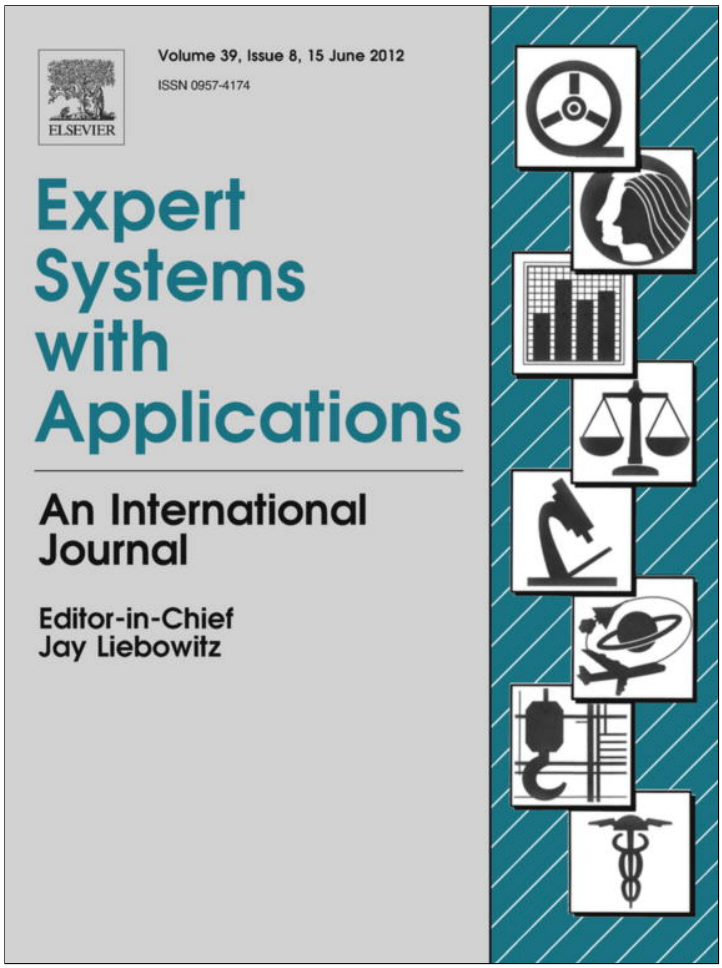

Expert Systems with Applications 39 (2012) 7536–7545

Contents lists available at SciVerse ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

Improving the security level of the FUSION@ multi-agent architecture

Cristian I. Pinzón [a], Juan F. De Paz [b], Dante I. Tapia [b], Javier Bajo [c,*], Juan M. Corchado [b]

[a] Universidad Tecnológica de Panamá, Campus Metropolitano «Dr. Víctor Levi Sasso», Vía Ricardo J. Alfaro, P.O. Box 0819-07289 Panamá, Panama
[b] Departamento de Informática y Automática, Universidad de Salamanca, Plaza de la Merced, s/n, 37008 Salamanca, Spain
[c] Facultad de Informática, Universidad Pontificia de Salamanca, Compañía 5, 37002 Salamanca, Spain

ARTICLE INFO

Keywords:
Service-oriented architectures
Multi-agent systems
Security
Case-based reasoning

ABSTRACT

The use of architectures based on services and multi-agent systems has become an increasingly important part of the solution set used for the development of distributed systems. Nevertheless, these models pose a variety of problems with regards to security. This article presents the Adaptive Intrusion Detection Multi-agent System (AIDeMaS), a mechanism that has been designed to detect and block malicious SOAP messages within distributed systems built by service based architectures. AIDeMaS has been implemented as part of FUSION@, a multi-agent architecture that facilitates the integration of distributed services and applications to optimize the construction of highly-dynamic multi-agent systems. One of the main features of AIDeMaS is that is employs case-based reasoning mechanisms, which provide it with great learning and adaptation capabilities that can be used for classifying SOAP messages. This research presents a case study that uses the ALZ-MAS system, a multi-agent system built around FUSION@, in order to confirm the effectiveness of AIDeMaS. The preliminary results are presented in this paper.

1. Introduction

The last several years have witnessed a rapid growth in service oriented architecture systems in such fields as e-commerce (He & Leung, 2002), the government (Elsas, 2003), e-business (Chung, 2008), education (Vossen & Westerkamp, 2003), and health (Hori & Ohashi, 2005), among others. Service oriented architectures provide benefits for distributed data handling, but their implementation has incited a surge in security related problems.

The security problems inherent in these environments are rooted in the use of new and open standards that were developed without taking security needs into account (Pulier & Taylor, 2005). Different security specifications subsequently appeared for Web Services, such as WS-Security (Nadalin, Kaler, Monzillo, & Hallam-Baker, 2006), WS-SecurityPolicy (Della-Libera et al., 2005), WS-Trust (Anderson et al., 2004a), WS-SecureConversation (Anderson et al., 2004b), etc. However, all of these specifications focus on the aspects of message integrity and confidentiality, and user authentication and authorization. None of them directly addresses a way to ensure the availability of web services that can deal with the risk of external attacks. A common type of attack at the service-oriented level of an architecture is the denial of service (DoS) attack. The high level of occurrence of DoS attacks is due to the fact that XML-encoded SOAP messages, which are used for communication, must be parsed in the server. This opens the possibility of an attack if the messages themselves are not well structured or if they include some type of malicious code. Resources available in the server (memory and CPU cycles) of the provider can be drastically reduced or exhausted while a malicious SOAP message is being parsed. These attacks are critical in that they compromise the availability of services and resources (Gruschka & Luttenberger, 2006), rendering them inoperative for legitimate users.

This paper presents the Adaptive Intrusion Detection Multi-agent System (AIDeMaS), a novel mechanism for detecting and preventing DoS attacks that has been integrated into FUSION@ (Flexible Users and ServIces Oriented multi-ageNt Architecture) (Corchado, Tapia, & Bajo, 2009). FUSION@ proposes a new and easier method to develop distributed intelligent ubiquitous systems, where applications and services can communicate in a distributed way with intelligent agents, even from mobile devices, regardless of time and location restrictions. It is important to note that FUSION@ did already include a security component within its structure consisting of an agent specialized in evaluating all incoming messages and determining their reliability. However, the security method employed by this agent is limited in scope, as the agent lacks the ability to evolve when dealing with either new and more complex attacks or a variation in known attack patterns, thus making available services vulnerable to attack. Given the limitations that exist in the current security system in FUSION@, the AIDeMaS system is now proposed as a model for the detection and prevention of attacks. AIDeMaS has evolved from previous research in SQL injection attacks (Bajo, Corchado, Pinzón,

* Corresponding author. Tel.: +34 923 277100; fax: +34 923 277101.
E-mail addresses: cristian_ivanp@usal.es (C.I. Pinzón), fcofds@usal.es (J.F. De Paz), dantetapia@usal.es (D.I. Tapia), jbajope@upsa.es (J. Bajo), corchado@usal.es (J.M. Corchado).

0957-4174/$ - see front matter © 2012 Elsevier Ltd. All rights reserved.
doi:10.1016/j.eswa.2012.01.127

Paz, & Pérez-Lancho, 2008; Pinzón, De Paz, & Bajo, 2008) where a multi-agent architecture SQLMAS was developed. AIDeMaS is based on a group of agents specially designed to work together intelligently and adaptively to solve the problem of the reliability of SOAP messages sent in service requests. The core of AIDeMaS is a classification mechanism that incorporates a two-phase strategy to classify SOAP messages. The first phase applies an initial filter for detecting simple attacks without requiring an excessive amount of resources. The second phase involves a more complex process that ends up using a significantly higher amount of resources. In this way, a two-phase strategy improves the overall response time of the classification mechanism. Each of the phases incorporates an intelligent agent that integrates a CBR engine with advanced classification capabilities.

The idea of a CBR mechanism is to exploit the experience gained from similar problems in the past and then adapt successful solutions to the current problem. The CBR engine initiates what is known as the CBR cycle, which is comprised of four stages (Aamodt & Plaza, 1994). The approach presented in this paper proposes a classifier agent for the first phase (Classifier F1 Agent) that incorporates a classification strategy based on a Naives Bayes classifier, and a classifier agent for the second phase (Classifier F2 Agent) that incorporates a neural network. Each of these classification strategies is incorporated into the respective re-use stage of the CBR cycle integrated into the corresponding agent. As a result, the system can learn and adapt to the attacks and the changes in the techniques used in the attacks. The model proposed in this study is innovative, since proposes a new perspective to address the DoS attacks problem in service oriented architectures. Additionally, the specific use of FUSION@ allows for a notable improvement in the security system initially presented. The model is not intended to replace the existing security solutions, but to be established as an additional layer to existing security measures, providing advanced knowledge that can support the decision making process in those cases where the security and availability of the service information are at risk.

The remainder of the paper is structured as follows: Section 2 presents a general description of the FUSION@ architecture. Section 3 focuses on the limitations of the current security problems found in FUSION@; Section 4 presents the new security mechanism in detail. Section 5 describes a case study using an application that incorporates FUSION@. Finally, the final results and conclusion are presented in Section 6.

2. FUSION@ description

FUSION@ (Corchado et al., 2009) is based on agents with such characteristics as autonomy, reasoning, reactivity, social abilities, pro-activity, mobility, organization, etc. that allow them to cover several needs for distributed environments. FUSION@ combines a services-oriented approach with intelligent agents (Brena, Aguirre, Carlos, Ramírez, & Leonardo, 2007) to obtain an innovative architecture that facilitates ubiquitous computation and communication, and high levels of human-system-environment interaction. It also provides an advanced flexibility and customization to easily add, modify or remove applications or services on demand, regardless of the programming language. Finally, FUSION@ provides a flexible distribution of resources and facilitates the inclusion of new functionalities in highly dynamic environments. It also provides the systems with a greater ability to recover from errors and better flexibility to change their behaviour at execution time.

Because the architecture acts as an interpreter, users can run applications and services programmed in virtually any language, but they must follow a communication protocol that all applications and services must incorporate. Another important functionality, which is the result of the agents' inherent capabilities, is the ability of the systems developed to utilize reasoning mechanisms or learning techniques to handle services and applications according to context characteristics, which can change dynamically over time. Agents, applications and services can communicate in a distributed way, even from mobile devices. This makes it possible to use resources regardless of their location. It also allows for agents, applications, services or devices to start or stop separately, without affecting the rest of resources, which endows the system with an elevated adaptability and capacity for error recovery.

FUSION@ framework defines four basic blocks: (a) Applications represent all the programs that can be used to exploit the system functionalities. They can be executed locally or remotely. (b) Services represent the activities that the architecture offers. (c) Agents Platform is the core of the architecture and integrates a set of agents, each one with special characteristics and behaviours. In FUSION@, services are managed and coordinated by deliberative BDI agents with distributed computation and coordination abilities. (d) Communication Protocol allows applications and services to communicate directly with the agents platform. The protocol is completely open and independent of any programming language, facilitating ubiquitous communication capabilities. This protocol is based on SOAP specification to capture all messages between the platform and the services and applications (Cerami, 2002). The communication among agents in the platform follows the FIPA Agent Communication Language (ACL) specification.

Fig. 1 shows the basic schema of FUSION@, where all requests and responses are handled by the agents in the platform. The agents analyze all requests and invoke the specified services either locally or remotely. Services process the requests and execute the specified tasks. Then, services send back a response with the result of the specific task.

FUSION@ does not include a service discovery mechanism, so applications must use only the services listed in the platform. In addition, all communication is handled by the platform, so there is no way to interact directly between applications and services. Moreover, the platform makes use of deliberative agents to select the optimal option for performing a task, so users do not need to find and specify the service to be invoked by the application. These features have been introduced in FUSION@ to create secure communication between applications and services. They also facilitate the inclusion of new services that users can use regarding their location and application.

FUSION@ is a modular multi-agent architecture, where services and applications are managed and controlled by deliberative BDI (Belief, Desire, Intention) agents (Pokahr, Braubach, & Lamersdorf, 2003; Bratman, Israel, & Pollack, 1988; Jennings & Wooldridge, 1995). Deliberative BDI agents are able to cooperate, propose solutions on very dynamic environments, and face real problems, even when they have a limited description of the problem and few resources available. There are different kinds of agents in the architecture, each one with specific roles, capabilities and characteristics. This fact facilitates the flexibility of the architecture in incorporating new agents. However, there are pre-defined agents which provide the basic functionalities of the architecture:

- *CommApp agent*. This agent is responsible for all communication between applications and the platform. Applications send XML messages to the agent requesting a service, then the agent creates a new thread to start communication by using sockets. The agent sends all requests to the Admin Agent which processes the request. The socket remains open until a response to the specific request is sent back to the application using another XML message. All messages are sent to Security Agent for their structure and syntax to be analyzed.

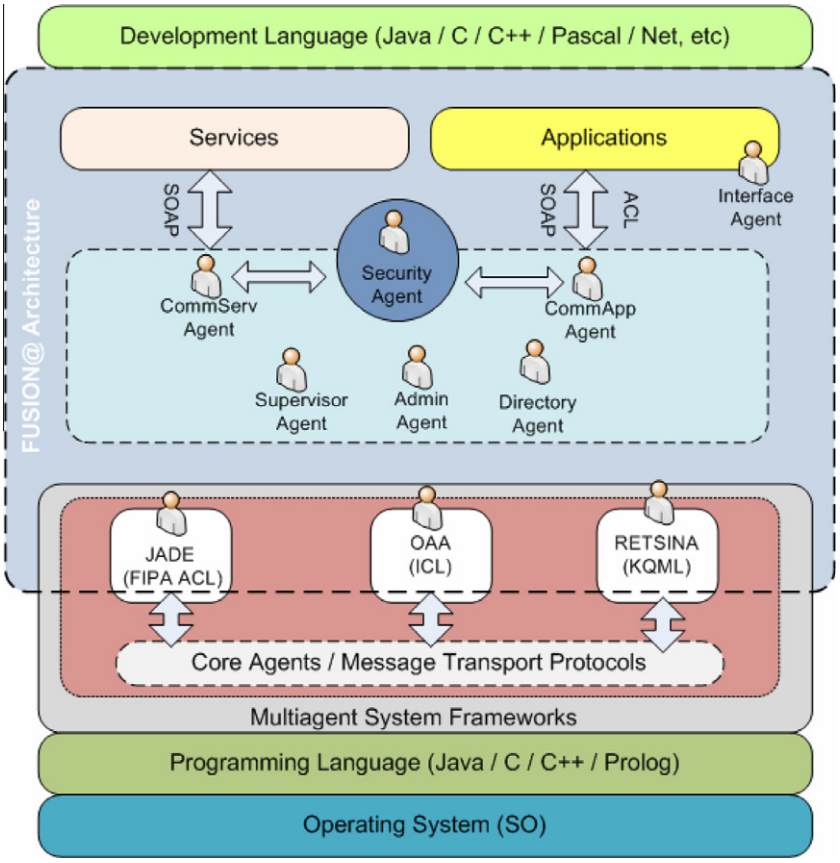

Fig. 1. FUSION@ basic schema.

- *CommServ agent*. Responsible for all communications between services and the platform. Admin Agent signals to CommServ Agent which service must be invoked. Then, CommServ Agent creates a new thread with its respective socket and sends an XML message to the service. The socket remains open until the service sends back a response. All messages are sent to Security Agent for their structure and syntax to be analyzed. This agent also periodically checks the status of all services to know if they are idle, busy, or crashed.
- *Directory agent*. Manages the list of services that can be used by the system. For security reasons (Lauro & Gian Luca, 2007), the list of services is static and can only be modified manually; however, services can be added, erased or modified dynamically. The list contains the information of all trusted available services. The name and description of the service, parameters required, and the IP address of the computer where the service is running are some of the information stored in the list of services. However, there is dynamic information that is constantly being modified: the service performance (average time to respond to requests), the number of executions, and the quality of the service. This last bit of data is very important for managing services.
- *Supervisor agent*. This agent supervises the correct functioning of the other agents in the system. Supervisor Agent periodically verifies the status of all agents registered in the architecture by sending ping messages. If there is no response, the Supervisor agent kills the agent and creates another instance of that agent.
- *Security agent*. This agent analyzes the structure and syntax of all incoming and outgoing XML messages. If a message is not correct, the Security Agent informs the corresponding agent (CommApp or CommServ) that the message cannot be delivered. This agent also directs the problem to the Directory Agent, which modifies the QoS of the service where the message was sent.
- *Admin agent*. Decides which agent must be called by taking the QoS and user preferences into account. Users can explicitly invoke a service, or can let the Admin Agent decide which service is best to accomplish the requested task. If there are several services that can resolve the task requested by an application, the agent selects the optimal choice. An optimal choice has higher QoS and better performance. Admin Agent has a routing list to manage messages from all applications and services. This agent also checks if services are working properly to ensure that QoS is always current.
- *Interface agent*. This kind of agent was designed to be embedded in user applications. Interface agents communicate directly with the agents in FUSION@ so there is no need to employ the communication protocol, rather the FIPA ACL specification. The requests are sent directly to the Security Agent, which analyzes the requests and sends them to the Admin Agent. The rest of the process follows the same guidelines for calling any service.

FUSION@ is an open architecture that allows developers to modify the structure of the agents previously described, so that agents are not defined in a static manner. Developers can add new agent types or extend the existing ones to conform to their projects needs. However, most of the agents' functionalities should be modelled as services, releasing them from tasks that could be performed by services.

Services represent all functionalities that the architecture offers to users and uses itself. To add a new service, it is necessary to

manually store its information into the directory list managed by the Directory Agent. Then, CommServ Agent sends a ping message to the service. The service responds to the ping message and the service is added to the platform. A service can be virtually any program that performs a specific task and shares its resources with the platform. These programs can provide methods to access data bases, manage connections, analyze data, get information from external devices (e.g., sensors, readers, screens, etc.), publish information, or even make use of other services. Developers are free to use any programming language. The only requirement is that they must follow the communication protocol based on transactions of XML (SOAP) messages.

The following section review the limitations of the current security issue configured in FUSION@.

3. Security problems in FUSION@

Security is considered an important element within the FUSION@ architecture because of the significance of the information that is exchanged at the service level. As a result FUSION@ incorporates a security mechanism that validates all incoming messages dealing with service requests. The initial strategy of FUSION@ is based on incorporating a "Security" agent, which is responsible for carrying out security-related tasks within the architecture. The strategy is centered on the role of the Security agent and its attempt to protect services that are facing potential attacks hidden within service requests. When a service request (embedded within a SOAP message) is sent through the available application interfaces, the CommApp agent receives the SOAP message and resends it to the Security agent for evaluation. The Security agent carries out a structure and content analysis of the SOAP message and determines its reliability.

The security mechanism contained within the Security agent is simple and efficient for known attacks, although it presents a series of limitations when it comes to protecting the architecture and services during a more complex attack. The basic analyzing mechanism evaluates the structure and content of the messages so that messages containing certain inconsistencies are rejected. The main problem with this security mechanism is rooted in the fact that the majority of the attacks made against service based environments use complex techniques that are difficult to detect with a simple XML code review found in the SOAP message.

One example of a complex attack directed at service based environments is the denial of service attack. DoS attacks may occur because XML messages must be parsed in the server, which opens the possibility of an attack if the messages themselves are not well structured or if they include some type of malicious code. Resources available in the server (memory and CPU cycles) of the provider can be drastically reduced or exhausted while a malicious SOAP message is being parsed. This attack is successfully carried out when it manages to severely compromise legitimate user access to services and resources. DoS Attacks usually occur when the SOAP message either comes from a malicious user or is intercepted during its transmission by a malicious node that introduces different kinds of attacks. The following list contains descriptions of some known types of attacks that can result in a DoS attack, as noted in Loh, Yau, Wong, and Ho (2006), Yee, Shin, and Rao (2007) and Jensen, Gruschka, Herkenhoner, and Luttenberger (2007).

- *Oversize payload:* Reduces or eliminates the availability of a service when a message with a large payload is parsed within the server.
- *Coercive parsing:* An XML parser can analyze a complex format and lead to an attack because the memory and processing resources are being used up.
- *Injection XML:* The structure of a XML document is modified with a malicious code.
- *SOAP header attack:* Some SOAP message headers are overwritten while they are passing through different nodes before arriving at their destination.
- *Replay attack:* Sent messages are completely valid, but they are sent en masse over short periods of time in order to overload the service.

Given the existence of complicated techniques such as DoS, the level of protection provided by the Security agent within the FUSION@ architecture can be considered somewhat weak and lacking in adaptation capability. Because the attack techniques tend to evolve quickly, it is likely that the Security agent's strategy is incapable of automatically adapting to the changes in attack patterns. As a result, the available services within the architecture can be affected by some type of attack, remaining blocked by the applications and the users who request them.

In summary, the initial security mechanism incorporated within FUSION@ presents the following limitations:

- It utilizes a mechanism that focuses on evaluating SOAP messages, which can bottleneck response time during an instance of high service requests and negatively affect the architecture's performance.
- The security mechanism strategy is limited with regards to analyzing the structure and content of SOAP messages. This strategy can only detect and block a limited number of known attacks, and cannot handle attacks that are more complicated in nature.
- Finally, the security mechanism is incapable of adapting to new attack patterns. This limitation prevents the security mechanism from confronting new attacks or fast-paced changes in known attack patterns.

4. Improved the security role in FUSION@ with AIDeMaS

The security feature in the FUSION@ architecture has been regarded as an important component since its initial design. However, the initial security mechanism incorporated in FUSION@ presents a series of limitations when facing certain types of more complicated attacks. Given these limitations and the importance of protecting information at the architecture level, the AIDeMaS system was proposed as an extension of the existing security mechanism.

One very important limitation of the initial security mechanism is the inability of the Security agent to adapt to the quickly changing attack patterns or to detect unknown attacks, which can severely compromise its ability to detect and block attacks. In order to overcome these limitations, AIDeMaS includes adaptation and learning capabilities based on novel algorithms. Additionally, the strategy used for carrying out tasks that involve the evaluation of SOAP messages within AIDeMaS has been proposed from a distributed perspective. The result is an original security mechanism that is more robust, dynamic and flexible than what was initially configured in the FUSION@ architecture. AIDeMaS is based on the incorporation of a new security block composed of a set of agents with special capabilities. The new proposed mechanism is based on our previous research in SQL injection attacks (Bajo et al., 2008; Pinzón et al., 2008) which developed a multi-agent SQLMAS architecture. In this way, some resources are reused and the knowledge acquired from previous work is adapted in order to provide an evolution of the mechanism proposed.

Fig. 2 presents the integration of AIDeMas in the FUSION@ architecture. As shown in Fig. 2, AIDeMaS is comprised of the

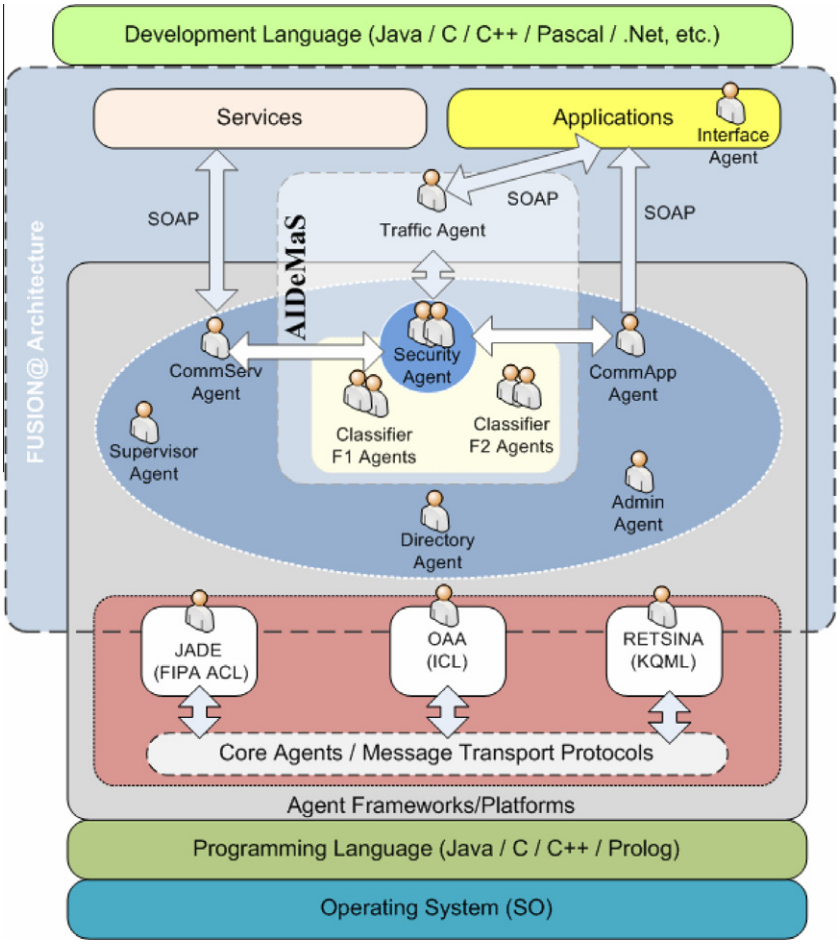

Fig. 2. FUSION@ basic schema with AIDeMaS incorporated.

Security and Admin agent, both of which were already included in the FUSION@ architecture, as well as the Traffic agent and two others that will function as classifiers: Classifier agent F1 and Classifier agent F2. AIDeMaS was designed as a two phased classification mechanism for classifying SOAP messages, as explained by Pinzón et al. (2008) and Bajo et al. (2008). The first phase applies the initial filter for detecting simple attacks without requiring an excessive amount of resources. The second phase involves a more complex process which ends up using a significantly higher amount of resources. This two-phased strategy improves the overall response time of the classification mechanism, facilitating a quick classification of any incoming SOAP messages that were thought to contain significant features during the first phase. The second phase is executed only for those SOAP messages with complex characteristics identified as suspicious during the first phase and requiring a more detailed evaluation. Each of the phases incorporates a CBR-BDI (Laza, Pavón, & Corchado, 2003) agent with reasoning, learning and adaptation capabilities.

The following section provides a detailed description of the characteristics and tasks related to each of the agents that constitute the AIDeMaS platform.

- *Traffic agent:* This agent has a type of sensor feature that allows it to identify and capture SOAP messages that have been sent from external applications and that request a particular type of service. The agent captures the messages and redirects them to the Security agent for evaluation.
- *Security agent:* This agent carries out tasks similar to those assigned with the original FUSION@ security mechanism. The agent is in charge of receiving SOAP message that contain service requests. It performs a quick analysis of the message, and the data obtained are sent to the agent at the first phase of the classification mechanism. With cases that are considered suspicious, the Security agent submits the XML message to a more comprehensive syntactic analysis in order to obtain the necessary data for carrying out the second phase of the classification mechanism. There can be more than one Security agent, depending on the amount of workload.
- *Classifier F1 agent:* This is one of the key agents in the classification process. These agents initiate a classification by incorporating a CBR engine that in turn incorporates a Naives Bayes strategy in the re-use phase. The main goal of this initial phase is to carry out an effective classification, but without requiring an excessive amount of resources. There can be more than one Classifier F1 agent depending on the amount of workload.
- *Classifier F2 agent:* This agent completes the classification mechanism. In order to initiate this phase, it is necessary to have previously started a syntactic analysis on the SOAP message to extract the required data. Once the data have been extracted from the message, a CBR mechanism is initiated by using a Multilayer Perceptron (MLP) neural network in the re-use phase. There can be more than one Classifier F2 agent, depending on the amount of workload.
- *Admin agent:* In addition to the functions already mentioned in the FUSION@ architecture, this agent is responsible for overseeing the correct functioning of the classification process and for coordinating the distribution of tasks.

4.1. First phase of the mechanism of classification – classifier F1

The main goal of this initial phase is to carry out an effective classification, but without requiring an excessive amount of resources and time. The initial phase incorporates a Naive Bayes classifier, mainly due to its low computational costs. As a CBR strategy is used, it is necessary to define the case structure used by the Classifier F1 agents. The fields of the case are obtained from the headers of the packages of the HTTP/TCP-IP transport protocol. Table 1 shows the fields taken into consideration to describe the problem.

4.1.1. Retrieve

During this stage, those cases associated with the requested web service are recovered from the memory of cases. It is necessary to recover the cases for the service and the subnet mask:

$$c_{.im} = f_s(C) = \{c_{j.im} \in C / c_{j.i} = c_{n+1.i}, c_{j.m} = c_{n+1.m}\} \quad (1)$$

where $c_{j.x}$ represents the case j and x one of the properties shown in Tables 1 and 2. In (1), $c_{j.i}$ represents the case j, service i; $c_{j.m}$ service j, mask m. f_s represents the function used to carry out the cases selection. If $c_{.im} = \phi$ or $\#c_{.im}$ the number of elements of the set is better than a threshold, 25 by default, then the cases are selected taking only into account the service.

4.1.2. Reuse

Once the cases in the $C_{.im}$ memory have been recovered, the Naive Bayes (Duda & Hart, 1973) classification is applied in order to estimate the probability that the new case will belong to classes a, g in the classification of requests. In order to carry out this task, various types of data were considered, since it is necessary to work with either continuous variables or a many different categories, meaning that the Bayes classifier cannot be applied as originally defined. If it had been, the final probability for each of the classes would be zero, due to the fact that some of the variables have a variety of different values.

Table 1
Problem description first phase – classifier F1 agent.

Fields	Type	Variable
IDService	Int	i
Subnet mask	String	m
SizeMessage	Int	s
NTimeRouting	Int	n
LengthSOAPAction	Int	l
TFMessageSent	Int	w

Table 2
Case description second phase – classifier F2 agent.

Fields	Type	Variable
IDService	Int	i
MaskSubnet	String	m
SizeMessage	Int	s
NTimeRouting	Int	n
LengthSOAPAction	Int	l
MustUnderstandTrue	Boolean	u
NumberHeaderBlock	Int	h
NElementsBody	Int	b
NestingDepthElements	Int	d
NXMLTagRepeated	Int	t
NLeafNodesBody	Int	f
NAttributesDeclared	Int	a
CPUTimeParsing	Int	c
SizeKbMemoryParser	Int	k

$$P_a = P(X = a) \prod_{i=1}^{n} \begin{cases} P(A_i = a_i | X = a) & a_i discrete \\ P(A_i < a_i | X = a) & a_i continue \end{cases} \quad (2)$$

$$P_l = P(C = l) \prod_{i=1}^{n} \begin{cases} P(A_i = a_i | c = g) & a_i discrete \\ P(A_i > a_i | c = g) & a_i continue \end{cases} \quad (3)$$

where $X = \{a, l\}$ a and g represent attack and legal and $a_i \in \{s, n, l, w, x^p, x^r\}$. The attributes are defined in Table 1. n represents the number of attributes of the case.

4.1.3. Revise

The revise phase considers the different probabilities calculated during the previous phase. Depending on the probability of whether the requests will be attack or suspicious, the results and the corresponding classification are examined. This process is carried out in the following manner:

$$\text{If } \frac{P_a}{P_l} > \mu_1 \rightarrow \text{attack}$$

$$\text{If } \frac{P_l}{P_a} > \mu_1 \rightarrow \text{legal}$$

$$\text{Otherwise} \rightarrow \text{suspicious}$$

If the classification is determined to be suspicious, the second phase of the classification mechanism is initiated.

4.1.4. Retain

Ultimately, a case is only stored if the probability P_a o P_l, is less than a predetermined threshold. This is because the high probability of attack or legal means that the case is similar to other cases previously stored in the case base. This allows the growth of the case memory to be kept at a minimum.

4.2. Second phase of the mechanism of classification – classifier F2 agent

The second phase of the classification mechanism is carried out by the Classifier F2 agents. Because they are CBR-BDI agents, it is necessary to provide a case description. The fields are extracted from the SOAP message and provide the case description for the Classifier F2 agents. Table 2 presents the fields used in describing the problem for the CBR in this layer.

Applying the nomenclature shown in the table above, each case description is given by the following tuple:

$$c = (i, m, s, n, l, u, h, b, d, t, f, a, c, k, P/c_{.im}, x^p, x^r) \quad (4)$$

For each incoming message received by the agent that requires classification, we will consider both the class that the agent predicts and the class to which the message actually belongs. x^p represents the class predicted by the Classifier F2 agents belonging to the group. $x^p \in X = \{a, l, u\}$; a, l and u represent attack, legal and undefined, respectively; and x^r is the class to which the attack actually belongs; $P/c_{.im}$ is the solution provided by the neural network MLP associated to service i and subnet mask m.

The reasoning memory used by the agent is defined by the following expression: $P = \{p_1, \ldots, p_n\}$ and is implemented by means of a MLP neural network. Each P_i is a reasoning memory related to a group of cases that depend on the service and subnet mask of the client. The Multilayer Perceptron (MLP) is the most widely applied and researched artificial neural network (ANN) model. MLP networks implement mappings from input space to output space and are normally applied to supervised learning tasks (Gallagher & Downs, 2003). The Sigmoidal function was selected as the MLP activation function, with a range of values in the interval [0,1]. It is used to detect if the SOAP message is classified as an attack or not. The value 0 represents a legal message (non-attack)

and 1 a malicious message (attack). The sigmoidal activation function is given by

$$f(x) = \frac{1}{1 + e^{-ax}} \tag{5}$$

The CBR mechanism executes the following phases:

4.2.1. Retrieve

Retrieves the cases that are most similar to the current problem, considering both the type of service to which the message belongs and the subnet mask that contains the message.

- Expression (3) is used to select cases from the case memory based on the type of service and the subnet mask.

$$c_{\cdot im} = f_s(C) = \{c_j \in C / c_{j \cdot i} = c_{n+1 \cdot i}, c_{j \cdot m} = c_{n+1 \cdot m}\} \tag{6}$$

- Once the similar cases have been recovered, the neural network MLP $P/c_{\cdot im}$ associated to service i and subnet mask m is then recovered. If no cases are retrieved or the number of cases is lower than the threshold, that is 25, the neural network associated to the service $P/c_{\cdot i}$ is reused.

4.2.2. Reuse

The classification of the message begins in this phase, based on the subnet mask and the recovered cases. It is only necessary to retrain the neural network when it has not had previous training. The entries for the neural network correspond to the case elements s, n, l, u, h, b, d, t, f, a, c, k. Because the neurons exiting from the hidden layer of the neural network contain sigmoidal neurons with values between [0, 1], the incoming variables are redefined so that their range falls between [0.2–0.8]. This transformation is necessary because the network does not deal with values that fall outside of this range. The outgoing values are similarly limited to the range of [0.2, 0.8] with the value 0.2 corresponding to a non-attack and the value 0.8 corresponding to an attack. The training for the network is carried out by the error Backpropagation Algorithm (LeCun, Bottou, Orr, & Müller, 1998). The weights and biases for the neurons at the exit layer are updated by following equations:

$$w_{kj}^p(t+1) = w_{kj}^p(t) + \eta(d_k^p - y_k^p)(1 - y_k^p)y_k^p y_j^p + \mu(w_{kj}^p(t) - w_{kj}^p(t-1)) \tag{7}$$

$$\theta_k^p(t+1) = \theta_k^p(t) + \eta(d_k^p - y_k^p)(1 - y_k^p)y_k^p + \mu(\theta_k^p(t) - \theta_k^p(t-1)) \tag{8}$$

The neurons at the intermediate layer are updated by following a procedure similar to the previous case using the following equations:

$$w_{ji}^p(t+1) = w_{ji}^p(t) + \eta(1 - y_j^p)y_j^p \times \left(\sum_{k=1}^{M}(d_k^p - y_k^p)(1 - y_k^p)y_k^p w_{kj}\right)x_i^p + \mu(w_{ji}^p(t) - w_{ji}^p(t-1)) \tag{9}$$

$$\theta_j^p(t+1) = \theta_j^p(t) + \eta(1 - y_j^p)y_j^p \times \left(\sum_{k=1}^{M}(d_k^p - y_k^p)(1 - y_k^p)y_k^p w_{kj}\right) + \mu(\theta_j^p(t) - \theta_j^p(t-1)) \tag{10}$$

where w_{kj}^p represents the weight that joins neuron j from the intermediate layer with neuron k from the exit layer, t the moment of time and p the pattern in question. d_k^p represents the desired value, y_k^p the value obtained for neuron k from the exit layer, y_j^p the value obtained for neuron j from the intermediate layer, η the learning rate and μ the momentum. θ_k^p represents the bia value k from the exit layer. The variables for the intermediate layer are defined analogously, keeping in mind that i represents the neuron from the entrance level, j is the neuron from the intermediate level, M is the number of neurons from the exit layer.

When a previously trained network is already available, the message classification process is carried out in the revise phase. If a previously trained network is not available, the training is carried out following the entire procedure beginning with the cases related to the service and subnet mask, as shown in Eq. (11).

$$p_r = \mathrm{MLP}^t(c_{\cdot im}) \tag{11}$$

4.2.3. Revise

This phase reviews the classification performed in the previous phase. The value obtained by exiting the network $y = P_r^e(c_{n+1})$ may yield the following situations:

- If $y > \mu_1$ then it is considered an attack.
- Otherwise, if $y < \mu_2$, then the message is considered a non-attack or legal.
- Otherwise, the message is marked as suspicious and is filtered for subsequent revision by a human expert. To facilitate the revision, an analysis of the neural network sensibility is shown so that the relevance of the entrances can be determined with respect to the predicted value.

4.2.4. Retain

If the result of the classification is suspicious or if the administrator identifies the classification as erroneous, then the network $P/c_{\cdot im}$ repeats the training by incorporating a new case and following the BackPropagation training algorithm. Only the neural network associated to the service $P/c_{\cdot i}$ is trained.

$$p_r = \mathrm{MLP}^t(c_{\cdot im} \cup c_{n+1}) \tag{12}$$

The next section describes a case study developed to evaluate the AIDeMaS prototype presented in this paper.

5. Case study: prevention of attacks in geriatric environments

A case study was used to evaluate the efficacy of the integration of AIDeMaS within the FUSION@ architecture. The ALZ-MAS 2.0 multi-agent system was implemented through FUSION@ and used to construct a tool for dependent environments (Corchado et al., 2009). ALZ-MAS 2.0 is an improved version of the existing ALZ-MAS (ALZheimer Multi-Agent System) solution (Corchado, Bajo, de Paz, & Tapia, 2007), a multi-agent system aimed at enhancing the assistance and health care for Alzheimer patients living in geriatric residences. The main functionalities in this system include reasoning and planning mechanisms for scheduling the activities of the medical staff that are embedded into deliberative BDI agents. With the use of FUSION@ in constructing ALZ-MAS 2.0, the main components of the original ALZ-MAS are modeled as distributed and independent services, releasing the agents from high demanding computational processes such as the planning mechanism for the nurses' daily tasks, etc. The agents' functionalities in ALZ-MAS 2.0 have been separated and modeled as services. However, all functionalities are the same in both approaches.

In order to evaluate AIDeMaS, several external applications available in the ALZ-MAS 2.0 multi-agent system were used. Three specific services available in ALZ-MAS 2.0 were selected for external users. Table 3 lists these services in detail.

The selection of the services shown in Table 3 is based on the fact that the services are available to all users via the internet and are accessible from any mobile device, which can cause them to be vulnerable to different mechanism of DoS attacks. In the case of the *ObtainListDoctors* service, there are no entry parameters from the user interface, since it is an internal service that simply

Table 3
External services available for the users in ALZ-MAS 2.0.

Input Parameter	Type
RequestTreamentPatient(): Consult a treatment for a patient via Internet	
IdPatient	Int
Start_Time_Treatment	Date
Start_Date_Treatment	Date
End_Time_Treatment	Date
End_Date_Treatment	Date
RequestScheduleDoctor(): Consult the agenda of a doctor via Internet	
IdDoctor	Int
Date_Schedule	Date
Time_Schedule	Date
RequestAppointment(): Request an appointment with the doctor via Internet	
IdDoctor	Int
PatientName	String
Date_Appointment	Date
Time_Appointment	Date
Descripción	String
ObtainListDoctors(): Obtain a list of available doctors	

provides the list of available doctors to the *RequestScheduleDoctor* and *RequestAppointment* services and, were therefore not considered during the execution of the tests.

These characteristics were used to establish a controlled testing environment. The experiments were carried out in two stages; the first stage was to obtain the test data used for training the classifiers, and the second stage was to evaluate the classification mechanism within the FUSION@ architecture. In order to obtain the test data in the first stage, the Traffic agent was configured to capture the incoming SOAP messages without redirecting them to the services. In order to send the SOAP messages, 5 points (nodes) were established, from which various requests for selected services were executed. Each of these nodes belonged to a different network, i.e., each node connected to the internet using a different IP and subnetwork mask. In the first stage, each node was configured with 10 requests (SOAP messages) to be sent to each of the three selected services. Each node sent a total of 30 requests so that the total number of requests made by the five nodes to the three services was equal to 150 requests. The 30 requests sent by each node, including legal and malicious (incorrectly formed messages), were distributed as presented in Table 4. For the second stage of testing, the number of nodes and services was the same as in the first stage, but the number of requests was configured at five requests per node. At this stage, once the requests were captured by the Traffic agent, they were sent to AIDeMaS to be evaluated and classified. A total of 75 requests (legal and malicious) were sent to AIDeMaS for evaluation. The distribution of the test data is presented in Table 4.

Table 4
Distribution of requests between the nodes and services for the two stages.

Stage 1 – Data retrieval				Stage 2 – Test			
Node	Legal	Malicious	Service	Node	Legal	Malicious	Service
Node 1	9	1	1	Node 1	5	0	1
Node 1	10	0	2	Node 1	5	0	2
Node 1	8	2	3	Node 1	4	1	3
Node 2	6	4	1	Node 2	2	3	1
Node 2	8	2	2	Node 2	3	2	2
Node 2	1	9	3	Node 2	1	4	3
Node 3	5	5	1	Node 3	3	2	1
Node 3	4	6	2	Node 3	2	3	2
Node 3	3	7	3	Node 3	3	2	3
Node 4	9	1	1	Node 4	4	1	1
Node 4	9	1	2	Node 4	5	0	2
Node 4	8	2	3	Node 4	4	1	3
Node 5	4	6	1	Node 5	2	3	1
Node 5	3	7	2	Node 5	3	2	2
Node 5	1	9	3	Node 5	1	4	3

To conclude the description of the case study, some technical aspects of the equipment used to conduct the tests will now be provided. These aspects are an influential factor in the results obtained, since the performance of the system is a critical factor when assessing this type of approach. The prototype, and more specifically the classification mechanism, was tested using two standard PC connected via a 100Mbps Ethernet network, using a physical switch that was in turn connected to the local network where the FUSION@ Architecture was installed to capture the SOAP messages sent by the nodes. Each PC used by the classifiers was an HP Pavilion Intel Core 2 Duo E7200 with 4GB RAM. The tasks for the classification mechanism were distributed between the two PCs.

The following section presents the results and conclusions obtained during the case study tests.

6. Results and conclusions

The AIDeMaS security system incorporated in the FUSION@ architecture that was presented in this research study was evaluated with the data obtained in the case study outlined in Section 5. The objectives of the evaluation of the results were to confirm the correct functioning of the security system and to confirm the increase in the efficacy of the FUSION@ architecture in the detection and prevention of attacks that are hidden in service requests embedded with SOAP messages. This section will now evaluate, first of all, the functioning of the classification mechanism proposed in AIDeMaS. Secondly, the impact of the integration of the AIDeMaS within the FUSION@ architecture, as used in its online mode, is evaluated. Finally, we will present the conclusions from the evaluation of the results obtained by AIDeMas in a test case scenario.

In order to evaluate the initial prototype developed within the framework of this research project, we first analyzed the data obtained in the first stage of the case study presented in Section 5. This allowed us to perform an offline analysis to evaluate the classification ability of AIDeMaS. To do so, we recovered the 150 values from the first stage of the classification mechanism so that each of the requests could be classified. In this first phase, 149 of the cases were used to train the classifier. The classifier was then applied to the remaining case. This was done in order to facilitate the testing phase and eliminate the need to manually generate too many attacks since testing the system requires a balance to be maintained between legal queries and attacks. The second phase of the classification mechanism was carried out similarly to the first phase, except for the crossvalidation that was performed on 10% of the data. The remaining data were used to train the neural network. The results obtained for the 150 cases are shown in Table 5, which lists each of the phases for the classification mechanisms, their corresponding classifier, the total number of messages classified for each type of attack (Legal, Attack, Suspicious), the number of real messages, and the number of incorrect classifications.

Fig. 3 shows the results obtained in the first stage of the classification mechanism (Bayes classifier). Each of the stacked bars represents one of the messages and is composed of segments. The first

Table 5
Results obtained for the 150 messages evaluated.

Classifier	Classification	Messages	Erroneous
First stage – N. Bayes	Attack	52	5 False positive
First stage – N. Bayes	Legal	55	2 False negative
First stage – N. Bayes	Suspicious	43	
Second stage – ANN	Attack	16	4
Second stage – ANN	Legal	25	0
Second stage – ANN	Suspicious	2	

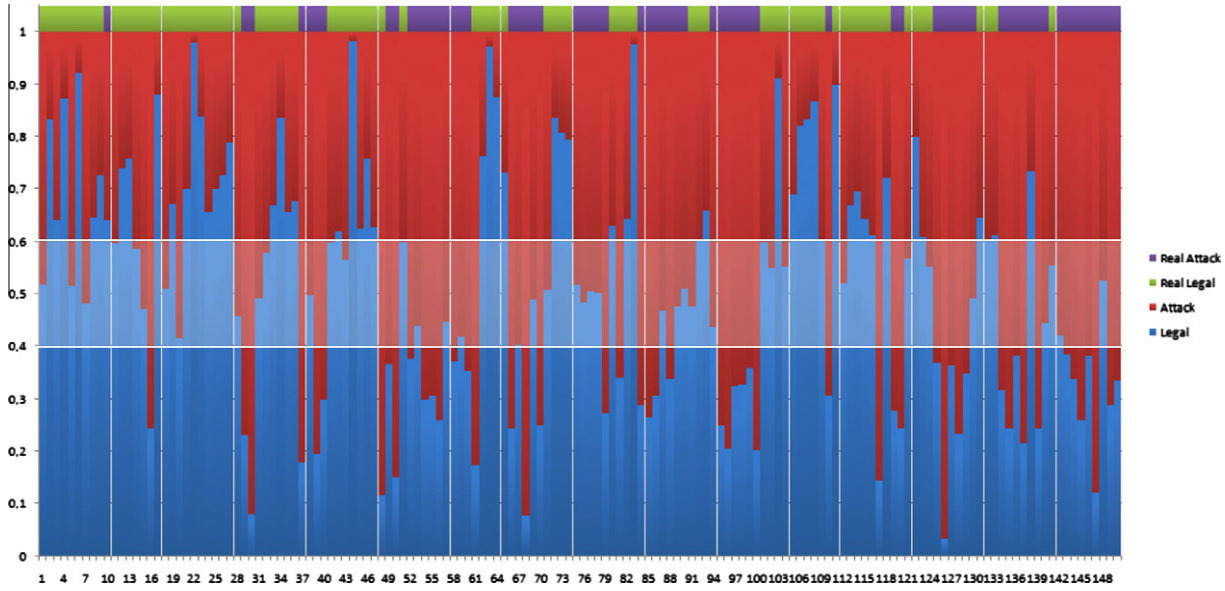

Fig. 3. Naive Bayes classification – first phase.

segment shows the probability that the service request is legal. The second segment shows the probability that the service request is malicious and contains an attack. Finally, the last segment represents whether the column is attack or legal. In the center portion of Fig. 3 there is an area that contains the suspicious requests. If the stacked bars for a legal and attack request connect in this area, then the message is considered suspicious. If the bars are located outside of this area, then the request is considered to be the type represented by the highest bar. In the upper portion of Fig. 3, there is a green colored segment if the original query was legal and violet if it was not. Looking at Fig. 3 it is possible to determine the requests that were correctly classified: if the request is legal, the blue vertical bar should exceed the value of 0.6 for the classification to be correct, otherwise the classification would be erroneous. When classifying the attack queries, the requests that were considered attacks are those where the blue segment does not exceed a value of 0.4, and the requests that were considered suspicious are those that come together in the shaded area in the center. If the results presented in Fig. 3 and Table 5 are analyzed, it is possible to conclude that the classification mechanism is very precise, given that there are only 7 errors in the first phase and 4 in the second phase of the classification mechanism. These results are particularly valuable if one considers that CBR systems function optimally when they incorporate cases into their memory, and that the experiments in this instance were performed with a limited number of cases.

In order to further evaluate the impact of AIDeMaS in the FUSION@ architecture, an online analysis was performed during the second phase of the case study. It is important to note that the security mechanism initially incorporated in FUSION@ was incapable of detecting the attack patterns used in the case study. In order to perform the online analysis, each of the cases was analyzed in the order received so that as they arrived they were classified and introduced into the system. This allowed the number of cases in the memory of cases to increase while previous cases were being analyzed. At the end of the study, the final number of cases in the memory was 203, since 22 of the new cases studied were very similar to previous cases and were not stored so as to avoid an excessive increase in the memory of cases. The results obtained are shown in Table 6, which presents a striking similarity to those in Table 5.

Table 6
Results obtained for the 75 new messages evaluated.

Classifier	Classification	Messages	Erroneous
First stage – N. Bayes	Attack	20	3 False positive
First stage – N. Bayes	Legal	36	1 False negative
First stage – N. Bayes	Suspicious	19	
Second stage – ANN	Attack	12	2
Second stage – ANN	Legal	6	0
Second stage – ANN	Suspicious	1	

An analysis of the evolution of the online classifier demonstrates that if we begin with the 150 cases in the first stage of the case study, and if each new message is classified with regards to the other messages, then it is possible to obtain an evolution of the error rate that corresponds to the increase in the case memory. In Fig. 4, where the *x*-axis indicates the number of cases introduced and the *y*-axis indicates the error, it is possible to appreciate how the error rate decreases as the number of cases increases. Fig. 4 also demonstrates the evolution of the average difference between the value for legal and attack. The average difference between both output values was initially 0.54, while the final value obtained is 0.77. This makes it possible to reduce the number of suspicious service requests and to make a classification with a lower level of uncertainty. In order to calculate these values, the output values from the neural network as well as the Bayes classifier were rescaled to take values in the interval 0.1–0.9 so that both methods could be compared as the iterations were applied.

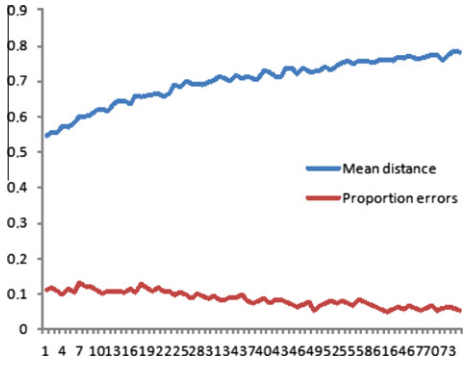

Fig. 4. Error rate evolution related to the number of messages.

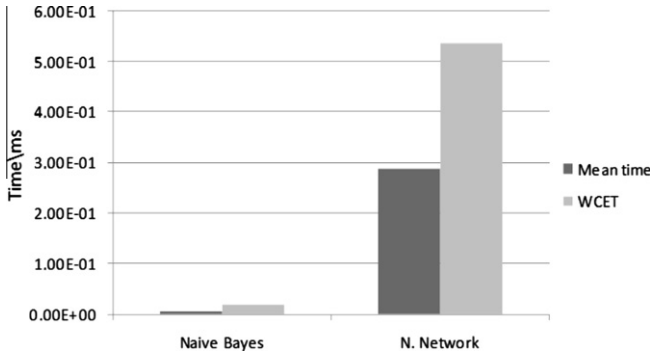

Fig. 5. Mean execution and worst case estimated time (WCET) of Naive Bayes and ANN.

Finally, this paper evaluates the average execution time and the worst case estimated obtained during the experiments for both the Naive Bayes and the ANN techniques. These two indicators are shown in Fig. 5. As can be seen in Fig. 5, the average execution time for the Naive Bayes classifier is notably lower than the average execution time obtained for the artificial neural network. It is also possible to observe the worst case estimated time for both techniques, and how the Naive Bayes classifier provides a too much better performance for the worst situation.

The AIDeMaS presented in this paper proposes a new perspective for detecting and blocking attacks in web service environments. Specifically, the use of FUSION@ improves the functionality of the mechanism that was previously installed. The results are promising and allow us to conclude that AIDeMaS can be considered as a solid alternative to prevent and detect DoS attacks in service environments. However, there is still much work to do, especially with regards to checking the validity of our architecture in heterogeneous real environments. These are our next challenges.

Acknowledgements

This work has been partially supported by the MICINN project TIN 2009-13839-C03-03

References

Aamodt, A., & Plaza, E. (1994). Case-based reasoning: foundational issues, methodological variations, and system approaches. *AI Communications, 7*(1), 39–59.

Anderson, S., Bohren, J., Boubez, T., Chanliau, M., Della, G., Dixon, B., (2004) Web Services Trust Language (WS-Trust). http://www.ibm.com/developerworks/library/specification/ws-trust/. Accessed 18 April 2009.

Anderson, S., Bohren, J., Boubez, T., Chanliau, M., Della-Libera, G., Dixon, B., (2004) Web Services Secure Conversation Language (WS-SecureConversation) Version 1.1. http://www.msdn.microsoft.com/ws/2004/04/ws-secure-conversation/. Accessed 18 April 2009.

Bajo, J., Corchado, J. M., Pinzón, C., Paz, Y. D., & Pérez-Lancho, B. (2008). SCMAS: A Distributed Hierarchical Multi-Agent Architecture for Blocking Attacks to Databases. *International Journal of Innovative Computing, Information and Control*.

Bratman, M. E., Israel, D. J., & Pollack, M. E. (1988). Plans and resource-bounded practical reasoning. *Computational Intelligence, 4*(3), 349–355.

Brena, R. F., Aguirre, J. L., Carlos, C., Ramírez, E. H., & Leonardo, G. (2007). Knowledge and information distribution leveraged by intelligent agents. *Knowledge and Information Systems, 12*(2), 203–227.

Cerami, E., (2002). Web Services Essentials Distributed Applications with XML-RPC, SOAP, UDDI & WSDL. First Edition O'Reilly & Associates 2002.

Chung, J-Y. (2008) Emerging view of service-oriented computing and applications. In 12th International Conference on Computer Supported Cooperative Work in Design (pp. 3-3).

Corchado, J. M., Bajo, J., de Paz, Y., & Tapia, D. I. (2007). Intelligent environment for monitoring Alzheimer patients, agent technology for health care. *Decision Support Systems, 44*(2), 382–396.

Corchado, J. M., Tapia, D., & Bajo, J. (2009). A Multi-Agent Architecture for Distributed Services and Applications. *International Journal of Ambient Computing and Intelligence - IJACI.*, 15–26.

Della-Libera, G., Gudgin, M., Hallam-Baker, P., Hondo, M., Granqvist, H., Kaler, C., Maruyama, H., McIntosh, M., Nadalin, A., Nagaratnam, N., Philpott, R., Prafullchandra, H., Shewchuk, J., Walter, D., Zolfonoon, R., (2005) Web services security policy language Version 1.0 (WS-SecurityPolicy). http://www.specs.xmlsoap.org/ws/2005/07/securitypolicy/ws-securitypolicy.pdf. Accessed 20 April 2009.

Duda, R. O., & Hart, P. E. (1973). *Pattern Classification and Scene Analysis*. New York: John Willey and Sons.

Elsas, A. (2003). *Integration of e-Government and e-Commerce with Web Services*. *Electronic Government* (Vol. 2739/2003). Heidelberg, Berlin: Springer.

Gallagher, M., & Downs, T. (2003). Visualization of learning in multilayer perceptron networks using principal component analysis. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, 33*(1), 28–34.

Gruschka, N., & Luttenberger, N. (2006). *Protecting Web Services from DoS Attacks by SOAP Message Validation*. *Security and Privacy in Dynamic Environments* (Vol. 201/2006). Boston: Springer.

He, M., & Leung, H.-f. (2002). Agents in E-commerce: state of the art. *Knowledge and Information Systems, 4*(3), 257–282.

Hori, M., & Ohashi, M. (2005). Applying XML Web Services into Health Care Management, 38th Annual Hawaii International Conference on System Sciences, 2005 - HICSS '05, IEEE Computer Society, Vol. 6: pp. 155-155.

Jennings, N. R., & Wooldridge, M. (1995). Applying agent technology. *Applied Artificial Intelligence, 9*(4), 357–369.

Jensen, M., Gruschka, N., Herkenhoner, R., Luttenberger, N., (2007) SOA and Web Services: New Technologies, New Standards - New Attacks, Fifth European Conference on Web Services, pp. 35-44.

Lauro, S., & Gian Luca, F. (2007). Knowledge representation for ambient security. *Expert Systems, 24*(5), 321–333.

Laza, R., Pavón, R., Corchado, J. M., (2003) A Reasoning Model for CBR_BDI Agents Using an Adaptable Fuzzy Inference System, 10th Conference of the Spanish Association for Artificial Intelligence - CAEPIA-TTIA03, Springer, Vol. 3040: pp. 96-106.

LeCun, Y., Bottou, L., Orr, G. B., & Müller, K. R. (1998). *Efficient BackProp, Neural Networks: Tricks of the Trade. Vol. 1524/1998*. Berlin / Heidelberg: Springer.

Loh, Y-S., Yau, W-C., Wong, C-T., Ho, W-C., (2006) Design and Implementation of an XML Firewall, International Conference on Computational Intelligence and Security, Vol. 2: pp. 1147-1150.

Nadalin, A., Kaler, C., Monzillo, R., Hallam-Baker, P., (2006) Web Services Security: SOAP Message Security 1.1 (WS-Security 2004). http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf. Accessed 21 April 2009.

Pinzón, C., De Paz, Y., & Bajo, J. (2008). *A Multiagent Based Strategy for Detecting Attacks in Databases in a Distributed Mode*. *International Symposium on Distributed Computing and Artificial Intelligence - DCAI2008* (Vol. 50). Berlin / Heidelberg: Spring.

Pokahr, A., Braubach, L., & Lamersdorf, W. (2003). Jadex: Implementing a BDI-Infrastructure for JADE Agents, EXP – in search of innovation (special issue on JADE) 3(3), 76–85.

Pulier, E., & Taylor, H. (2005). *Understanding Enterprise SOA*. Greenwich, CT, USA: Manning Publications Co.

Vossen, G., & Westerkamp, P. (2003). E-Learning as a Web Service, Database Engineering and Applications Symposium, International IEEE Computer Society. 0 242.

Yee, C. G., Shin, W. H., Rao, G. S. V. R. K., (2007) An Adaptive Intrusion Detection and Prevention (ID/IP) Framework for Web Services, International Conference on Convergence Information Technology (ICCIT '07), IEEE Computer Society, pp. 528-534.