A Multi-Agent System Infrastructure for Large-Scale Autonomous Distributed Real-Time Intelligence Gathering Systems

Matthias Scheutz
Human-Robot Interaction Laboratory
Indiana University
Bloomington, IN, 47406

Abstract

Complex national security applications require support for component distribution, secure communication, fault-tolerance, and dynamic system reconfiguration that generally fall under the purview of multi-agent systems (MAS). Current MAS, however, do not provide the support and flexibility for the design and long-term operation of large-scale sophisticated intelligence applications. We argue that such systems require features that result from the synergy of both MAS and SAS (single agent systems) and introduce our `ADE` system, showing how it supports three such synergistic features that we believe are critical to long-term, large-scale homeland security systems.

1 Introduction

Future national security applications for automatically monitoring, analyzing and reasoning with data from real-time multi-modal data streams will have to use complex, large-scale, distributed reasoning architectures that can combine evidence from different sources. This, in turn, will require an intelligent, highly fault-tolerant, secure, distributed, real-time computational infrastructure. Consider, for example, an automated multi-modal threat detection system that analyzes data originating from multiple heterogeneous data sources. These heterogeneous data sources may include both "soft data" (e.g., information stemming from text or verbal reports) and "hard data" (i.e., information stemming from physical sensors that monitor critical areas). The system corroborates all provided information (both from current and previous data) to determine its credibility in order to detect potential threats. To be able to accomplish this functionality, such a system needs to be highly parallel to deal with a large amount of streaming data, performing fast, multi-lingual language parsing and semantic analyses to determine the meaning of soft data, then storing the semantic representations in a language neutral (logical) form in distributed databases indexed by various keys. It is expected that data mining tools will be able to access the information stored in databases in order to determine patterns, which will then invoke reasoning tools that operate in parallel on semantic representations to determine potential threats and their likelihood, while partial or inconclusive results from analyses will be stored for future reference (e.g., [5]). The system will also be able to produce written reports summarizing its findings. Interactive access to the system will allow investigators to query the database and formulate testable hypotheses whose certainty the system can determine based on the stored data. While all this activity is taking place, the system monitors its "health", dynamically reconfiguring itself to relocate components from hosts that crashed or are not reachable via network connections – all while the system stays fully operational.

In general, these national security applications will require the computational infrastructure to support highly controlled access to different kinds of classified information stored within the system, in addition to sophisticated analysis and reasoning tools that will integrate a great variety of current and future AI technology (from natural language processing and understanding, to machine translation, hypothetical reasoning with uncertainty, various forms of machine learning, etc.). Moreover, refined internal monitoring and supervision tools are needed to detect failures of components, initiate recovery from failure, and ensure the long-term autonomous operation of the system. What kinds of system can provide the necessary infrastructure for the implementation of such an autonomous intelligent distributed application?

The natural place to look for an answer are *multi-agent systems* (MAS), which provide an agent-based infrastructure for distributed computing. Agents in the context of MAS, call them "MAS-agents", are computational processes that implement the autonomous, communicating functionality of a distributed application [2]. While many MAS provide the necessary tools to implement secure, fault-tolerant distributed systems as determined by the "MAS-architecture" (i.e.,

the overall blueprint of the multi-agent system), they do not provide support for the implementation of functional components of the architectures of their constituent MAS-agents. Yet, support for components like memories, reasoning, planning, or learning engines, etc. is likely going to be necessary for the implementation of the many different kinds of sophisticated analyses and reasoning tasks described above.

This kind of support is typically found in *single agent systems* (SAS), which focus developing control architectures for intelligent agents ("SAS-architectures"). "SAS-agents" are typically *situated* in some environment and are capable of *flexible autonomous* action in order to meet their design objectives [4]. However, SAS are not concerned with highly parallel processing environments with hard real-time requirements, having been largely developed for single CPU architectures. Hence, they do not provide the kinds of distribution mechanisms to preserve the parallelism that their architectures might allow for, nor do they provide mechanisms for secure, controlled access to parts of their architecture.

We believe that a *synthesis* of both kinds of systems is necessary, which can provide the support for SAS agent architecture development together with the infrastructure necessary for the interaction of MAS components. Specifically, we propose a MAS system infrastructure that allows for "MAS-agents" to be designed with "SAS-architectures" (e.g., the cognitive architecture SOAR), where the components of these "SAS architectures" (e.g., rule-base, working memory, etc.) are themselves "MAS-agents". The main utility of such an infrastructure for intelligent national security systems is that it allows (1) for a secure, large-scale distribution of the system at the level of architectural components of SAS-agents, (2) for the implementation of sophisticated, yet efficient reasoning tools in existing cognitive framework, and (3) for *synergistic features* resulting from integrating SAS and MAS that will significantly improve the long-term operation and maintenance of such systems.

2 SAS and MAS Features

Single agent systems are concerned with the design and implementation of individual agents. We identify a non-exhaustive list of ten typical SAS features to be able to see their potential contribution for national security applications.

The first five features – the *functional* features – support the design of agent architectures. *Architecture support* (**S1**) provides support for a specific architecture type (e.g., behavior-based, BDI, 3-layer hybrid, etc.). Architecture support may be provided at a component or a program language level. Additionally (or alternatively), systems may provide pre-defined routines, enabling rapid agent development. These include *architectural components* (**S2**; i.e., schedulers, rule-interpreters, learning mechanisms, etc.) and *data processing routines* (**S3**; i.e., voice recognizers, natural language parsers, graphical analysis tools, etc.). Another major concern of an agent system is to provide facilities and standards for the interaction of an agent and its environment, which we refer to as the system's *device specification* (**S4**) and *interface abstraction* (**S5**). The first concerns particular types of interface devices to gather data (sensors) and to transmit information (effectors); different examples of a single type may be available and should be easy to specify and change as needed. The second refers to the abstractions that many devices have in common, separating the form of a device from its function.

The remaining five features can be characterized as *implementation* features. Integrated *simulation* (**S6**) and *data logging* (**S7**) tools ease the burden of testing and debugging. The availability and use of simulated versus real *concurrency* (**S8**) in agent computing is often a requirement for good agent performance, also supported by the ability to use a *variable update frequency* (**S9**), so that sensor information is made available as soon as it is acquired. These features are typically enabled by the designer's *programming language choice* (**S10**). That choice should not be dictated by the agent system itself; a designer might want to use a specific language for implementation of agent control code or agent function.

Different from SAS, MAS are primarily concerned with the design of distributed applications in which multiple, often heterogeneous agents interact. In general, MAS provide the framework in which individual agents operate without specifying the function or architecture for individual agents (only for agents that effectively implement the MAS infrastructure such as "registry agents", "yellow and white pages agents", etc. are functional roles and implementations given). We identify ten MAS features in total, four that can be considered *functional*, five that are *implementational*, and one that has elements of both.

We begin with the functional features, starting with the level of *communication abstraction* (**M1**) supplied by the system, from sockets to a well-defined languages (such as KQML or the FIPA ACL). Communication between agents raises issues of trust, and should be subject to *agent authentication* (**M2**), while the system should also provide some level of *system access control* (**M3**). Also at the system level, *agent management* (**M4**) tools provide a system supervisor the ability to start, stop, suspend, or remove individual agents. Some of the security features fall under

the purview of *distribution services* (**M5**), which encompasses additional aspects of distributing applications, both functional and implementational, possibly including "white pages" (or naming services), "yellow pages", agent migration, and agent management.

The remaining five features we identify are implementation dependent. As a MAS moves toward providing an open multi-agent environment, security concerns grow proportionally and should be explicitly addressed. In addition to **M2** and **M3**, an implemented system might support various forms of *message encryption* (**M6**). MAS often need system-wide monitoring. One aspect of this, similar to SAS, is the need for *logging facilities* (**M7**), which can be applied to *system monitoring* (**M8**; i.e., infrastructure components) or *agent monitoring* (**M9**; i.e., the run-time state of individual agents). A *monitoring graphical user interface* (**M10**) may border on being required if the system is large and/or complex.

Current cognitive architectures (SOAR, ACT-R, EPIC, and others) do not support any of the distributed mechanisms of MAS systems, while some MAS systems support some SAS features. We briefly review two common systems, RETSINA [10] and JADE [1], representative for others.

RETSINA facilitates complex agent interactions by establishing an open environment in which heterogeneous agents can participate. A variety of "components" are linked to form the environment's framework. The *Communicator* provides an abstraction of the physical transmission layer for both synchronous and asynchronous agent communication. An *Agent Naming Service* provides mechanisms that allow location transparency. *Multi-agent Management Services* allow both logging and some amount of control of agent activity. The existence of *Matchmakers* supply *yellow pages* functionality, which provide the means of mediating or brokering agent connections according to services offered by an agent. Finally, the security features found in RETSINA are especially strong. Since the goal of RETSINA is to provide the infrastructure of a multi-agent environment, little support is given for SAS-agent design and implementation.

The Java Agent Development (JADE) Framework conforms to the specifications laid out by the Foundation for Intelligent Physical Agents (FIPA). Being FIPA-compliant carries with it the requirement that JADE define agent management services, a communication channel, an ANS that performs both white and yellow page services (called the *directory facilitator*), a communication language, and various security features. At a conceptual level, the included features are similar to those found in RETSINA. The underlying infrastructure in JADE is composed of *platforms* and *containers*. A platform refers to a coherent instance of JADE that may either be on a single host or distributed across a network. This includes the Java virtual machine(s), the management agents (a manager service, an ANS, a communication channel, and a message dispatcher), and the user-defined agents. A platform is further broken down into at least two "agent containers", which utilize Java's remote method invocation (RMI) mechanisms as RMI server objects responsible for controlling the execution of agents on a single host. This infrastructure is different from RETSINA and other agent systems, where each agent executes as an operating system process. JADE does supply some SAS features such as agent skeletons and the incorporation of the Java Expert System Shell, a rule-based/expert system language.

3 Merging SAS-MAS Systems: The ADE System

Table 1 summarizes the features present in JADE and RETSINA. While both systems provide the infrastructure necessary for agent-to-agent communication, including distribution services, middle-agents, collaborative or competitive agent interaction protocols, and agent management facilities, they do not provide much support for complex SAS-architecture design and implementation. Yet, there is great utility to be gained from directly supporting SAS-agents and their architectures in the context of MAS systems, in particular, when complex MAS-agents can be defined in terms of SAS-architectures whose components can themselves be implemented in terms of MAS-agents.

	Feature	JADE	RETSINA
	S1: Architecture Support	√	-
	S2: Architectural Routines	√	√
	S3: Data Processing Routines	-	-
S	S4: Simulation	-	-
A	S5: Logging Facilities	√	√
S	S6: Concurrency/Threading	√	-
	S7: Programming Lang. Support	√	√
	S8: Device Configuration	-	-
	S9: Interface Abstraction	-	-
	S10: Device Update Frequency	-	-
	M1: Communication Abstraction	√	√
	M2: Distribution Services	√	√
	M3: Logging Facilities	√	√
M	M4: Agent Monitoring	√	-
A	M5: System Monitoring	√	√
S	M6: Monitoring GUI	√	√
	M7: Agent Management	√	√
	M8: Message Encryption	√	√
	M9: Agent Authentication	√	√
	M10: System Access Control	√	√

Table 1: Comparison of two MAS, RETSINA and JADE, with respect to 10 SAS and 10 MAS features.

In the context of the threat detection system, a complex MAS-agent could be a natural language processing agent NLP_L that is capable of providing a language-independent semantic summary of the content of given piece of text in a particular target language L. Multiple instances of NLP_L coexist in the

running system as they are processing large numbers of data streams in parallel. Each NLP_L is implemented in the cognitive architecture SOAR (e.g., using NL-SOAR [6]), where the rule-base is implemented as a MAS-agent and thus sharable among agents. The immediate advantage is that rules learned by one NLP_L agent (e.g., via "chunking" in SOAR) can be used by others.

We have developed a MAS system for embodied real-time systems (like robots or most intelligence gathering applications), called `ADE` [7], which implements all features in Table 1 and directly supports the recursive implementation of components of SAS-architectures in terms of MAS-agents (i.e., components of SAS-agent can consist of heterogeneous MAS-agents, which themselves have SAS-architectures that consists of MAS-agents, etc.). `ADE` comes with a large set of implemented `ADE` SAS components: from *perceptual processing* (including machine vision, laser localization, speech processing and recognition), to *goal and action planning and processing* (including goal and action management, action scripting and sequencing, navigation planners and primitive motor behaviors), to *natural language processing* (including incremental and non-incremental parsers, discourse and dialogue engines, and text generation components), and *planning and reasoning* (including interfaces for linking in complete or partial cognitive architectures, task-planners, and reasoning engines), most of which have been used successfully in our DIARC architecture [9].

The `ADE` server model was designed for potentially hostile insecure distributed dynamic multi-OS computing environments, where connections between any two hosts participating in the `ADE` system cannot be assumed to be secure nor present throughout the lifetime of an application. Consequently, mechanisms are required for `ADE` servers to protect communicated information, detect failures of connections, and recover from failure. Moreover, different components in `ADE` might have different levels of classification and thus require special access control (e.g., operators maintaining the system might be permitted to access the rule-base of an NLP_L agent, but not its working memory that stores sensitive facts). Both requirements are met by `ADE`'s "heartbeat mechanism", which is implemented by every `ADE-Server` server: right after registration with an `ADE-Registry`, an `ADE-Server` starts sending (possible encrypted) status packets at regular intervals. Each status packet contains the server's ID, its unique credentials, which are created dynamically by the registry in response to each packet and need to be included in the followup heartbeat packet, and the status of its currently active connections. When a client requests a server connection, the registry forwards the request to the server passing along the client's ID and credentials. If the request is accepted, the server will return a remote reference to itself to the registry, which is then returned to client, and a separate peer-to-peer heartbeat connection is started between client and server (in addition to the one between server and registry). Whenever a heartbeat times out (i.e., when a packet fails to arrive in time), a special function is called within the MAS-agents on each side of the connection to deal with the failure.[1]

4 Synergistic Features of SAS-MAS Systems

The combination of SAS and MAS mechanisms in a system like `ADE` gives rise to several synergistic features that we believe are of crucial importance to homeland security applications. The first synergistic feature we identify is *Arbitrary-level Access Control*. In a standard MAS, like JADE and RETSINA, it is possible to assign privileges to individual distributed MAS-agents and allow users selected access to them. For example, an analysis agent AA started by an investigator as part of an interactive session (e.g., to determine whether a particular hypothesized relation between two very different data sources exists) will inherit the investigator's privileges. Hence, if AA attempts to request a resource for which the investigator does not have clearance (e.g., access to a database, another analysis agent, etc.) AA's access operation will fail. While it is possible in such a system to restrict access to MAS-agents like AA, it is not possible to restrict access to their constituent components (e.g., to different sets of AA's production rules).

However, such finer-grained access control can be extremely useful and lead to better scalability, performance, and security of an application. This might require another synergistic feature, the ability to use *Shared, Location Independent Components*. For example, if AA is itself distributed, then different instances of AA can share components in the system, which reduces memory, CPU, and storage requirements. Different privileges could then, for example, be assigned to sets of rules that allow for different ways of data combination or special purpose data mining algorithms that connect information within a database in a particular way, and only analysis agents with the appropriate access rights will be able to use them. Moreover, the distribution of components of AA might also improve the agent's performance (e.g., due to parallelization). Sharing components is particularly desirable when resources are specialized and/or limited.

[1] More details on the functionality of `ADE` can be obtained at `http://ade.sourceforge.net/`.

While there two ways to address the component-level access problem in standard MAS, neither of them is satisfactory (although the second also addresses the component-level sharing problem). The first requires the definition of different MAS analysis agents for each access class, thus duplicating programming and maintenance effort and adding run-time resource overhead because common parts of the analysis agents cannot be shared among instances. The second approach uses MAS-agents to define the components of *AA*. However, since standard MAS do not provide much support for SAS-architectures, the distributed SAS architecture of the *AA* will have to be largely developed from scratch and the communication among the different MAS-agents of *AA* will likely be slowed down due to various forms of overhead (authentication, distribution, ACL, etc.). Moreover, explicit bookkeeping of all MAS-agents belonging to one *AA* is required including which MAS-agent can be shared (by whom and at what time). In sum, while it is not impossible to implement Arbitrary-level Access Control on top of the access control provided by standard MAS, both development and run-time overhead will be significant.

To avoid this overhead of reimplementing SAS components from scratch, `ADE` features a rich set of sever APIs for different types of SAS components (as described above) together with reference implementations for each interface, which can either be used directly in an `ADE` system or extended to implement custom functionality. The implemented services can then be `ADE-Server` advertised by an `ADE-Server` as soon as it comes up in an `ADE` system (e.g., a vision processing component for the detection of faces in images) and used by consumers (i.e., other `ADE-Server` s). Access to its services is controlled at the method level by an `ADE-Registry`, which provides MAS security features. A set of `ADE-Server` s and their connections can then form a SAS-architecture of a complex MAS-agent like *AA*, thus automatically allowing for Shared, Location Independent Components (no inherent distinction is made in `ADE` between MAS-agents and components of a SAS architecture in `ADE`).

There are other advantages of the access control in `ADE` that are important for long-term operation and maintenance of distributed applications: components (i.e., MAS agents) common to a large class of more complex MAS-agents (like *AA*) can be easily upgraded in the whole system. One of several possible ways to do this in `ADE` is for the new component MAS-agent to register with an `ADE-Registry`, which, in turn, will deregister the old component agent (whenever an instance of *AA* requires the particular component, it will get an instance of the new type).

The ability to share arbitrary components and provide selective accesses to them is complemented by component-level *Dynamic System Reconfiguration*. This feature is critical for long-term operations for several reasons. For one, because applications that are resource demanding, but cannot continue at a given time due to resource constraints, will be able to continue as long as there are some resources available at some place in the system. Suppose a reasoning engine runs out of working memory on a host computer (e.g., because it has accumulated too many facts in working memory over time) and suppose further that working memory consists of component MAS-agents (e.g., each agent represents a set of facts). Then it is possible for the system to automatically reconfigure itself by moving currently unnecessary working memory entries to other hosts (e.g., by using last access as a criterion analogous to some caching algorithms in operating systems). Other reasons for Dynamic System Reconfiguration are the need to restart components elsewhere due to various failures on hosts (e.g., hardware failures, crashes, or temporary network unavailability) or to improve the load, response time, and reliability of the running application.

In `ADE`, a `ADE-Registry` can monitor the performance of its registered servers and in order to avoid overloading them (e.g., as measured by the server's response time) start a duplicate component, which can be subsequently used (another possibility is re-route existing connections as long as the state of the server can be saved and recreated remotely). Alternatively, an extension to `ADE-Server` functionality might allow dynamic changes to be made to a component's allowable number of connections, forcing requests for that component to be redirected or fail. The heartbeat mechanism can also be used to control *load balancing*, which, in some sense, can be viewed as a controlled component failure and recovery, where the component is restarted on a different host. This not only requires mechanisms for maintaining a component's state, but also a means of "forwarding" connections to the component's new location. Used in this way, `ADE` is able to perform *auto-migration*, making the system as a whole more robust and increasing performance.

5 Summary

Each of the three synergistic features, *Arbitrary-level Access Control*, *Shared, Location Independent Components* and *Dynamic System Reconfiguration*, blurs the separation between agent and system architecture levels, giving SAS-architecture components, whenever necessary and useful, MAS-agent capabilities. Different from other MAS systems, `ADE` allows for a direct mapping between a common SAS-architecture to its distributed implementation. In fact, in the sim-

plest case, ADE allows for a distribution of an otherwise non-distributed SAS architecture (for example, we have implemented various distributed action selection mechanisms for different robotic architectures in ADE [8]). In more complex cases, MAS agents can have architectural components that are themselves MAS agents, which themselves are MAS agents, etc.

We believe that the three synergistic features described above are only a small set of the kinds of architectural mechanisms that a combined SAS-MAS system like ADE can potentially support. For example, to implement a form of error recovery, we have used the heartbeat mechanism to detect component failure, triggering a mechanism that restarts it on the same or a different host [3]. The substitution of one instance of a component for another allows a "fixed" individual architecture to transparently take advantage of distributed components by a simple redirection of the request for connection.

Testing new components that should replace existing ones while the system is running is a more substantial example that relies on the three synergistic features. This can be accomplished by limiting access to components to authorized system maintainers and designers, who request a second set of connections to the requisite, shared ADE-Server s. The new component runs simultaneously and in parallel with the old one, taking in all the inputs of the old component, but without having its outputs effect changes in the system. Both components are monitored until testing has confirmed the proper function of the new component. The old component can then be removed and the system dynamically reconfigured without interruption of overall system function.

Besides identifying and implementing more synergistic features, future work with ADE will include the development of a prototype threat detection system similar to the one described in the introduction. The system will employ distributed natural language processing modules together with probabilistic inference and evidence combination mechanisms and utilize error recovery and "self-healing" mechanisms incorporated into ADE to satisfy the demands for reliability, security, and long-term operation of large-scale autonomously operating intelligence systems.

Acknowledgements

This work was in part supported by ONR grant #N00014-10-1-0104. Thanks to Jim Kramer for his help with the analysis of SAS and MAS systems and its features.

References

[1] F. Bellifemine, A. Poggi, and G. Rimassa. JADE - a FIPA-compliant agent framework. In *Proceedings of the PAAM99 Conference*, 97–108, London, April 1999.

[2] Foundations for Intelligent Physical Agents. FIPA agent management specification (sc00023k). World Wide Web, 2004.

[3] James Kramer and Matthias Scheutz Reflection and Reasoning Mechanisms for Failure Detection and Recovery in a Distributed Robotic Architecture for Complex Robots In *Proceedings of the 2007 IEEE International Conference on Robotics and Automation*, 3699–3704, Rome, Italy, Apirl 2007.

[4] N. R. Jennings, K. Sycara, and M. Wooldridge. A roadmap of agent research and development. *Journal of Autonomous Agents and Multi-Agent Systems*, 1(1):7–38, 1998.

[5] K. Premaratne, M. N. Murthi, J. Zhang, M. Scheutz, and P. H. Bauer. A Dempster-Shafer theoretic conditional approach to evidence updating for fusion of hard and soft data. In *Proc. International Conference on Information Fusion (ICIF'09)*, 2122–2129, Seattle, WA, July 2009.

[6] C. A. Rytting and D. Lonsdale. Integrating wordnet with nl-soar. In *WordNet and other lexical resources: Applications, extensions, and customizations*, 162–164. North American Association for Computational Linguistics, 2001.

[7] M. Scheutz. ADE - steps towards a distributed development and runtime environment for complex robotic agent architectures. *Applied Artificial Intelligence*, 20(4-5), 2006.

[8] M. Scheutz and V. Andronache. Architectural mechanisms for dynamic changes of behavior selection strategies in behavior-based systems. *IEEE Transactions of System, Man, and Cybernetics Part B: Cybernetics*, 34(6), 2004.

[9] Matthias Scheutz, Paul Schermerhorn, James Kramer, and David Anderson. First steps toward natural human-like HRI. *Autonomous Robots*, 22(4):411–423, May 2007.

[10] K. Sycara, M. Paolucci, M. V. Velsen, and J. Giampapa. The RETSINA MAS infrastructure. *Autonomous Agents and Multi-Agent Systems*, 7(1):29–48, 2003.