**Australian Journal of Basic and Applied Sciences**, 9(11) May 2015, Pages: 444-450

ISSN:1991-8178

**Australian Journal of Basic and Applied Sciences**

Journal home page: www.ajbasweb.com

# Developmentof Intelligent Enterprise Applications using Multi-Agent Systems

[1]Aluizio Haendchen Filhoand [2]César Ricardo Maia de Vasconcelos

*[1]University Center of Brusque,Information Technology Center, Durval Luz St., 123 – Santa Terezinha88352-400,Brusque/SC, Brazil*
*[2]Potiguar University,Postgraduate Programin Management, Eng. Roberto Freire Ave., 2184 – Ponta Negra 59082-902, Natal/RN, Brazil*

**ARTICLE INFO**

***Article history:***
*Received 12 March 2015*
*Accepted 28 April 2015*
*Available online 5 May 2015*

***Keywords:***
*Intelligent information systems, information systems arquitecture, intelligent ERP, intelligent enterprise application*

**ABSTRACT**

Typically, information resources in Web systems are dynamic, distributed, and heterogeneous. Since these computing environments are open, information resources can be connected or disconnected at any time. Middleware technologies change so rapidly that designers must adapt existing software architectures to incorporate new emerging ones. This scenario represents a natural field for multi-agent systems, a usual solution for a large set of multi-task distributed and heterogeneous application. In this paper presents a SOA based framework to support development of agent-based systems. Software agents have been introduced as the next step in the development of intelligent enterprise applications, in which autonomous proactive objects replace the most commonly used reactive components.

**© 2015 AENSI Publisher All rights reserved**.

**ToCite ThisArticle:**Aluizio Haendchen Filho and César Ricardo Maia de Vasconcelos, Development of Intelligent Enterprise Applications using Multi-Agent Systems. ***Aust. J. Basic & Appl. Sci.,*** *9(11): 444-450, 2015*

## INTRODUCTION

Large organizations frequently use heterogeneous and complex information systems in which components need to exchange information in order to provide both efficient support to business processes and consistent information management. Current ERP systems move from monolithic architectures based on passive software components to flexible, adaptive and opened architectures, composed of dynamic and proactive agents. This scenario requires flexible and evolving applications to support the dynamic of these systems and the constant evolution of requirements.

Software agents have been introduced as the next step in the development of enterprise information systems, in which autonomous proactive objects replace the most commonly used reactive components (Buschmann, 2002). They can dynamically discover, orchestrate and compose services, check activities, run business processes, and integrate heterogeneous and distributed applications. Due to these features, MAS have been considered a natural solution for a large set of multi-task distributed (Jain and Dahiyal, 2012), heterogeneous and complex application.

One important advance in the context of software development that can help in facilitating the creation of distributed information systems is SOA, a service-oriented architecture. SOA remains the best option available for legacy systems integration and leverage (Tapia *et al*., 2008). The technologies to implement SOA will certainly evolve to address emerging needs, but its basic concept will remain. One of the primary benefits of SOA is the ability to compose applications, processes, and assemble new functionalities from existing services. Both SOA and multi agent systems are inherently distributed software systems, having significant implications in their applicability (Srinivasan; Singh; Kumar, 2011). This paper presents a SOA-Based Multi-Agent Framework which provides a middleware and a framework to facilitate the development of intelligent enterprise information systems.

### *The Architecture Overview*

Middleware provides infrastructure services, making possible for developers to abstract complex functionalities such as management, communication, concurrency, platform heterogeneity, language, and network. It offers high-level abstractions that facilitate the understanding of the structure, and improves the capability of developers to modulate, reuse and extend the software infrastructure in order to work more efficiently with distributed environments (Gamma, 1995). The architecture is composed by a front-end server (FES) and agent containers (AC). FES performs the platform integration rules; it decouples the containers eliminating the complexity of forwarding requests and service providers' look-up. Figure 1 shows the platform generic architecture.

**Corresponding Author:**César Ricardo Maia de Vasconcelos,Potiguar University (UnP), Postgraduate Program in Management. Eng. Roberto Freire Ave., 2184 - CEP. 59082-902 Ponta Negra, Natal/RN, Brazil. Tel: +55 (84) 3216 8616E-mail: cesar.vasconcelos@terra.com.br

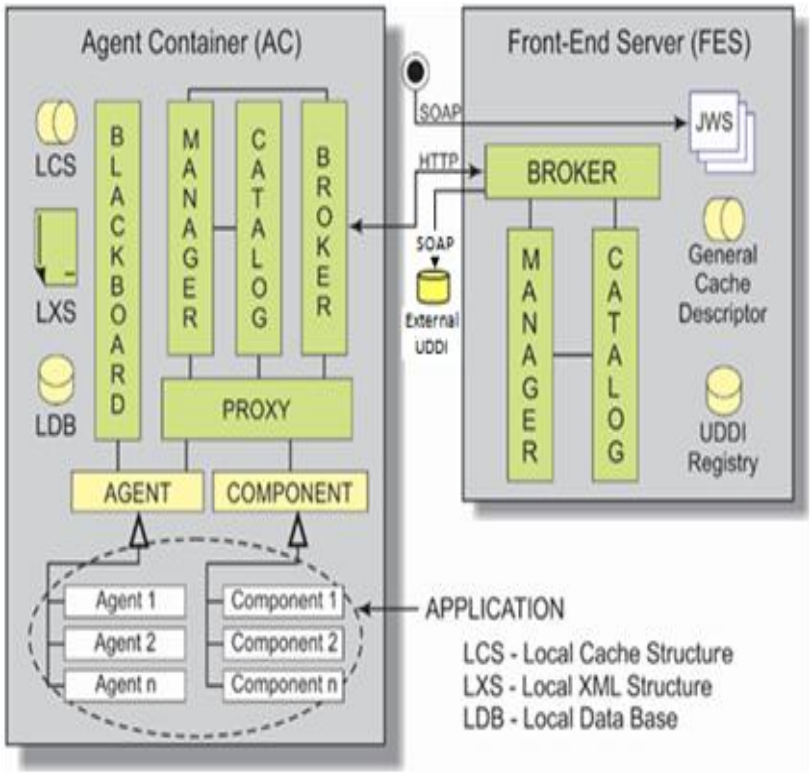


**Fig. 1:** Generic architecture and its main concepts.

AC is a lightweight container where application agents and components are hosted. In a practical example, we can assume that different ACs can be used in departments of a company, such as marketing, sales, finance and so on. They can also be simply used to encapsulate procedures of legacy systems as services wrapper components. Internal communication between ACs and the FES is carried via HTTP, and the interoperability with external applications and services is assured via SOAP. An application can hold concrete agents and/or components. Agents are entities that interact with other agents and a structured environment, aiming at some global purpose. Components are independent processing units, characterized as purely reactive elements. The Blackboard supplies an asynchronous and indirect communication way for the agents' workflow. Local resources structure placed in the container is represented by LCS and LXS descriptions maintained in XML files. The consolidated resources are represented by General Cache Descriptor, and UDDI represents the web services made available by the platform. Application is a virtual entity representing all applications developed in a Container.

***The Front-End Server Architecture***

The architecture encompasses the Message-Oriented Model (MOM), Resource-Oriented Model (ROM), Management-Oriented Model (MGT), and Service-Oriented Model (SOM) that are respectively implemented by Broker, Catalog, Manager, and Proxy agents. These models follow the Web Services Architecture (WSA) reference model (Lerman, 2000) introducing a set of concepts and high-level abstractions in order to facilitate the understanding of the architecture. MOM focuses on message structure and path; ROM concentrates on resources and how they are manipulated; MGT is responsible for managing transactions, services and resources; and SOM are related to services and supply chains. There is no SOM (Proxy) in the FES, since there is no service providers related to this server. Figure 2 shows the FES package diagram with their classes and relationships.

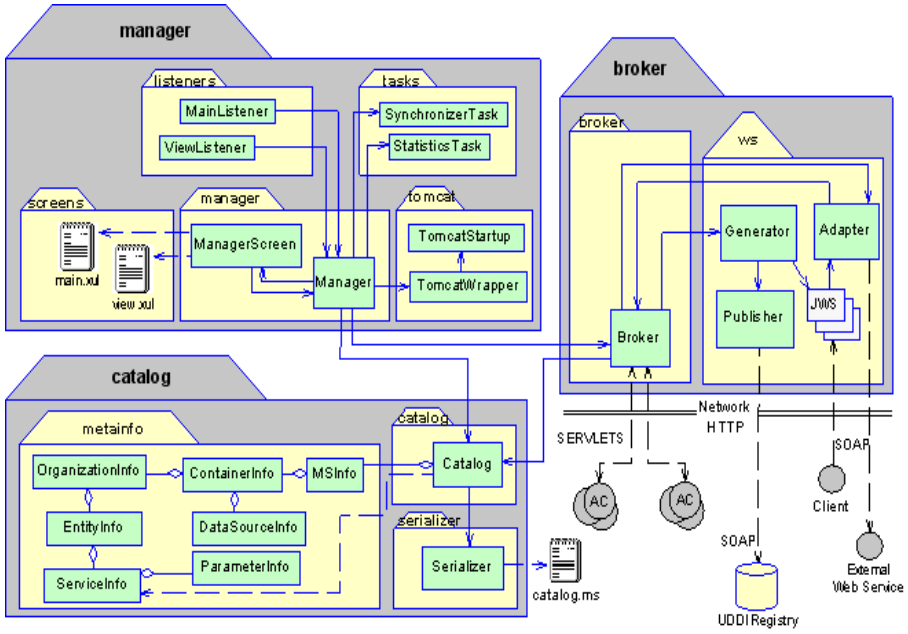


**Fig. 2:** The FES framework architecture.

We use the concept of agent to define the middleware elements of the architecture. The introduction of middleware agents performing infrastructure services as encapsulations of their behavior leads to a dynamic and collaborative architecture (Silva *et al*., 2003).

The Broker agent placed on FES decouples components that interact by remote service invocation, and plays two important roles. The first, related to message transport, involves the following procedures: (i) performing the synchronization procedures with the AC; (ii) message receiving/forwarding and pack/unpack; (iii) receiving messages from AC when it desires to self-register in the FES; and (iv) receives messages to service updating in the resource catalogue.The second, related to interoperability, includes the following procedures: (i) send/receive web services requests using the SOAP protocol; (ii) perform WS automatic generation and (iii) translate the received messages from SOAP to MIDAS interface format and vice versa. The Broker class is a servlet responsible for receiving requests arriving from AC and forwarding them to the service provider. During the message processing it interactswith Catalog to get the URL of the AC which provides the service to forward the request. Broker also interacts with the Manager providing information such as response time, exceptions occurrence, container synchronism, among others.

Procedures of Web services such as handling, receiving, translation and generation are performed by the classes placed in *ws* package. The SOAP requests arriving from an external web service are read by the Axis and stored as JWS (Java Web Services) in *jws* files. The Apache Axis is an open source framework used for construction of web services in SOAP. The Adapter class is in charge of translating messages from SOAP to MIDAS interface and vice-versa. The Generator class performs automatic generation of WSDL service specification departing from the *jws* files. The Publisher class is responsible for the publication of WSDL specification in UDDI register.

The procedures for management and handling of the global resource structure are performed by the FES Catalog agent, which consists of three packages: *catalog*, *metainfo*, and *serializer*. All service requests are received by the Catalog agent through a single gateway; it is responsible for informing whether the service is available or not. The Serializer class performs all procedures related to the persistence of the resource structure. The data structure is consolidated and stored in serialized files and the representations are kept in cache memory increasing efficiency. Quick service information is obtained using the service name as an index, without traversing the structure.

The Manager agent offers and manipulates the GUI resources of the platform. Requests for services visualization and discovery are forwarded from Manager to the Catalog. The FES Manager agent plays a set of tasks that enables the control over the platform, such as containers synchronism management, activity checking, and statistic reports. In order to perform these tasks, the corresponding responsibilities were assigned to five packages:*manager*, *screens*, *listeners*, *tomcat* and *tasks*.

The Manager class placed in the *manager* package acts as a gateway, receiving messages from other middleware or application agents. The ManagerScreen class offers GUI wizards for specification, visualization and service discovery in

the resource structure. The *screens* package contains the GUIs specification stored in XML format, in *view.xul* and *build.xul* files. This enables the evolution of the GUI simply by changing the XML file without modifying the source code. The *listeners* package contains the classes in charge of listening and checking the actions performed by users. The classes responsible for keeping the Tomcat server are placed in the *tomcat* package, making the connection and configuration of network procedures transparent. The *tasks* package contains the classes responsible for server monitoring tasks, statistics, and QoS report. The SynchronizerTask class continuously performs synchronism verification procedures, checking the containers connected to the platform. The StatisticsTask class is in charge of updating information about memory use, active number of threads, gathering statistics and metrics. One example is the response time and exceptions that occur in the system execution.

***The Agent Container Architecture***

AC is a container in which agent-based applications can be developed. Its architecture is composed of two basic structures: one includes the middleware agents and other comprises the Agent Framework. Although the middleware agents placed in AC have similar functions to those performed in the FES, they have behavioral and structural differences. According to Booth (Lerman, 2000), Web architectures are manageable if they include a set of operations that support management of messages, services, resources, and security. Figure 3 presents the AC overview.

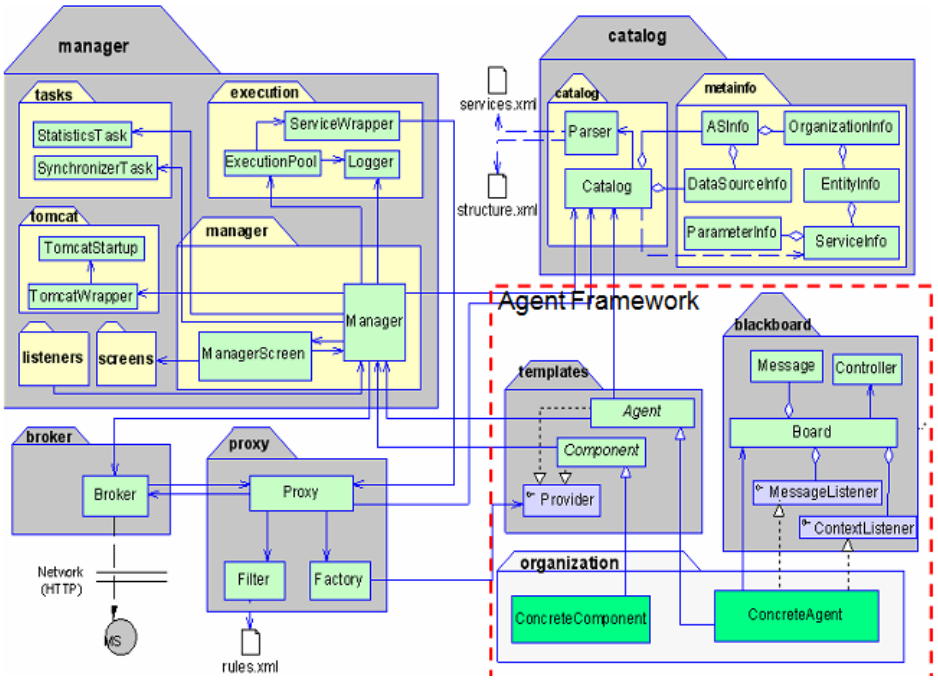


**Fig. 3:** The AC architecture.

The main difference between Broker placed in AC to those located in FES is that the AC Broker does not need to interact with external applications. The messages changed internally in the platform move through HTTP, using the MIDAS native interface. Likewise in the FES, the AC Broker class is also a servlet that uses similar procedures to send and receive messages. While in the FES there is no SOM, since there is no applications service providers, the functions in the AC related to SOM are performed by the Proxy agent.

***Service-Oriented Model: Architecture Adaptability, Flexibility, and Location Transparency***

The Proxy acts as a service provider representative ensuring location transparency, flexibility, and adaptability to the architecture. It also contributes to facilitating the communication among elements of the presentation, logic, and database layers.During the service request processing, the following procedures are performed by the Proxy: (i) it validates the service, verifying if the request is correct and if the service can be available at that moment; (ii) identifies the service provider, carried out in collaboration with Catalog agent; (iii) creates the instance and invokes the required service.

The external service requests are sent by the Broker and the internal ones by the Manager. When the Proxy receives an external service request, it inquires the Catalog about the service availability. If the service is available, the Proxy obtains from

Catalog the name of the service provider entity. Then, the Proxy forwards the message to the Filter class, which performs message verification and authentication, in order to make sure that there is no rule against the use of that service. The rules are stored in an XML file and, at any moment, they can be altered or the administrator can insert new ones without modifying the source code.

The Factory class fulfills an important role, since it is in charge of the dynamic configuration and dynamic creation of instances, making the architecture flexible and adaptable. Dynamic configuration refers to the capacity of an agent running in the JVM, which can be changed at runtime. The dynamic instances creation is performed by using the Factory Method pattern (Silva *et al*, 2003), which abstracts the process of instances creating. Thus, new agents can be dynamically inserted into the platform without having to change the Factory class code. Figure 4 illustrates the Factory class and its methods.

**Fig. 4:** The AC framework architecture.

The *reloadClasses* method is invoked to perform the container configuration. It uses the *java.net.URLClassLoader* library that provides an URLClassLoader type object. The object allows updating the codes that are running on the JVM. The *loadClasses* method is invoked to perform dynamic loading of classes. The *invokeEntity* method is used for a service request execution. It invokes the *instantiateEntity* method to obtain the interface of the provider agent. Then, it dynamically creates an instance and invokes a procedure for service execution.

The Proxy agent provides location transparency, eliminating the difficulties in identifying services providers and in forwarding requests. The proxy eliminates the complexity of forwarding requests and service providers' look-up. The presentation layer needs only to know the desired service name, leaving the remaining procedures to the architecture.

***Resource-Oriented Model Ensures Reusability***

ROM is based upon the concept of resource; each resource is an accessible distributed service that is handled through a standard, common interface. An important aspect of SOA is the extensive use of metadata (Craig, 1993). It is possible to insert new services by simply registering the service in the XML descriptors files. The agents are invoked through a unique interface, instead of having implemented different classes for each service. A resource description is a metadata representation that makes possible for a human or software agent to discover service and providers. The metadata representation is relevant because it fosters interoperability by requiring increased precision in the documentation of services. In the representation of the resources we use the concept of meta-class. This resource structure is described in the *metainfo* package, in which each element is represented by a meta-class implemented as an entity. The ACInfo class is the root of a meta-classes hierarchy and contains information about all resources the stored in the AC.

The procedures of resources management and handling are performed by Catalog and Parser classes, placed in the catalog package.The Parser class performs procedures for handling XML files using the DOM (Document Object Model) to information store and retrieving. The resources representation and description are stored in two XML files type: *structure.xml* and *services.xml*. As XML has a hierarchical structure, the representation of the resource in the first type is intuitive. The second type ensures efficiency, since quick service information can be obtained using the service name as an index, without traversing the structure.

***Management Model: Request Abstraction***

Likewise in the FES, the AC Manager has many similarities with the FES Manager related to the container network server and GUI management. However, it has responsibilities beyond those presented in FES. The main one is the execution layer. Manager uses a message execution pool of synchronous and asynchronous calls in order to deal with high concurrency. The *execution* package contains classes responsible for processing service requests. The aim is to allow more intuitive and flexible operation in the procedures of handling service requests. It offers a wrapper object, which facilitates the parameter handling during request assembly, enabling full transparency and simplicity in the ordering service processes. As can be seen in Figure 4, the tasks of the *execution* package are distributed into three classes:

- Logger: implements a log file that is used to store information about requests processing and statistics, allowing certify if the service requests were successfully met or if errors have occurred, error type, etc.
- ExecutionPool: controls the execution pool of synchronous and asynchronous calls, providing two methods: (i) *runService*, used to make synchronous calls, and (ii) *submitService*, used for asynchronous calls. The *submitService* method returns an object type Future <List> of *java.lang.concurrency* Java API which is a known entity of concurrent

programming, considering a future that may or may not happen.

- ServiceWrapper: enables the performance of more intuitive and flexible operations to build a service request. It has methods such as *addParameter* and *removeParameter* through which the agent can add parameters to a request without worrying about its order, location or casting for type conversion, since each attribute represents a type.

Requests abstraction standardizes the parameters that go along with a request. It is enough to know the desired service name, adding its corresponding parameters, and the architecture takes care of the remaining procedures. The introduction of middleware agents performing infrastructure services as encapsulations of their behavior leads to a dynamic and collaborative architecture (Odell, 2005). By taking middleware and application agents as different entities, the architecture enables a clearer and more coherent view of the agents roles and its participation in the architecture.

***The Agent Model:***

The Agent Model (AM) introduces a framework to simplify the development of intelligent agent-based applications. Three packages compose AM: *templates*, *organization* and *blackboard*. The AM core is the *templates* package that encompasses the abstract classes Agent, Component, and the Provider interface. The abstract classes define an algorithm that includes concrete and abstract methods. The concrete methods supply a set of already implemented functions, such as the interfaces to interact with the architecture, the procedures that start the agent execution thread, and the basic methods for the agents' lifecycle management. The abstract methods provide the hot spots from which specific characteristics of the application agents (or components) can be implemented. The abstract methods also supply signatures to handle the agent lifecycle, and methods to enable the blackboard.

The abstract class Agent implements the Provider interface, which defines a basic signature where agents receive service requests forwarded by Proxy. The interface enables one of the basic requirements for the dynamic creation of agents and components, making them able to reply requests. The abstract class also implements the Callable interface (Java Runnable interface improvement) that enables to create threads using the Java concurrency API. *Organization* is a virtual entity that can hold concrete agents and/or components. Agent organizations can be understood as entities in where a multitude of agents interact, within a structured environment, aiming at some global purpose. They demand the integration of organizational and individual perspectives and the dynamic adaptation of the agents to environmental changes.

Blackboard is one of the most used information exchange techniques in symbolic cognitive MAS. It permits to send multicast messages for all the listeners agents registered or only for a specific agent. In the blackboard model, agents are called sources of knowledge and the blackboard is a global memory, accessible to all agents. It contains the current state of the problem and the actions performed by agents gradually modify the data structure on the blackboard. The blackboard is a metaphor: agents communicate with each other "writing and reading the blackboard," as humans would do in a brainstorming section (Bakken and Middleware, 2003). Its structure follows the basic blackboard pattern: the knowledge sources represent the agents, the data structure is visible to all agents and the controller is responsible for notifying the agents about the changes in the environment.

The structure of Blackboard is composed by the Board, Controller, Message classes and by the MessageListener and ContextListener interfaces. The blackboard acts as a sensor that perceives changes in the environment, addressing them to those application agents that implement the interfaces. The MessageListener interface defines one signature, used to notify the agents about messages of interest. The ContextListener interface defines another signature, invoked by Board class to notify agents of changes on the structure of global variables. The Controller class maintains the list of registered listeners and groups. It provides methods that are invoked by the Board class to notify the agents of global variables messages or changes

## REFERENCES

Bakken, D. and E.Middleware, 2003. Encyclopedia Of Distributed Computing, *Kluwer Academic Press*, London.

Buschmann, F.,2002. "Patternn-Oriented Software Architecture," John Wiley &Soons Ltd. England.

Craig, I.D.,1993. "A new interpretation of the blackboard metaphor," *Technical Report*, University Of Warwick Coventry, Uk.

Gamma, E.,1995. Design Patterns – Elements Of Reusable Object-Oriented Software, Addison-Wesley Longman, Inc.

Jain, P. and D.Dahiyal, 2012. "An Architecture of a Multi Agent Enterprise Knowledge Management System Based on Service Oriented Architecture.",IJCSI International *Journal of Computer Science Issues*, Vol. 9, Issue 2, No 3, March 2012.

Lerman, K.,2000. Design and Mathematical Analysis of Agent-based Systems. In: First International Workshop, FAABS 2000 Greenbelt, MD, USA, April 5-7.

Odell, J.,2005. "Agent-Based Process Management for SOA and WS Applications," Omg Workshop On MDA, SOA and Web Services March 21-24, 2005, Orlando, Fl, Usa.

Silva, V., A.Garcia, A.Brandão, C.Chavez, C.Lucena, P.Alencar, 2003. "Taming Agents and Objects in Software Engineering", Software Engineering for Large-Scale Multi-Agent Systems. *Lecture Notes in Computer Science*, Volume 2603: 1-26.

Srinivasan, S., S.K. Singh, V.Kumar, 2011. "Multi-agent System based Service Oriented Architecture for Supply Chain Management System (MAS-SOA-SCM)", *International Journal of Computer Applications* (0975 – 8887) Volume 27– No.5.

Tapia, D.I, S.Rodríguez, J.Bajo, J.M.Corchado, 2008. "FUSION@, A SOA-Based Multi-agent Architecture.", International Symposium on Distributed Computing and Artificial Intelligence 2008 (DCAI 2008). *Advances in Soft Computing*, Volume 50.

Tapia, D.I., S.Rodríguez, J.Bajo, J.M. Corchado, 2009. "FUSION@, A SOA-Based Multi-agent Architecture.", International Symposium on Distributed Computing and Artificial Intelligence 2008 (DCAI 2008). *Advances in Soft Computing*, 50: 99-107.