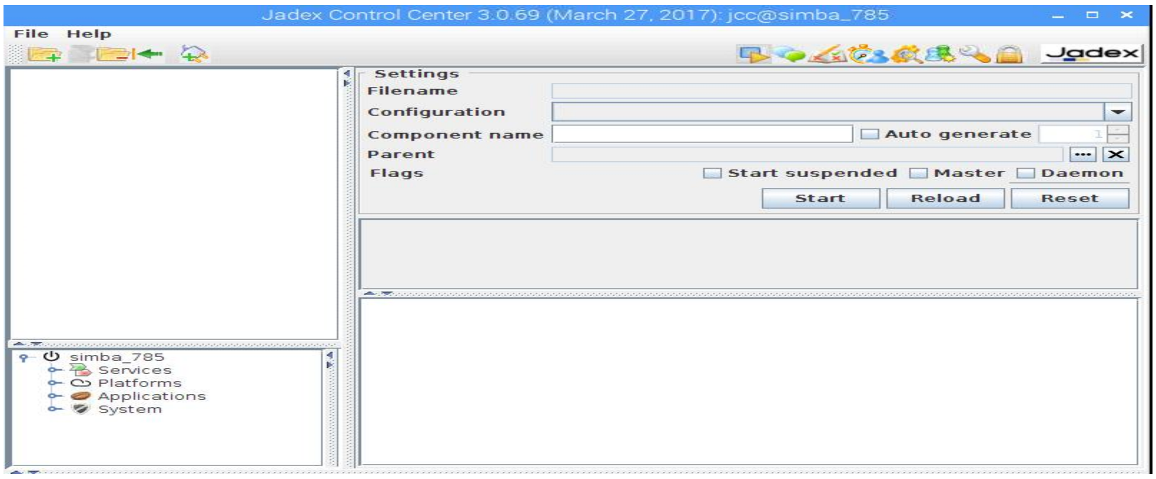
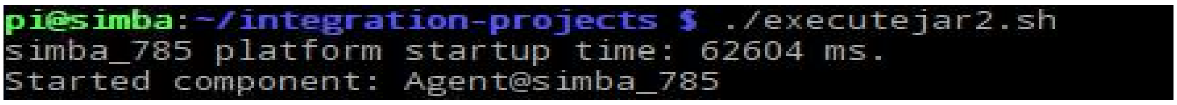
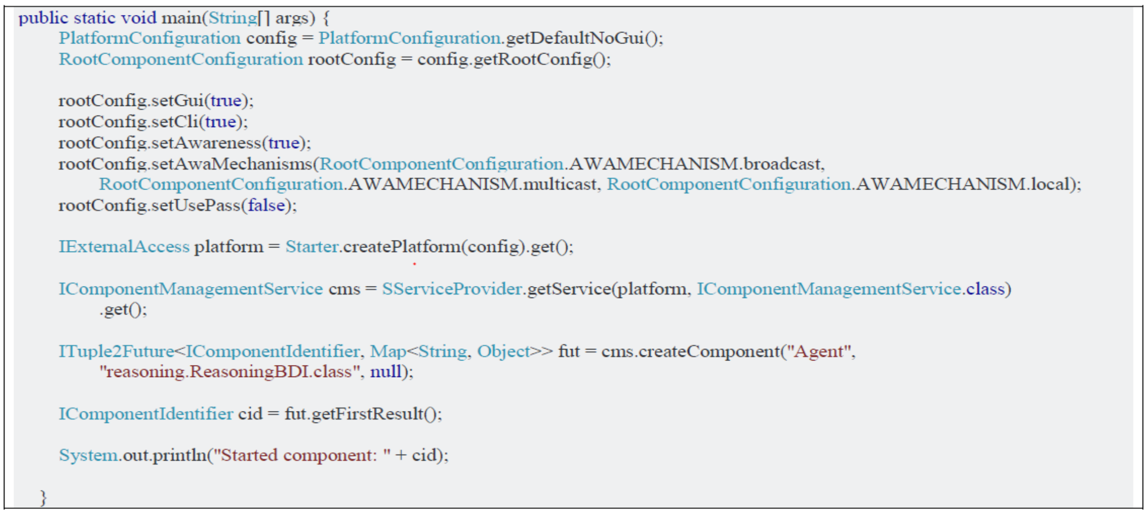


Figure 18: Jadex Control Center of the CollectingBDI



```
pi@simba:~/integration-projects $ ./executejar2.sh
simba_785 platform startup time: 62604 ms.
Started component: Agent@simba_785
```

Figure 19: CollectingBDI console



```
public static void main(String[] args) {
      PlatformConfiguration config = PlatformConfiguration.getDefaultNoGui();
      RootComponentConfiguration rootConfig = config.getRootConfig();

      rootConfig.setGui(true);
      rootConfig.setCli(true);
      rootConfig.setAwareness(true);
      rootConfig.setAwaMechanisms(RootComponentConfiguration.AWAMECHANISM.broadcast,
            RootComponentConfiguration.AWAMECHANISM.multicast, RootComponentConfiguration.AWAMECHANISM.local);
      rootConfig.setUsePass(false);

      IExternalAccess platform = Starter.createPlatform(config).get();

      IComponentManagementService cms = SServiceProvider.getService(platform, IComponentManagementService.class)
            .get();

      ITuple2Future<IComponentIdentifier, Map<String, Object>> fut = cms.createComponent("Agent",
            "reasoning.ReasoningBDI.class", null);

      IComponentIdentifier cid = fut.getFirstResult();

      System.out.println("Started component: " + cid);

   }
```

Figure 20: Main method of the ReasoningBDI

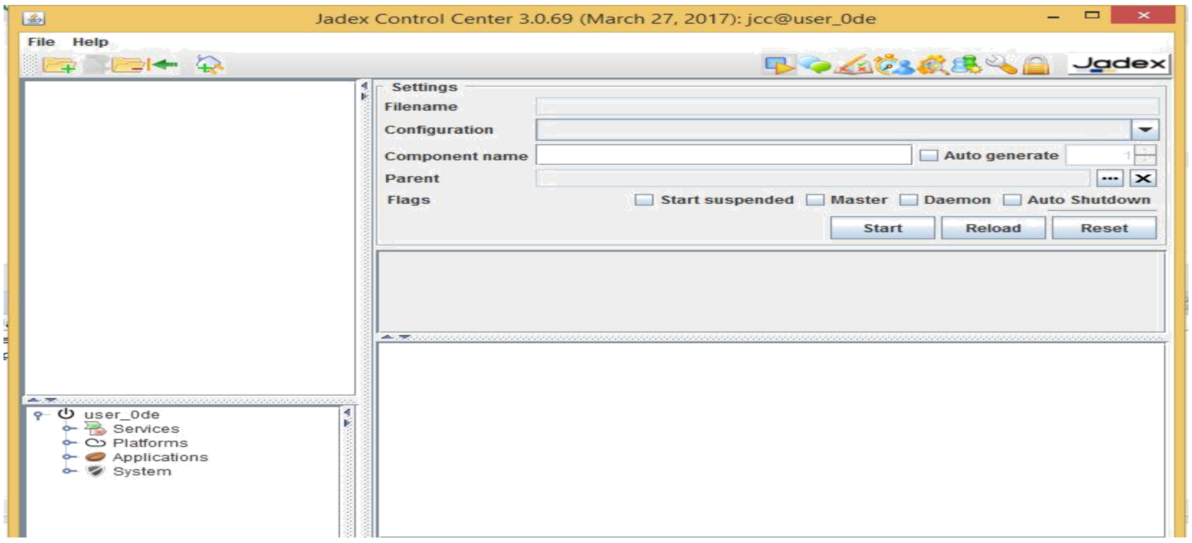
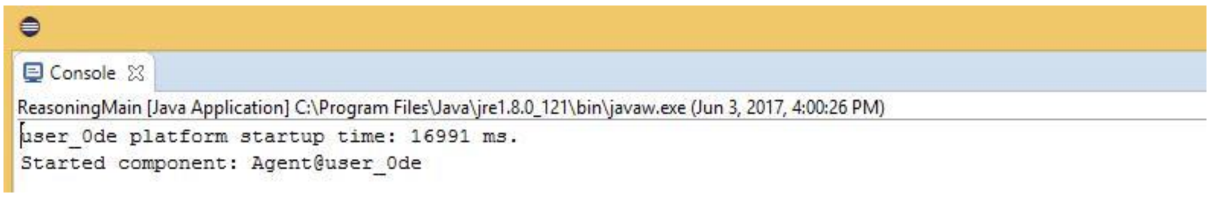


Figure 21: Jadex Control Center of the ReasoningBDI

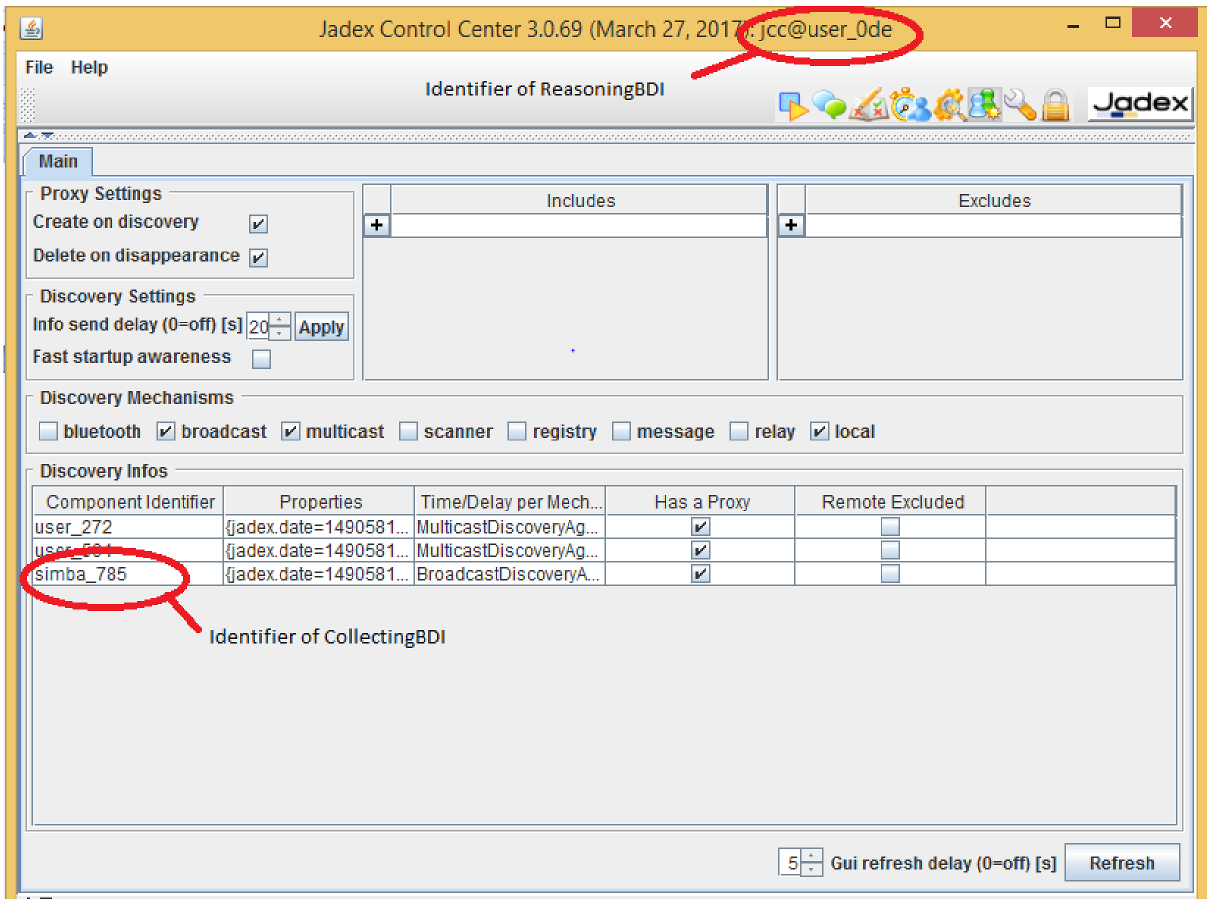


```
Console
ReasoningMain [Java Application] C:\Program Files\Java\jre1.8.0_121\bin\javaw.exe (Jun 3, 2017, 4:00:26 PM)
user_0de platform startup time: 16991 ms.
Started component: Agent@user_0de
```

Figure 22: ReasoningBDI console

5.3.2 Interaction between agents

In the header of the JCC there is an identifier for the started agent which is used to search for the agent through the *IAwarnessManagementService* provided by Jadex. Figure 23 lists the available agents running on remote platforms including the *CollectingBDI* identified as simba_785, which indicates it is found by the *ReasoningBDI* identified by jcc@user_0de. Both can start to interact.


Jadex Control Center 3.0.69 (March 27, 2017): jcc@user_0de
File
Help
Identifier of ReasoningBDI
Jadex
Main
Includes
Excludes
Identifier of CollectingBDI
Apply
Refresh

Proxy Settings
Create on discovery [✓]
Delete on disappearance [✓]

Discovery Settings
Info send delay (0=off) [s] 20
Fast startup awareness [ ]

Discovery Mechanisms
[ ] bluetooth [✓] broadcast [✓] multicast [ ] scanner [ ] registry [ ] message [ ] relay [✓] local

Discovery Infos

Component Identifier	Properties	Time/Delay per Mech...	Has a Proxy	Remote Excluded
user_272	{jadex.date=1490581...	MulticastDiscoveryAg...	✓	☐
user_5[illegible]	{jadex.date=1490581...	MulticastDiscoveryAg...	✓	☐
simba_785	{jadex.date=1490581...	BroadcastDiscoveryA...	✓	☐

5 Gui refresh delay (0=off) [s]

Figure 23: Remote platforms found by the ReasoningBDI

Adding to the *CollectingBDI* and *ReasoningBDI* projects forming the agent's sensors network, another project is necessary to facilitate requiring and providing services among them. The *service* project provides a shared functionality between both agents which is a service interface shown in Figure 24 that is required by the *CollectingBDI* and provided by the *ReasoningBDI.* It takes the signal and its time as inputs from the collector and returns a string containing the security status of the device from the reasoning agent. The detailed behavior of this service and how it is exchanged between both agents will be explained in the next section that covers the structure and functionality of both agents.

```
@Security(Security.UNRESTRICTED)
public interface IAnalyzeSignal {
    public IFuture<String> analyze(String sig, String time);

}
```

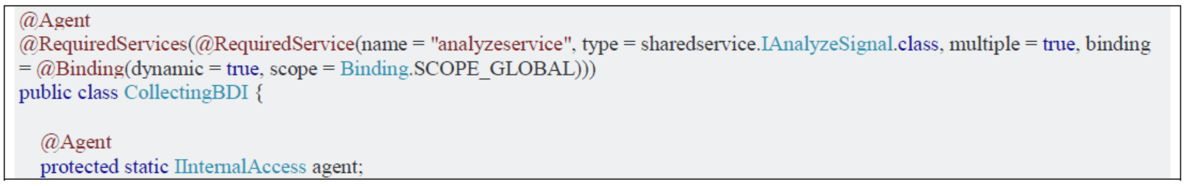
Figure 24: The interface of the shared service

5.3.3 Implementation of the sensing agent

The code structure of the sensing agent is an integration between the three different functionalities:

1) <u>The BDI agent features:</u>

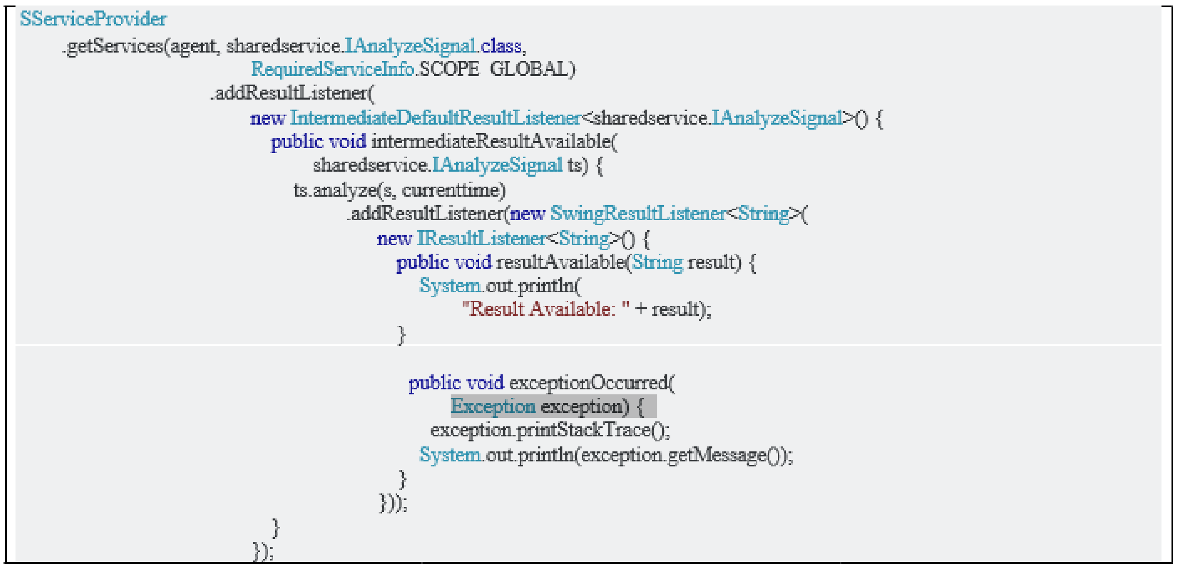
   For the agent to be recognized as a BDI agent, its class name should end with BDI and an @Agent annotation is included before the class declaration. To be able to use a specific service, it has to be declared as a required service inside the agent through the @RequiredService annotation. This annotation includes a name (analyzeservice) to reference the required service, the name of the service interface (IAnalayzeSignal) and multiple arguments to enable using multiple instances of the service if needed. Furthermore, the annotation @Binding defines the parameters of the service binding that is established by Jadex between the providing and requiring agents. Those parameters represent the scope which is set to be able to find the service provided by a remote agent and dynamic, set to true, which initiates a new search each time the service is accessed by another agent. The agent has one attribute declared as an IInternalAccess which refers to the provider of the agent and enables fetching the required service as shown in Figure 25.



```
@Agent
@RequiredServices(@RequiredService(name = "analyzeservice", type = sharedservice.IAnalyzeSignal.class, multiple = true, binding
= @Binding(dynamic = true, scope = Binding.SCOPE_GLOBAL)))
public class CollectingBDI {

   @Agent
   protected static IInternalAccess agent;
```

Figure 25: Declaration of the CollectingBDI

Figure 26 illustrates how the service invocation is created in the context of a BDI agent. As soon as the signal is received or sent by the agent, the agent starts to look up the remote service and waits for the response, which is the responsibility of the result listener. Once the service is found, *analyze ( ) method* included in the service interface mentioned earlier is invoked on the service. It takes as an input the service and when it was transmitted and returns a statement printed on the console of the sensing agent.



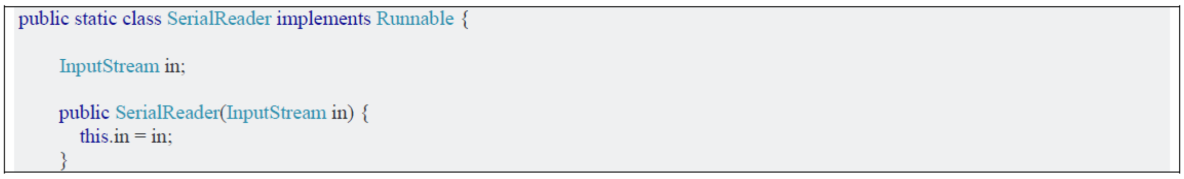
```
SServiceProvider
    .getServices(agent, sharedservice.IAnalyzeSignal.class,
                                RequiredServiceInfo.SCOPE_GLOBAL)
                        .addResultListener(
                            new IntermediateDefaultResultListener<sharedservice.IAnalyzeSignal>() {
                                public void intermediateResultAvailable(
                                    sharedservice.IAnalyzeSignal ts) {
                                ts.analyze(s, currenttime)
                                    .addResultListener(new SwingResultListener<String>(
                                        new IResultListener<String>() {
                                            public void resultAvailable(String result) {
                                                System.out.println(
                                                    "Result Available: " + result);
                                            }

                                            public void exceptionOccurred(
                                                Exception exception) {
                                              exception.printStackTrace();
                                              System.out.println(exception.getMessage());
                                            }
                                        }));
                                }
                            });
```

Figure 26: Agent requesting analyze service

2) <u>The Serial Communication feature:</u>

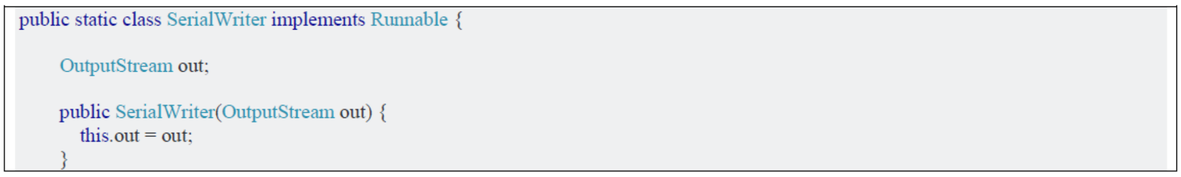
Before requesting the service, the agent either sends or receives a signal. The signal transmission is facilitated by the adoption of the serial communication technology. It is implemented through two inner classes in the sensing agent, *SerialReader* and *SerialWriter* as well as method *connect ( )* presented in Figure 27, 28 and 29. In the *main ( )* method the connection to the CUL *device* was established with *connect ( )* method, which creates a connection to the port of the device and starts the *SerialWriter* and *SerialReader.* Referring back to the process of sending and receiving signals in Figure 18, the *SerialReader* is responsible for the receiving process. After starting the agent, the button of the sensor is pressed to transmit a signal. When the agent starts the reader of the *CUL device* starts as well and it keeps reading any bytes available at the port with its *InputStream* until a complete signal is fetched by the agent, which is known when the reader sees a new line in form of /r/n. Afterwards a *Date* object is created to get the current time of the signal, save both information in the database and send them to the central agent by requesting the shared service. This functionality is included in method *run ( )* in the *SerialReader* found in Figure 30.

The *SerialWriter* functions in the same way. However, it enables sending signals so the data flows in the opposite direction. It writes the stream of bytes representing the signal via its *OutputStream* on the port as Figure 31 shows and once the signal is complete, it is transmitted by the *CUL device* to the switching socket connected to the device. At the same time the available information is also stored in the database and received by the *ReasoningBDI* to react on it.



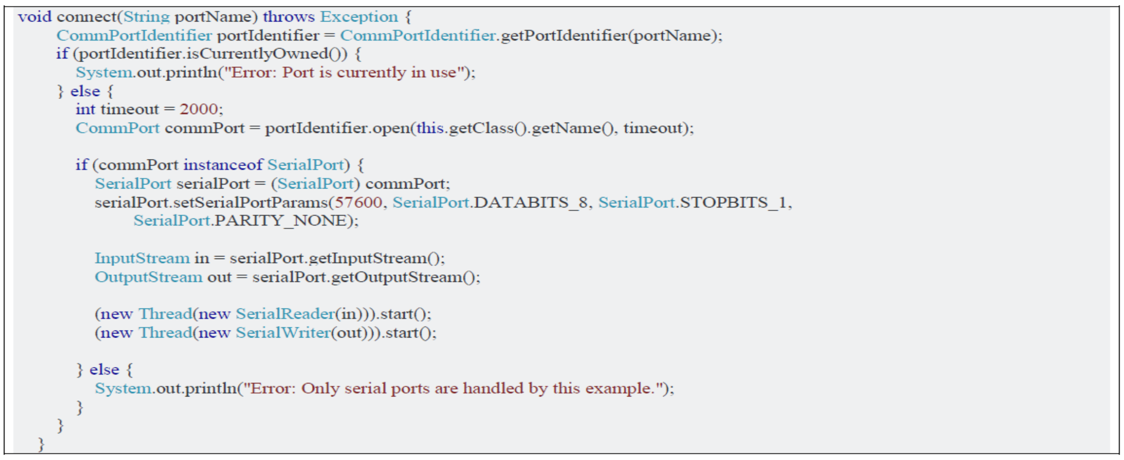
```
public static class SerialReader implements Runnable {

    InputStream in;

    public SerialReader(InputStream in) {
        this.in = in;
    }
```

Figure 27: SerialReader of the CollectingBDI



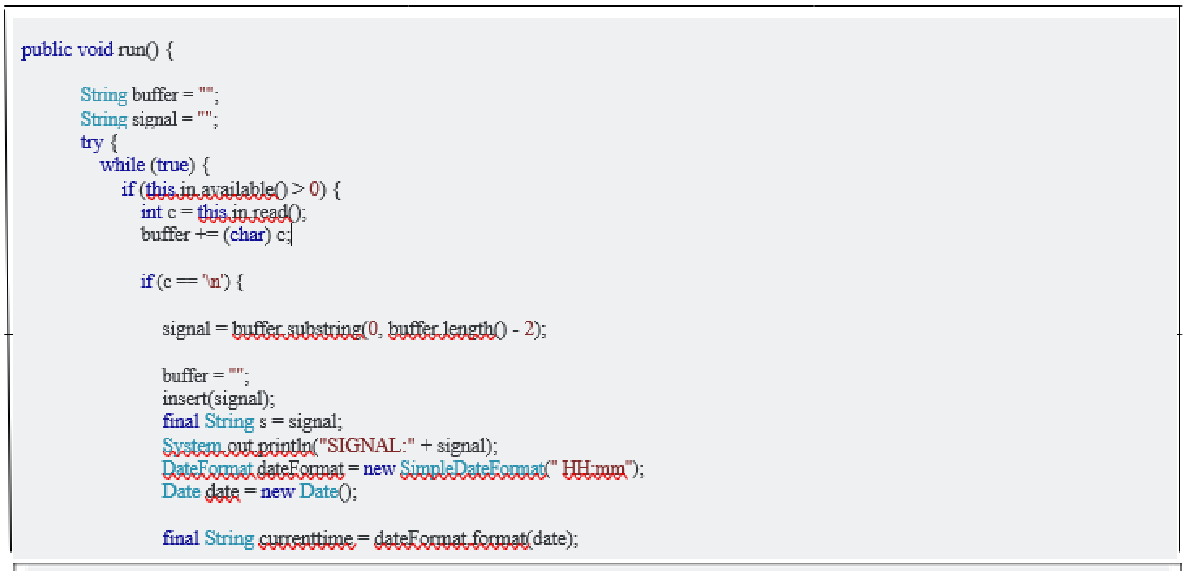
```
public static class SerialWriter implements Runnable {

    OutputStream out;

    public SerialWriter(OutputStream out) {
        this.out = out;
    }
```

Figure 28: SerialWriter of the CollectingBDI



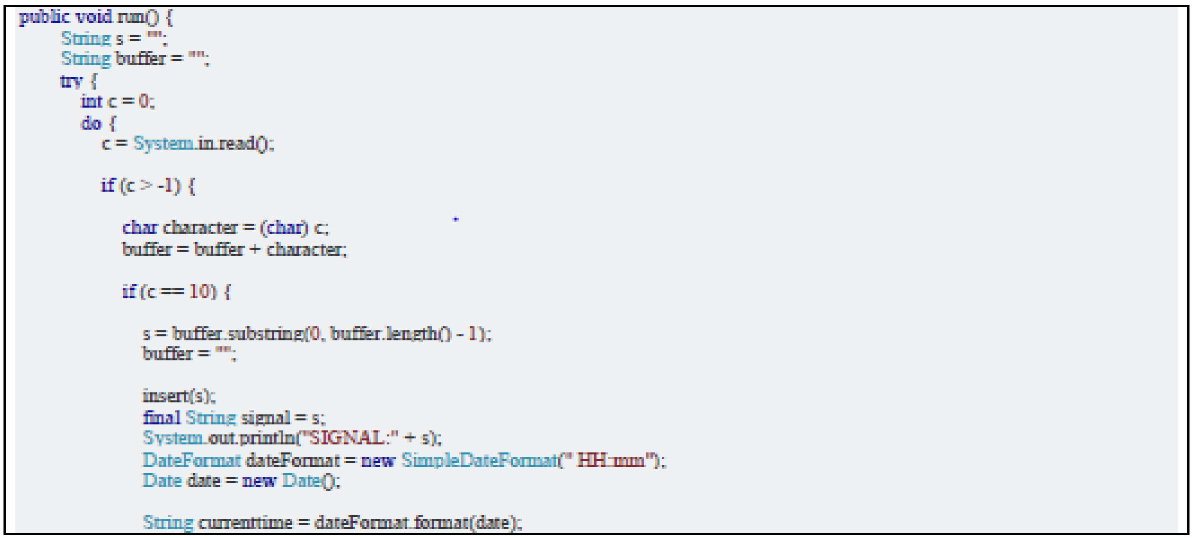
```
void connect(String portName) throws Exception {
    CommPortIdentifier portIdentifier = CommPortIdentifier.getPortIdentifier(portName);
    if (portIdentifier.isCurrentlyOwned()) {
        System.out.println("Error: Port is currently in use");
    } else {
        int timeout = 2000;
        CommPort commPort = portIdentifier.open(this.getClass().getName(), timeout);

        if (commPort instanceof SerialPort) {
            SerialPort serialPort = (SerialPort) commPort;
            serialPort.setSerialPortParams(57600, SerialPort.DATABITS_8, SerialPort.STOPBITS_1,
                    SerialPort.PARITY_NONE);

            InputStream in = serialPort.getInputStream();
            OutputStream out = serialPort.getOutputStream();

            (new Thread(new SerialReader(in))).start();
            (new Thread(new SerialWriter(out))).start();

        } else {
            System.out.println("Error: Only serial ports are handled by this example.");
        }
    }
}
```

Figure 29: Method connect ( ) of the CollectingBDI



```
public void run() {

        String buffer = "";
        String signal = "";
        try {
          while (true) {
            if (this.in.available() > 0) {
              int c = this.in.read();
              buffer += (char) c;

              if (c == '\n') {

                signal = buffer.substring(0, buffer.length() - 2);

                buffer = "";
                insert(signal);
                final String s = signal;
                System.out.println("SIGNAL:" + signal);
                DateFormat dateFormat = new SimpleDateFormat(" HH:mm");
                Date date = new Date();

                final String currenttime = dateFormat.format(date);
```

Figure 30: Method run ( ) of the SerialReader

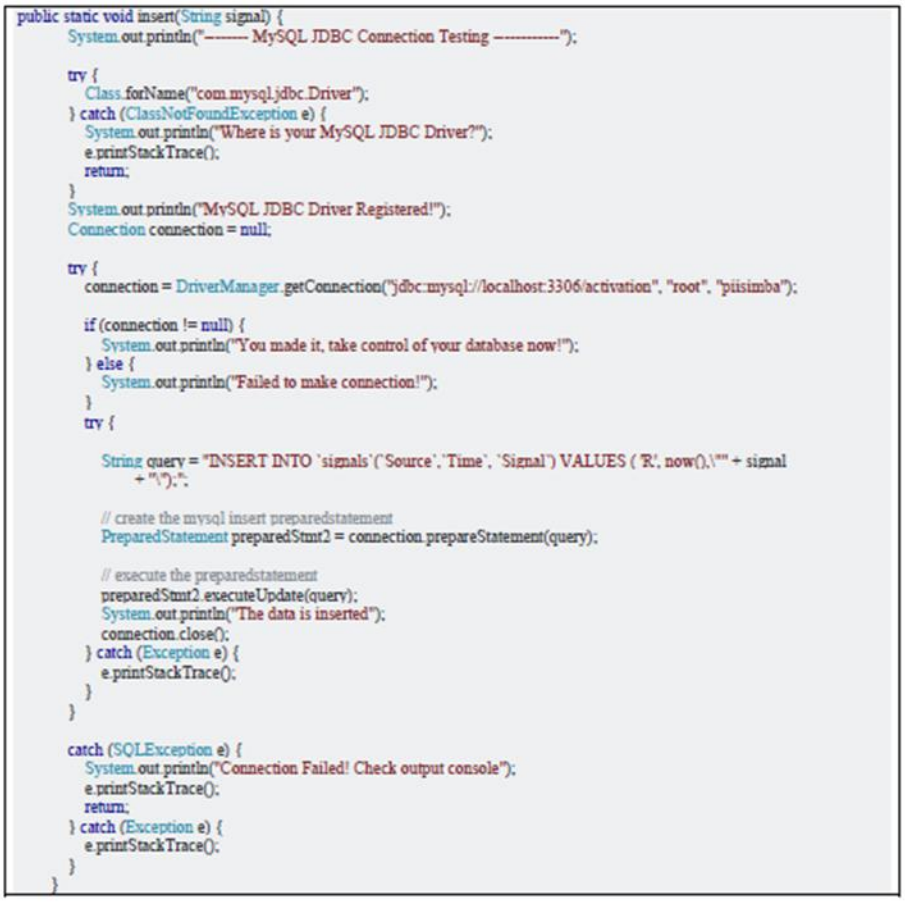


```
public void run() {
      String s = "";
      String buffer = "";
      try {
        int c = 0;
        do {
          c = System.in.read();

          if (c > -1) {

            char character = (char) c;
            buffer = buffer + character;

            if (c == 10) {

              s = buffer.substring(0, buffer.length() - 1);
              buffer = "";

              insert(s);
              final String signal = s;
              System.out.println("SIGNAL:" + s);
              DateFormat dateFormat = new SimpleDateFormat(" HH:mm");
              Date date = new Date();

              String currenttime = dateFormat.format(date);
```

Figure 31: Method run ( ) of the SerialWriter

3) <u>Storage Feature:</u>

Whether the signal is sent or received, in both cases it is stored in the database. When the signal is available at the sensing agent, method *insert ( )* illustrated in Figure 32 is called by method *run ( )*. It establishes a connection between the agent and the database called *activation* and then inserts the data in *signals* table depicted in Figure 33 by executing an insertion query. This query includes 3 columns: the source of the signal, which identifies whether it came from the writer (sent signal) or the reader (received signal), the signal itself and calls method *now ( )* that gets the current time. The ID attribute is not included in the query as it is incremented automatically. Figure 34 shows some inserted records in table *signals.*

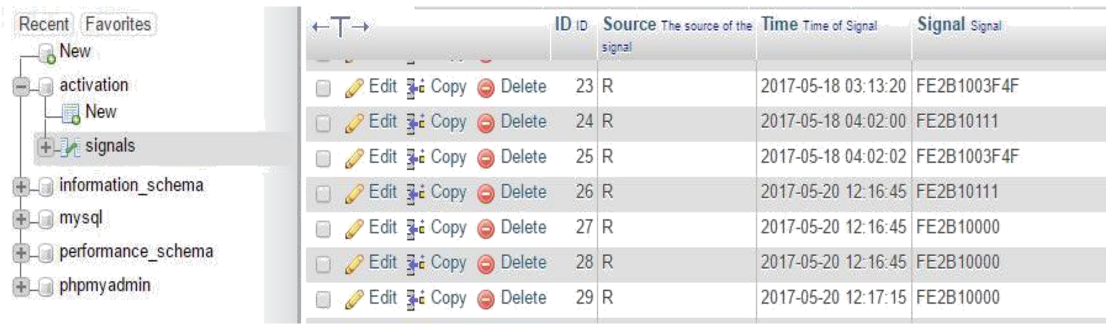


```java
public static void insert(String signal) {
        System.out.println("-------- MySQL JDBC Connection Testing ------------");

        try {
            Class.forName("com.mysql.jdbc.Driver");
        } catch (ClassNotFoundException e) {
            System.out.println("Where is your MySQL JDBC Driver?");
            e.printStackTrace();
            return;
        }
        System.out.println("MySQL JDBC Driver Registered!");
        Connection connection = null;

        try {
            connection = DriverManager.getConnection("jdbc:mysql://localhost:3306/activation", "root", "piisimba");

            if (connection != null) {
                System.out.println("You made it, take control of your database now!");
            } else {
                System.out.println("Failed to make connection!");
            }
            try {

                String query = "INSERT INTO `signals`(`Source`,`Time`, `Signal`) VALUES ('R', now(),\"" + signal
                        + "\");";

                // create the mysql insert preparedstatement
                PreparedStatement preparedStmt2 = connection.prepareStatement(query);

                // execute the preparedstatement
                preparedStmt2.executeUpdate(query);
                System.out.println("The data is inserted");
                connection.close();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }

        catch (SQLException e) {
            System.out.println("Connection Failed! Check output console");
            e.printStackTrace();
            return;
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

Figure 32: Method insert ( ) of the CollectingBDI

signals	
ID	INT
Source	VARCHAR(20)
Time	DATETIME
Signal	VARCHAR(20)

Figure 33: Schema of the database table



	ID ID	Source The source of the signal	Time Time of Signal	Signal Signal
Edit Copy Delete	23	R	2017-05-18 03:13:20	FE2B1003F4F
Edit Copy Delete	24	R	2017-05-18 04:02:00	FE2B10111
Edit Copy Delete	25	R	2017-05-18 04:02:02	FE2B1003F4F
Edit Copy Delete	26	R	2017-05-20 12:16:45	FE2B10111
Edit Copy Delete	27	R	2017-05-20 12:16:45	FE2B10000
Edit Copy Delete	28	R	2017-05-20 12:16:45	FE2B10000
Edit Copy Delete	29	R	2017-05-20 12:17:15	FE2B10000

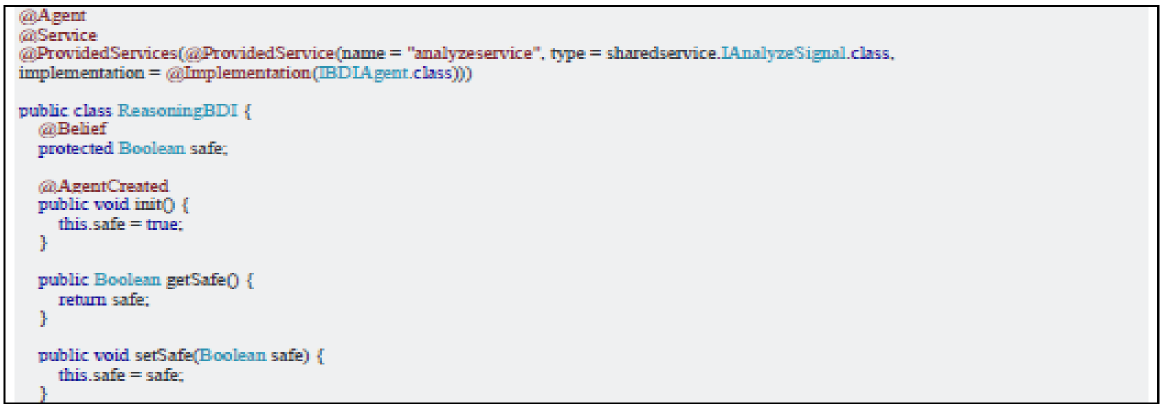
Figure 34: Records from table signals

5.3.4 Implementation of the reasoning agent

The reasoning agent (*ReasoningBDI*) serves as the provider of the required service as depicted in Figure 35. The agent has a belief of type Boolean that defines its security status which is set by default to true at the creation time of the agent. When the shared service between both agents is invoked transferring the signal and the time, the Plan *SecureDevicePlan* of the

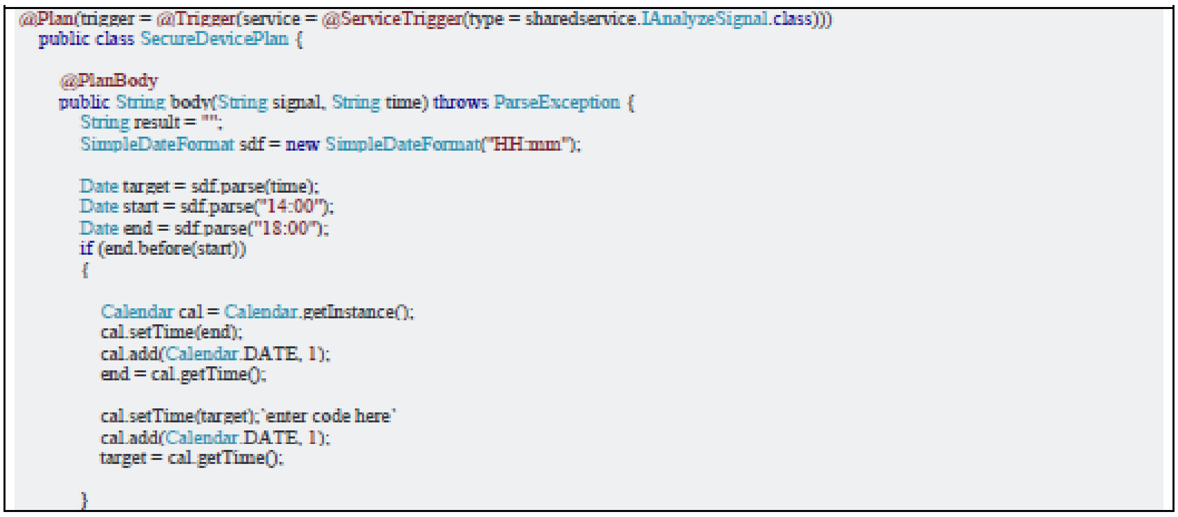
ReasoningBDI is triggered as defined in the header of the plan in Figure 36. The body of the plan is represented by method *body ( )* which takes the signal and time as parameters and returns a statement to the sensing agent, the caller of the service, indicating whether the device is safe or exposed to any threat. This output depends on checking a condition, which is performed by the agent that investigates if the current time (target) lies between a particular time range (start and end) or not. But first it tests if the end date came before the start date, then 24 hours are added to the end as well as the target date to form a proper time range occurring on the same day. If this is not the case, it checks immediately if the target date occurred after the start date and before the end date, which has four possible options divided to two groups. The first group belongs to the date which is within the specified range, while the other group is for dates out of the range and draws attention to any possible threat. The four options illustrated in Figure 37 consist of:

1) The target date is in range and the device has been already under control. Therefore, the reasoning agent outputs: “Device is still under control”.
2) The target date is in range but the device was not safe before. Hence, the agent sets *safe* to true and returns: “Device is now secured”.
3) The device has been already safe but the target date is not in range. So safe is set to false and the agent outputs: “Device is not secure anymore.” to warn the user against an existing abnormal behavior.
4) The device was not safe and also out of range. Hence, the agent returns: “Device is still not secured. Make an action!!” in order to draw the user’s attention to take an initiative with the purpose of securing the device.



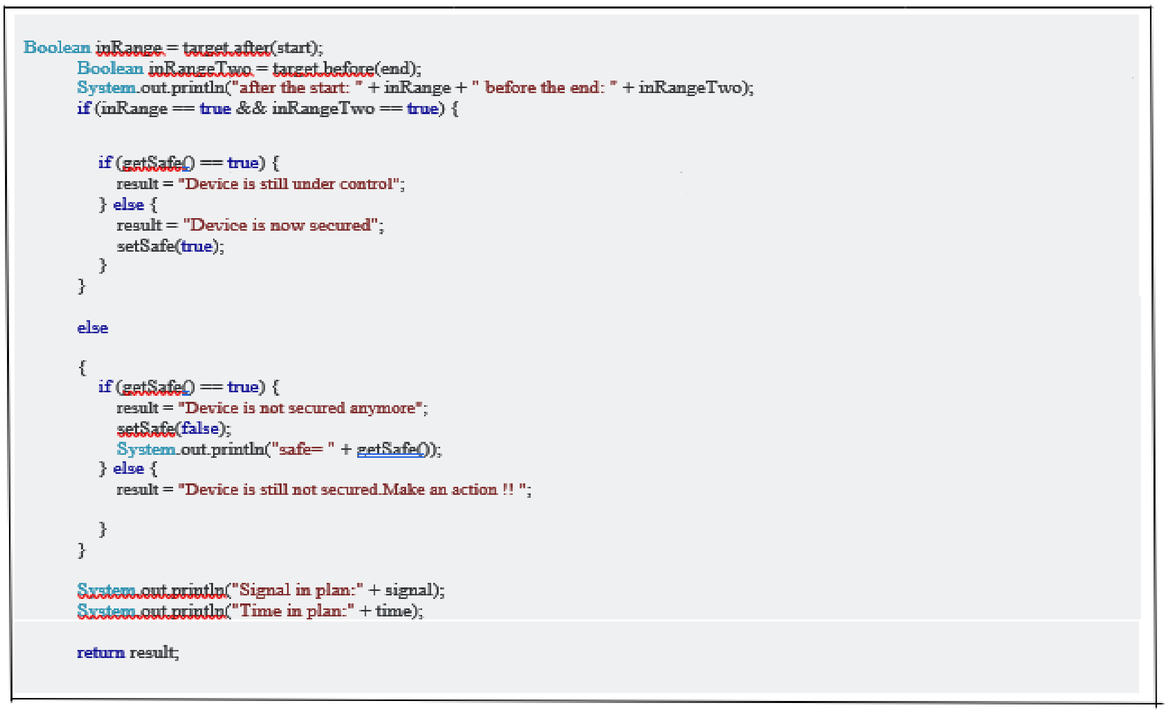
```
@Agent
@Service
@ProvidedServices(@ProvidedService(name = "analyzeservice", type = sharedservice.IAnalyzeSignal.class,
implementation = @Implementation(IBDIAgent.class)))

public class ReasoningBDI {
   @Belief
   protected Boolean safe;

   @AgentCreated
   public void init() {
      this.safe = true;
   }

   public Boolean getSafe() {
      return safe;
   }

   public void setSafe(Boolean safe) {
      this.safe = safe;
   }
```

Figure 35: Declaration of the ReasoningBDI



```
@Plan(trigger = @Trigger(service = @ServiceTrigger(type = sharedservice.IAnalyzeSignal.class)))
  public class SecureDevicePlan {

     @PlanBody
     public String body(String signal, String time) throws ParseException {
        String result = "";
        SimpleDateFormat sdf = new SimpleDateFormat("HH:mm");

        Date target = sdf.parse(time);
        Date start = sdf.parse("14:00");
        Date end = sdf.parse("18:00");
        if (end.before(start))
        {

           Calendar cal = Calendar.getInstance();
           cal.setTime(end);
           cal.add(Calendar.DATE, 1);
           end = cal.getTime();

           cal.setTime(target);`enter code here`
           cal.add(Calendar.DATE, 1);
           target = cal.getTime();

        }
```

Figure 36: Plan of the ReasoningBDI



```
Boolean inRange = target.after(start);
        Boolean inRangeTwo = target.before(end);
        System.out.println("after the start: " + inRange + " before the end: " + inRangeTwo);
        if (inRange == true && inRangeTwo == true) {

            if (getSafe() == true) {
                result = "Device is still under control";
            } else {
                result = "Device is now secured";
                setSafe(true);
            }
        }

        else

        {
            if (getSafe() == true) {
                result = "Device is not secured anymore";
                setSafe(false);
                System.out.println("safe= " + getSafe());
            } else {
                result = "Device is still not secured.Make an action !! ";

            }
        }

        System.out.println("Signal in plan:" + signal);
        System.out.println("Time in plan:" + time);

        return result;
```

Figure 37: The reasoning conditions of the ReasoningBDI

5.4 Output of running the application

Figure 38 depicts the communication process which takes place between both agents to make it clearer. The information flows from the sensing agent to the central agent, whereas the output is returned back in the reverse direction. Figure 39, 40 and 41 show the output which appears on the console of each agent when the process is completed. In the *CollectingBDI,* when a signal is received, the connection to the database is established, the data is inserted in it and the service is required. Eventually, the statement returned by the central reasoning agent is received and printed on the terminal. The sending process is performed exactly in the same way. Nonetheless, the signal is sent through the terminal as Figure 42 shows. Then it is read from the *SerialWriter* to the *CUL device's* port until it reaches the device. At the same time during the process, the signal and time which triggered the plan as well as the result of the checking condition are printed on the console of the central agent.

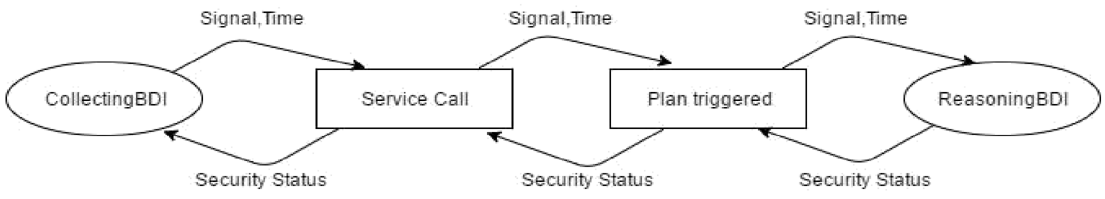
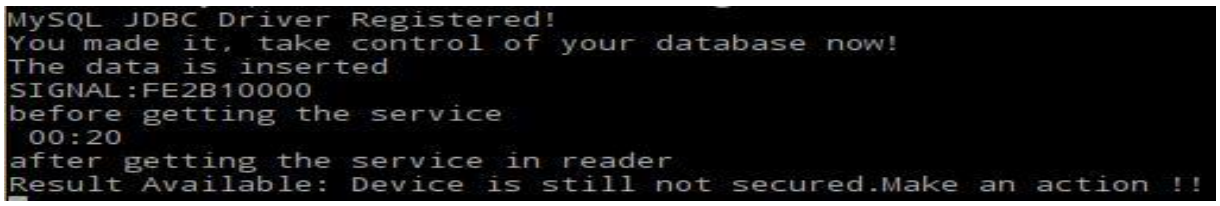
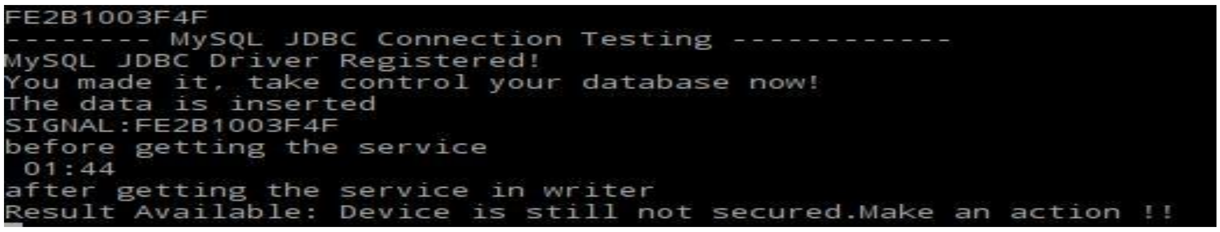


Figure 38: Communication process between agents



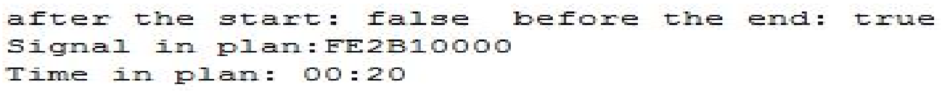
```
MySQL JDBC Driver Registered!
You made it, take control of your database now!
The data is inserted
SIGNAL:FE2B10000
before getting the service
 00:20
after getting the service in reader
Result Available: Device is still not secured.Make an action !!
```

Figure 39: Output of receiving a signal



```
FE2B1003F4F
-------- MySQL JDBC Connection Testing ------------
MySQL JDBC Driver Registered!
You made it, take control your database now!
The data is inserted
SIGNAL:FE2B1003F4F
before getting the service
 01:44
after getting the service in writer
Result Available: Device is still not secured.Make an action !!
```

Figure 40: Output of sending a signal



```
after the start: false  before the end: true
Signal in plan:FE2B10000
Time in plan: 00:20
```

Figure 41: Output console of the ReasoningBDI

5.5 Discussion

This article aims to assess the capabilities of the distributed intelligent agents in detecting anomalies in an IoT environment. The previous implementation makes it clear that it is feasible to deploy distributed intelligent agents on a network of IoT systems. According to the output presented in section 5.4, an intelligent agent can ensure the security of devices in the sense that they are only accessed during the time range specified by the authorized users. It is able to give a warning to the user that a suspicious event has occurred or that its device is secured and under control. This simple use case is facilitated by the reasoning capabilities provided by the intelligent agents, which enable them to keep track of the current security status of the device as well as compare the time at which the signal was transferred to the specified start and end times by the user.

The major focus was on setting up an environment representing all types of technologies including components that collect and transmit data as well as perform sensing and reasoning while minimizing the complexity resulting from using multiple agents. The project is able to implement a sensing component on resource-constrained hardware and to run a simple reasoning agent as a central service. This approach was presented by a simple use case to illustrate the conceptual idea behind anomaly detection in IoT networks. It has also proved that it is doable to build a network of distributed intelligent reasoning agents to serve this purpose.

Due to the limited scope of this article and the time constraints due to the need to integrate several independent technologies together, several aspects could not be covered within the scope of the project. First, the network consisted of only two BDI agents while it could have been expanded to deploy multiple agents enabling various types of reasoning mechanisms such as localized, centralized and collaborative. In addition, the built network adopted only the intelligence-based approach on the basis of the centralized reasoning principle. It should have also examined other different architectural structures of reasoning systems such as local as well as collaborative architectural paradigms to elaborate more on the advantages of distributing these agents. Distributed intelligent agents can either operate independently to reach individual rational decisions or interact with other agents to solve more complex problem sets, which are not recognized by individual acting agents.

The agents built here are the simplest form of BDI agents, having only a belief keeping track of the security of the device as well as a plan performing a simple reasoning task. Conceptually, the developed intelligent reasoning agent should have a goal indicating the desire the agent aims to achieve. Nonetheless, due to a development issue found in the backend system of the “Jadex” tool itself, utilized to develop these agents, the reasoning agent built in the use case was only a simplified version of a BDI agent. This limitation within the scope of this article prevented further investigation of the reasoning capabilities provided by the BDI model.

6 Conclusion

This thesis develops a methodology that agents can be utilized in an IoT environment. It also contributes in the sense of actually showing with a prototypical implementation that this works. The implementation presents a use case, which combines the deployment of low- powered devices that have attached sensors with the centralized agents' reasoning approach. Furthermore, IoT applications are facilitated by the emergence of those inexpensive, small yet powerful devices. Adding to the useful sensing capabilities of those small devices, they also bring all the necessary requirements together for hosting a reasoning agent deployed on a sensor network. This proves the feasibility of such an approach, which uses those different components and technologies combined with small and inexpensive devices. Moreover, the thesis demonstrates that the centralized distributed intelligent agents' approach is to a great extent capable of detecting anomalies in IoT as far as the simple use case proposed in the thesis is concerned.

References

- Agha, G. A. (1985). Actors: A model of concurrent computation in distributed systems (No. AI-TR-844). MASSACHUSETTS INST OF TECH CAMBRIDGE ARTIFICIAL INTELLIGENCE LAB.
- Ahmed, M., & Mahmood, A. N. (2015). Novel approach for network traffic pattern analysis using clustering-based collective anomaly detection. Annals of Data Science, 2(1), 111-130.
- Alghuried, A. (2017). A Model for Anomalies Detection in Internet of Things (IoT) Using Inverse Weight Clustering and Decision Tree.
- Amendola, S., Lodato, R., Manzari, S., Occhiuzzi, C., & Marrocco, G. (2014). RFID technology for IoTbased personal healthcare in smart spaces. IEEE Internet of things journal, 1(2), 144-152.
- Ashfaq, A. B., Ali, M. Q., & Khayam, S. A. (2011). Accuracy improving guidelines for network anomaly detection systems. Journal in computer virology, 7(1), 63-81.
- Atzori, L., Iera, A., & Morabito, G. (2010). The internet of things: A survey. Computer networks, 54(15), 2787-2805.
- Azzara, A., Bocchino, S., Pagano, P., Pellerano, G., & Petracca, M. (2013, September). Middleware solutions in WSN: The IoT oriented approach in the ICSI project. In Software, telecommunications and computer networks (SoftCOM), 2013 21st international conference on (pp. 1-6). IEEE.
- Babar, S., Stango, A., Prasad, N., Sen, J., & Prasad, R. (2011, February). Proposed embedded security framework for internet of things (iot). In Wireless Communication, Vehicular Technology, Information Theory and Aerospace & Electronic Systems Technology (Wireless VITAE), 2011 2nd International Conference on (pp. 1-5). IEEE.
- Bhuse, V., & Gupta, A. (2006). Anomaly intrusion detection in wireless sensor networks. Journal of High Speed Networks, 15(1), 33-51.
- Bhuyan, M. H., Bhattacharyya, D. K., & Kalita, J. K. (2014). Network anomaly detection: methods, systems and tools. Ieee communications surveys & tutorials, 16(1), 303-336.
- Bolton, R. J., & Hand, D. J. (2001). Unsupervised profiling methods for fraud detection. Credit Scoring and Credit Control VII, 235-255.
- Bolton, R. J., & Hand, D. J. (2002). Statistical fraud detection: A review. Statistical science, 235-249.
- Braubach, L., & Pokahr, A. (2012). Developing distributed systems with active components and Jadex. Scalable Computing: Practice and Experience, 13(2), 100-120.
- Braubach, L., Lamersdorf, W., & Pokahr, A. (2003). Jadex: Implementing a BDI-infrastructure for JADE agents. *EXP – in search of innovation*, 3(3),76–85.
- Braubach, L., Pokahr, A., & Lamersdorf, W. (2004, September). Jadex: A short overview. In Main Conference Net. ObjectDays (Vol. 2004, pp. 195-207).
- Braubach, L., Pokahr, A., & Lamersdorf, W. (2005). Jadex: A BDI-agent system combining middleware and reasoning. In Software agent-based applications, platforms and development kits (pp. 143-168). Birkhäuser Basel.
- Cai, X., Lyu, M. R., Wong, K. F., & Ko, R. (2000). Component-based software engineering: technologies, development frameworks, and quality assurance schemes. In Software Engineering Conference, 2000. APSEC 2000. Proceedings. Seventh Asia-Pacific (pp. 372-379). IEEE.
- Castanedo, F., Patricio, M. A., Garcia, J., & Molina, J. M. (2006, October). Extending surveillance systems capabilities using BDI cooperative sensor agents. In Proceedings of the 4th ACM international workshop on Video surveillance and sensor networks (pp. 131-138). ACM.
- Chandola, V., Banerjee, A., & Kumar, V. (2009). Anomaly detection: A survey. ACM computing surveys (CSUR), 41(3), 15.
- Clarke, D., Wrigstad, T., Östlund, J., & Johnsen, E. B. (2008, December). Minimal ownership for active objects. In Asian Symposium on Programming Languages and Systems (pp. 139-154). Springer Berlin Heidelberg.
- Coulouris, G., Dollimore, J., Kindberg, T., & Blair, G. (2012). Distributed systems (5th ed.). Boston: Addison-Wesley.
- de Souza, P. S. S., dos Santos Marques, W., Rossi, F. D., da Cunha Rodrigues, G., & Calheiros, R. N. (2017, January). Performance and accuracy trade-off analysis of techniques for anomaly detection in IoT sensors. In Information Networking (ICOIN), 2017 International Conference on (pp. 486-491). IEEE.
- Debar, H., Dacier, M., & Wespi, A. (1999). Towards a taxonomy of intrusion-detection systems. Computer Networks, 31(8), 805-822.
- Desnitsky, V. A., Kotenko, I. V., & Nogin, S. B. (2015, May). Detection of anomalies in data for monitoring of security components in the Internet of Things. In Soft Computing and Measurements (SCM), 2015 XVIII International Conference on (pp. 189-192). IEEE.
- Farooq, M. U., Waseem, M., Khairi, A., & Mazhar, S. (2015). A critical analysis on the security concerns of internet of things (IoT). International Journal of Computer Applications, 111(7).

- Garcia-Teodoro, P., Diaz-Verdejo, J., Maciá-Fernández, G., & Vázquez, E. (2009). Anomaly-based network intrusion detection: Techniques, systems and challenges. computers & security, 28(1), 18-28.
- Gubbi, J., Buyya, R., Marusic, S., & Palaniswami, M. (2013). Internet of Things (IoT): A vision, architectural elements, and future directions. Future Generation Computer Systems, 29(7), 1645-1660
- He, W., Yan, G., & Da Xu, L. (2014). Developing vehicular data cloud services in the IoT environment. IEEE Transactions on Industrial Informatics, 10(2), 1587-1595.
- Hodo, E., Bellekens, X., Hamilton, A., Dubouilh, P. L., Iorkyase, E., Tachtatzis, C., & Atkinson, R. (2016, May). Threat analysis of iot networks using artificial neural network intrusion detection system. In Networks, Computers and Communications (ISNCC), 2016 International Symposium on (pp. 1-6). IEEE.
- Howden, N., Rönnquist, R., Hodgson, A., & Lucas, A. (2001, May). JACK intelligent agents-summary of an agent infrastructure. In 5th International conference on autonomous agents.
- Jander, K., Braubach, L., & Pokahr, A. (2015). Extending the Communication Capabilities of Agents. Computing & Informatics, 34(1).
- Johnsen, E. B., & Owe, O. (2007). An asynchronous communication model for distributed concurrent objects. Software & Systems Modeling, 6(1), 39-58.
- Kaur, H., Singh, G., & Minhas, J. (2013). A review of machine learning based anomaly detection techniques. arXiv preprint arXiv:1307.7286.
- Kelly, S. D. T., Suryadevara, N. K., & Mukhopadhyay, S. C. (2013). Towards the implementation of IoT for environmental condition monitoring in homes. IEEE Sensors Journal, 13(10), 3846-3853.
- Kelm, A., Laußat, L., Meins-Becker, A., Platz, D., Khazaee, M. J., Costin, A. M., ... & Teizer, J. (2013). Mobile passive Radio Frequency Identification (RFID) portal for automated and rapid control of Personal Protective Equipment (PPE) on construction sites. Automation in Construction, 36, 38-52.
- Kemmerer, R., & Vigna, G. (2002). Intrusion detection: A brief history and overview. The Computer Journal, 35(4), 27-30. doi:10.1109/mc.2002.1012428
- Korecko, S., Herich, T., & Sobota, B. (2014, January). JBdiEmo—OCC model based emotional engine for Jadex BDI agent system. In Applied Machine Intelligence and Informatics (SAMI), 2014 IEEE 12th International Symposium on (pp. 299-304). IEEE.
- Kou, Y., Lu, C. T., Sirwongwattana, S., & Huang, Y. P. (2004). Survey of fraud detection techniques. In Networking, sensing and control, 2004 IEEE international conference on (Vol. 2, pp. 749-754). IEEE.
- Kumar, S. (1995). Classification and detection of computer intrusions (Doctoral dissertation, Purdue University).
- Lazarevic, A., Ertoz, L., Kumar, V., Ozgur, A., & Srivastava, J. (2003, May). A comparative study of anomaly detection schemes in network intrusion detection. In Proceedings of the 2003 SIAM International Conference on Data Mining (pp.25-36). Society for Industrial and Applied Mathematics.
- Liao, H. J., Lin, C. H. R., Lin, Y. C., & Tung, K. Y. (2013). Intrusion detection system: A comprehensive review. Journal of Network and Computer Applications, 36(1), 16-24.
- Luck, M., McBurney, P., Shehory, O., & Willmott, S. (2005). Agent Technology Roadmap: A Roadmap for Agent Based Computing. AgentLink III.
- Luo, H., Ci, S., Wu, D., Stergiou, N., & Siu, K. C. (2010). A remote markerless human gait tracking for e-healthcare based on content-aware wireless multimedia communications. IEEE Wireless Communications, 17(1).
- Ma, H. D. (2011). Internet of things: Objectives and scientific challenges. Journal of Computer science and Technology, 26(6), 919-924.
- Martin, C., Nasr, R., Hoersken, M., & Fuechtler, T. (2016, December). Automating Information Security assessments using intelligent software agents. In Privacy, Security and Trust (PST), 2016 14th Annual Conference on (pp. 736-744). IEEE.
- Minar, N., Gray, M., Roup, O., Krikorian, R., & Maes, P. (1999). Hive: Distributed agents for networking things. In Agent Systems and Applications, 1999 and Third International Symposium on Mobile Agents. Proceedings. First International Symposium on (pp. 118-129). IEEE.
- Mishra, K. S., & Tripathi, A. K. (2014). Some Issues, Challenges and Problems of Distributed Software System. International Journal of Computer Science and Information Technologies. Varanasi, India, 7, 3.
- Nadiminti, K., De Assunçao, M. D., & Buyya, R. (2006). Distributed systems and recent innovations: Challenges and benefits. InfoNet Magazine, 16(3), 1-5.
- Nguyen, N. T., & Jain, L. C. (2009). Intelligent agents in the evolution of Web and applications. Berlin: Springer.
- Omar, S., Ngadi, A., & Jebur, H. H. (2013). Machine learning techniques for anomaly detection: an overview. International Journal of Computer Applications, 79(2).
- O'Reilly, C., Gluhak, A., Imran, M. A., & Rajasegarar, S. (2014). Anomaly detection in wireless sensor networks in a non-stationary environment. IEEE Communications Surveys & Tutorials, 16(3), 1413-1432.
- Patcha, A., & Park, J. M. (2007). An overview of anomaly detection techniques: Existing solutions and latest technological trends. Computer networks, 51(12), 3448-3470.
- Phua, C., Lee, V., Smith, K., & Gayler, R. (2010). A comprehensive survey of data mining-based fraud detection research. arXiv preprint arXiv:1009.6119.

- Piunti, M., Ricci, A., Braubach, L., & Pokahr, A. (2008, December). Goal-directed interactions in artifact-based mas: Jadex agents playing in cartago environments. In Proceedings of the 2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology-Volume 02 (pp. 207-213). IEEE Computer Society.
- Pokahr, A., Braubach, L., & Jander, K. (2010, September). Unifying agent and component concepts. In German Conference on Multiagent System Technologies (pp. 100-112). Springer Berlin Heidelberg.
- Pokahr, A., Braubach, L., & Jander, K. (2013a). Jadex: A Generic Programming Model and One-Stop-Shop Middleware for Distributed Systems. Praxis der Informationsverarbeitung und Kommunikation, 36(2), 149-150.
- Pokahr, A., Braubach, L., & Jander, K. (2013b). The jadex project: Programming model. In Multiagent Systems and Applications (pp. 21-53). Springer Berlin Heidelberg.
- Pokahr, A., Braubach, L., & Lamersdorf, W. (2005). Jadex: A BDI reasoning engine.
  Multi-agent programming, 149-174.
- Rajasegarar, S., Leckie, C., & Palaniswami, M. (2008). Anomaly detection in wireless sensor networks. IEEE Wireless Communications, 15(4).
- Rajasegarar, S., Leckie, C., Palaniswami, M., & Bezdek, J. C. (2007, June). Quarter sphere based distributed anomaly detection in wireless sensor networks. In Communications, 2007. ICC'07. IEEE International Conference on (pp. 3864-3869). IEEE.
- Rajasegarar, S., Leckie, C., Palaniswami, M., & Bezdek, J. C. (2006, October). Distributed anomaly detection in wireless sensor networks. In Communication systems, 2006. ICCS 2006. 10th IEEE Singapore International Conference on (pp. 1-5). IEEE.
- Raut, A. S., & Singh, K. R. (2014). Anomaly based intrusion detection-a review. International Journal on Network Security, 5(3), 7.
- Sekar, R., Gupta, A., Frullo, J., Shanbhag, T., Tiwari, A., Yang, H., & Zhou, S. (2002, November). Specification-based anomaly detection: a new approach for detecting network intrusions. In Proceedings of the 9th ACM conference on Computer and communications security (pp. 265-274). ACM.
- Srivastava, A., Kundu, A., Sural, S., & Majumdar, A. (2008). Credit card fraud detection using hidden Markov model. IEEE Transactions on dependable and secure computing, 5(1), 37-48.
- Stallings, W., & Brown, L. (2012). *Computer Security: principles and practice* (2nd ed.). London: Pearson.
- Tanenbaum, A. S., & Van Steen, M. (2007). Distributed systems: principles and paradigms. Prentice-Hall.
- Washizaki, H., Yamamoto, H., & Fukazawa, Y. (2003, September). A metrics suite for measuring reusability of software components. In Software Metrics Symposium, 2003. Proceedings. Ninth International (pp. 211-223). IEEE.
- Whitman, M. E. (2003). Enemy at the gate: threats to information security. Communications of the ACM, 46(8), 91-95.
- Winikoff, M. (2006). JACK TM intelligent agents: An industrial strength platform.
  Multiagent Systems, Artificial Societies, and Simulated Organizations, vol. 15.
- Winikoff, M., Padgham, L., & Harland, J. (2001). Simplifying the development of intelligent agents. AI 2001: Advances in Artificial Intelligence, 557-568.
- Witzke, E. L. (2016, October). Computer network security: Then and now. In Security Technology (ICCST), 2016 IEEE International Carnahan Conference on (pp. 1-7). IEEE.
- Wooldridge, M. J. (2009). An introduction to multiagent systems. Chichester: JOHN WILEY & SONS, LTD.
- Xia, F., Yang, L. T., Wang, L., & Vinel, A. (2012). Internet of things. International Journal of Communication Systems, 25(9).
- Xie, M., Han, S., Tian, B., & Parvin, S. (2011). Anomaly detection in wireless sensor networks: A survey. Journal of Network and Computer Applications, 34(4), 1302-1325.
- Yu, E. (2001). Agent orientation as a modelling paradigm. Wirtschaftsinformatik, 43(2), 123-132.
- Zhang, Y., Meratnia, N., & Havinga, P. (2010). Outlier detection techniques for wireless sensor networks: A survey. IEEE Communications Surveys & Tutorials, 12(2), 159-170.
- Zhang, Z. K., Cho, M. C. Y., & Shieh, S. (2015, April). Emerging security threats and countermeasures in IoT. In Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security (pp. 1-6). ACM.