Comparison of Two Component Frameworks: The FIPA-Compliant Multi-Agent System and The Web-Centric J2EE Platform

Michelle Casagni Margaret Lyell
The MITRE Corporation
{ mcasagni, mlyell } @mitre.org

Abstract

This work compares and contrasts two component frameworks: (1) the web-centric Java 2 Enterprise Edition (J2EE) framework and (2) the FIPA-compliant multi-agent system (MAS). FIPA, the Foundation for Intelligent Physical Agents, provides specifications for agents and agent platforms. Both frameworks are component frameworks; servlets and Enterprise Java Beans (EJBs) in the case of J2EE and software agents in the case of MAS. Both frameworks are specification based. Both frameworks mandate platform responsibilities towards their respective component(s).

We develop a framework with which to structure the comparison of the component frameworks. We apply this comparison structure in the context of a 'Data Access' scenario to application development in the respective component frameworks. Furthermore, we have prototyped this scenario in each of the two component frameworks. We conclude with a discussion of the benefits, drawbacks, and issues of developing new applications in each of the component frameworks.

1. Introduction

Recent work by Griss and Pour [10] advanced the thesis that agent components are a potential foundation for "flexible, intelligent, Web-based enterprise application systems". In contrast, a leading contemporary component-based Web-centric architecture is that of the Java 2 Enterprise Edition (J2EE) platform. The question of how applications developed for the Web-centric, client-server, n-server tier, component-based architecture that is prevalent today will interoperate with applications developed in a software agent framework is the larger question that motivates our efforts. In this work, we develop a framework for comparing and contrasting the features of two classes of component frameworks: (1) the client-server, Web-based component architecture and (2) the software agent framework. In our work, we take the J2EE platform as the exemplar of the former. We consider only software agent components and multi-agent system (MAS) frameworks that are compliant with specifications set forth by the Foundation for Intelligent Physical Agents (FIPA). There are similarities between these two specification-grounded component frameworks. Obviously, both frameworks include components; servlets and Enterprise Java Beans in the case of J2EE and software agents in the case of FIPA-MAS. Both frameworks mandate platform responsibilities towards their respective component(s). There are also obvious differences in the nature of the components. For example, the paradigm that is software agents resides at a higher semantic level than that of the EJB or servlet component. Software agents have additional properties such as task-orientation, autonomy, and social-ability (communicative ability and co-cooperativeness) [23].

Efforts reported in the literature have involved disparate assessments of each of the individual component frameworks. EJBs have been evaluated against the quality attributes of (1) modifiability, (2) performance, and (3) availability in the work of [15]. With regard to software agent frameworks, previous assessments of MAS have not focused on FIPA-compliant systems nor have they utilized FIPA specifications. Rather, an architectural evaluation of MAS, organized by design patterns, has been given by [22]. This work considered the effect of MAS behavior on attributes of: (1) performance predictability, (2) security, (3) resiliency to modifiability of the environment, and (4) availability.

We develop a framework with which to structure the comparison between the J2EE framework and the FIPA-compliant agent framework. Our 'comparison structure' is informed by previous efforts that dealt with single framework assessment. We include consideration of design patterns and (an expanded set of) quality attributes as features in our comparison. Additional features in our comparison structure are a set of Design Properties for components and component-based applications. We discuss how these propagate into the quality attributes. (The propagation concept was put forth in another context by Bansiya [2].) Our comparison structure also considers the framework features that support component hosting, as well as design ramifications for the components due to framework considerations.

We apply the comparison structure in the context of a 'Data Access' scenario in order to highlight the similarities and differences that arise in the same application developed for the two different component frameworks. Additionally, we prototyped the 'Data Access' scenario in each of the component frameworks. The prototyping effort permits application of the 'resource consideration' and 'performance' elements of the comparison structure to each of the component-based solutions. Furthermore, the prototyping effort can potentially reveal absent features of the comparison structure.

2. Comparison structure

2.1. Comparison elements

The comparison structure consists of three sets: (1) Design Property Features, (2) Framework Impact Features, and (3) Quality Attributes. Design properties subdivide into those that pertain to a single component and to those that pertain to the overall application (composed of multiple components). We assume that the components in each of the frameworks have been developed in accordance with established Object Oriented (OO) design principles, and do not address this level. Use of a J2EE platform implies that component development will utilize Java. This situation does not strictly hold in the realm of FIPA-compliant software agent platforms. However, the vast majority of FIPA-compliant software agent platforms are developed in Java; individual agent development largely employs Java.

Comparison of the two frameworks draws on the J2EE specifications [14] and the FIPA specifications [8]. Elements of the comparison structure also require information pertaining to a platform implementation of the two respective frameworks. We use WebLogic (v6.1) and FIPA-OS (v2.0) as the exemplar implementations in Java. Other Java implementations of the FIPA specifications exists; for example, JADE [12]. However, this work is one thread of a larger effort that is utilizing FIPA-OS. (WebLogic is an established commercial platform [3]. FIPA-OS [9] is an open source reference implementation of the FIPA specifications developed by an industry team. The industry team and a developer community continue to add enhancements.)

The first set of elements in the comparison structure concerns Design Properties. Elements of this set, with their definitions are:

- Design Properties, Application Level
  - *Design Size*: Number of components in the application.
  - *Functional Cohesiveness*: A measure of how the aggregate components provide a unified 'application' to the user.
  - *Coupling*: Interactions among components that form an application.
- Design Properties, Component Level
  - *Inheritance / Templating*: Presence of base level requirements that must be satisfied in order to be a component, along with rules on component instantiation.
  - *Encapsulation*: A measure of how well the component is concerned with one 'topic'.
  - *Complexity*: A measure of the depth and breadth of a component's functionality.

The second set of elements in the comparison structure concerns Framework Impact issues. Elements of this set are:

- Framework Features supporting Component Hosting
  - *Platform Configuration Support*
  - *Descriptive Files*
  - *Packaging*
  - *Deployment*
  - *Services Offered by Platform to Component*
- Design Ramifications for Components due to Framework Considerations
  - *Component Access*
  - *Resource Restrictions Placed on Components by Platform*
  - *Design Pattern Usage*

The third set of elements in the comparison structure involves the Quality Attributes. Elements of this set, with their definitions are:

- *Availability*: Properties that promote or detract from the component or system's ability to perform its function.
- *Usability*: Assessment is from three viewpoints:
  - *Developer POV*: Properties that promote or detract from component development.
  - *Deployer POV*: Properties that promote or detract from component deployment.
  - *User POV*: Properties that promote or detract from application usage.
- *Modifiability*
  - *Maintainability*
    - *Extendibility*: Properties that support or hinder component or system modifications in order to incorporate new requirements.
    - *Reusability*: Properties that support or hinder a component's use in a new case without excessive effort.
  - *Flexibility:* Properties that promote or hinder the system's ability to respond to environmental changes (resources no longer available) without affecting the application's defined functionality.

- *Resource consideration*: Infrastructure-supplied resources expected to be utilized by the application.
- *Performance: (Restricted definition)* Timing measurements.

2.2. Application of comparison structure to component frameworks

Space limitations preclude an exposition of the full comparison structure applied to the two component frameworks. We present only selected elements in the comparison of the FIPA-compliant agent framework and the J2EE web-centric framework. For additional details, see [4].

A metric for *design size* is given by counting both the number of EJBs and servlets in the application hosted by the J2EE Platform, and the number of software agents in the multi-agent system that participate in the application. *Coupling*, another application level design property is highlighted in Table 1. The metric in table 1 yields the worst case for coupling between the components.

With regard to *inheritance/templating,* a component level comparison element, there are EJB specifications [7] regarding which interfaces must be implemented in order to develop {Entity, Session, or Message-Driven} EJBs. These interfaces extend specific 'javax.ejb' classes. Consider, for example, a Session EJB 'YYY'. In order to construct a Session Bean, a Remote interface and a Home interface must be defined. Remote interface 'YYY' must extend javax.ejb.EJBObject. Home interface 'YYYHome' must extend javax.ejb.EJBHome. The implementation 'YYYEJB' must implement javax.ejb.SessionBean.

In contrast, the FIPA specifications do *not* address this issue. Rather, the issue of agent templates or inheritance is the purview of the specific FIPA-compliant agent platform. For the case of the FIPA-OS platform, an application agent can extend a FIPA-OS agent. Specific tasks that an agent performs can extend the FIPA-OS Task class. Agents can also be part of an inheritance chain; the leaf agent is non-abstract.

Design patterns are used for application development in both of the component frameworks. In software agent systems, design patterns offer an approach to organizing roles and relationships in the multi-agent society. Common design patterns include the Mediator or Broker patterns, the Wrapper pattern, and the Peer-to-Peer pattern [11]. Within the J2EE framework, design patterns are employed to facilitate the connection of one component to another, potentially across tiers. Several design patterns have been mentioned in Table 1. Additional common patterns [20, 21] are the Service Locator pattern, used in a Session EJB, to hide the complexity of JNDI lookup calls, and the Model, View, Controller (MVC) pattern used in

Table 1. Design property: coupling

	EJBs and J2EE Framework	**Software Agents and FIPA-compliant MAS**
Definition Refinement	Interactions among EJBs or among servlets and EJBs.	Interactions among software agents in a MAS.
Comments	Interactions are via method invocation on business methods. (Business methods are defined in the Remote Interface.) HttpServlets are invoked by Get or Post methods from the client.	Interactions between two agents are via messaging. Messages are exchanged according to Interaction protocols. (FIPA defines standard semantically meaningful performatives and Interaction protocols.)
Issues	DESIGN ISSUE: Business Delegate and Façade Design patterns seek to minimize direct exposure of an EJB's business methods (to general clients). This controls granularity of access to EJBs. Entity EJBs may interact with each other. This is not desirable; it is mitigated via use of the Aggregate Entity Design pattern.	DESIGN ISSUE: Several agents in the MAS may form a sub-group. Communication to the sub-group may be restricted to only one agent in the sub-group.
Metric	Sum of [the number of Business Methods of each EJB multiplied by the number of remaining EJBs.] (Upper bound)	Sum of [number of Interaction protocols that an agent understands multiplied by the number of potential communicating agents in the MAS.]

servlets to separate the control logic from the presentation logic.

In the area of framework-driven design ramifications, specifically that of *component access* and *resource restrictions placed on components by platform*, it is noted that framework constraints are more severe on EJBs than on FIPA-compliant software agents. The J2EE application container mediates access to EJBs. An EJB should not act as a network server, nor listen, accept or multicast on a socket [1]. Furthermore, the application container manages resources, such as file I/O and threading, for the components. Resource restrictions placed on the EJB component affects design decisions, as will be evidenced in Section 3. In contrast, an agent in a FIPA-compliant MAS is free to use whatever resources it needs.

Another discriminator between the two component frameworks is the degree to which they support component hosting. In the area of *packaging*, the J2EE framework provides for servlets and other web components to be packaged in a '.war' file. EJB '.jar' files and relevant '.war' files can be packaged together in an '.ear' file. FIPA specifications do not address 'packaging' of software agent components that together form an application or offer functionality.

Table 2 looks at the *services that are offered by platforms* in each of the frameworks. The FIPA-compliant agent platform excels at offering a software agent component support for advertising its services and support for dynamically finding other agents and the services they offer. The J2EE platform excels in offering managed resources, such as database connections and Java Message Service destination channels. The J2EE platform, through the application container, also offers the Entity EJB components managed mapping of its data to database tables.

The quality attribute of *usability* is considered from multiple vantage points. With regard to the end-user, applications hosted in the J2EE platform generally offer the user a web-browser interface. Software agent systems do not offer a standard human-user interface to the system. Rather, the FIPA specifications focus on software agent to software agent interactions. Good functional cohesiveness of EJB components, together with appropriate design pattern usage, allows presentation of a unified application to a user. Reduced coupling among EJBs improves performance for the end-user.

From a deployer point-of-view, the J2EE platform offers standard directories for deploying '.war' files into the web container, and for deploying '.jar' and '.ear' files into the application container. Such files contain the Java classes that comprise the components. It is the FIPA-OS platform, rather than the FIPA specifications, which addresses software agent deployment. An agent loader allows automatic deployment of agents. A control GUI is also available for manual sequencing of agent start-up. Executable scripts can be used to control startup of individual agents, and allow for command line parameter passing. In the FIPA-OS agent system, attention is given to the mechanics of deployment, but not the organization of the deployment.

Table 2. Framework support for component hosting: services offered by the platform to the component

J2EE Platform	EJB lifecycle management, network access, security, transaction management, resource management, container managed persistence (Entity EJB). Platform offers servlet threading. Lookup provided by JNDI API.
FIPA-Compliant MAS Platform	Agent Management System (AMS) provides lifecycle management, white pages services. Directory Facilitator (DF) provides yellow pages services. Message transport services, for messages between agents on single FIPA-compliant platform and to other FIPA-compliant platforms. FIPA specifications provide a Service Description (SD) for any agent wishing to advertise its services with the DF. [The SD consists of descriptive {name,value} property pairs as well as sets of Interaction protocols understood by the agent/agent-service, set of ontologies used, and sets of agent content languages understood.] **NB: Specific to FIPA-OS** Conversation Managers and Task Managers are provided.

The deployer does not receive any assistance in the identification of resources from the FIPA-compliant software agent framework. The resources that a software agent must utilize, such as a JMS topic channel or a database connection, are not linked to the software agent in any standard manner. This is in contrast to what is offered by the J2EE specifications; resources required by components must be specified in standardized xml deployment descriptor files.

In MAS, a distinction is drawn between open and closed multi-agent systems. In an open MAS, a new software agent component joins the system; dynamic discovery by other agents permits the new agent to offer its services and be useful. However, the FIPA specifications do not require that an agent provide the necessary information for effective usage as part of its Service Description advertisement. For example, the agent is not required to list the Interaction protocols that it speaks. In a closed MAS, all of the software agent components and their Interaction protocol usage are considered known at design time.

These issues have implications for both *availability* and *reusability* in a FIPA-compliant MAS. Consider a scenario in which the "usual" agent offering a service becomes unavailable. A search for an alternative agent that offers the service is made using only key words for the service as search parameters. (The search can span multiple agent platforms.) Initial search results identify two other agents, SA1 and SA2, that offer the service. Neither SA1 nor SA2 advertise the Interaction protocols that they speak. Agent SA1 speaks a different Interaction protocol from the one that the requesting agent knows, but the requesting agent finds this out only when it communicates with SA1 and its request is not understood. At this point, the service is still unavailable to the requesting agent. The requestor then turns to agent SA2; it does speak the same Interaction protocol and handles the request. If FIPA specifications required agents to list the Interaction protocols they understand as part of their Service Description, requesting agents would be more likely to include Interaction protocol information in their search parameters. This could lessen the period of service unavailability.

With regard to *reusability*, in order to determine if an existing software agent component could be utilized in a new application, the developer has to inspect the code. The developer gains information as to what Interaction protocols, agent content languages and ontologies the software agent understands only by inspecting the code.

In contrast, the J2EE framework enhances component *reusability* and *availability*. The Remote Interface of an EJB lists the business methods that may be invoked on that component. This facilitates the use of that component in a new application. Failover strategies are not mandated by the J2EE specifications. However, commercial vendors do address this topic; enhancing the availability of an EJB component.

This section concludes with presentation of additional factors that affect *maintainability* in Table 3. In Section 4, two elements of the comparison structure, *resource consideration* and *performanc*e, are applied to each of the component frameworks in the context of a 'Data Access' scenario. This scenario has been prototyped and is described in the next section.

Table 3. Maintainability

	Component Level	**J2EE Platform Level**
Servlet	Excessive coupling (to EJBs) detracts from reuse; mitigated by use of MVC pattern. Good encapsulation facilitates maintainability.	Use of logical name and JNDI promotes reuse. Web container offers support to servlets, JSP's and HTML pages; promotes extendibility.
Enterprise JavaBean (EJB)	EJBs are portable among different vendor implementations of J2EE; promotes reusability. Good encapsulation facilitates maintainability. Encapsulation of JNDI information in Session EJB, and use of Service Locator Design pattern promotes maintainability.	Use of logical name and JNDI promotes reuse. Application container offers middleware support to EJBs; promotes reuse.
	Component Level	**FIPA-Compliant Agent System Level**
Software Agent	Good encapsulation, with clearly defined agent role, promotes agent reuse. An agent role with multiple responsibilities and complex task structure is not easily extended. User-defined Interaction protocols detract from agent reuse.	No requirement to provide elements of Service Description hinders reuse. Registration and searching functionality are supported by the system, supports adding new agents to meet new requirements (extendibility).

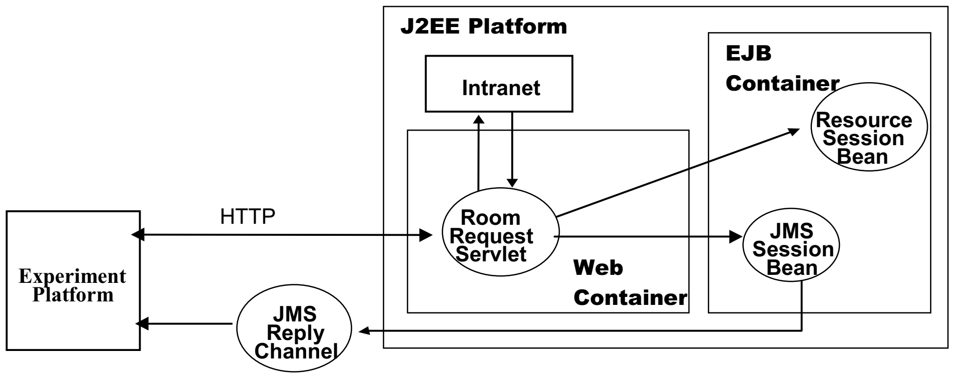

Figure 1. J2EE Hosted System Architecture

3. Data access scenario

A ‘Data Access’ scenario is used to provide a set of requirements for designing and implementing an application in both the J2EE platform and a FIPA-compliant Multi-Agent System. The experimental prototype further elucidates differences between the framework architectures and allows us to obtain basic performance results.

Users of the Experiment Platform, a pure Java application that is not based on agent frameworks or J2EE technology, are collaborating via a chat application and need to access information not available to them locally. To provide a reasonable example of data access from another platform, the experiment consists of answering a request based on both time sensitive information (changes frequently) and static information. Our scenario is to find a room available at a specific time that contains certain equipment and holds a specific number of people. The availability of a room is determined by searching an Intranet and the resources for each room are retrieved from a local data store.

In part I of the ‘Data Access’ experiments, a “Find Room Request” is generated by the Experiment Platform application. This request is processed by an application that is hosted by the J2EE framework. In part II of these experiments, the “Find Room Request” is processed by an application developed in a FIPA-MAS framework.

3.1. J2EE hosted application analysis and design

The analysis and design of the J2EE-based system began with a determination of which modes of communication to use between the Experiment Platform and the J2EE framework components. Using typical design choices for component development as a guideline, we iterated the identification of components in the framework to host the object functionality as we considered various communication options. Object-oriented analysis and design techniques as discussed in [19] were then used to design the objects, object attributes, relationship among objects and behavior of objects.

Resource restrictions placed on the components by the platform affected the design. For example, guidelines for developing Enterprise Java Beans [1] preclude them from accepting socket connections or listening on a port. Therefore, these components cannot listen directly for messages from the Experiment Platform. A Message-Driven Bean [7] could be hosted by the J2EE platform to receive requests from the experiment platform using the Java Message Service (JMS) [1]. To satisfy the scenario requirements, this approach requires an EJB to use the java.net.URLConnection class in order to check the schedule on the Intranet. The development guidelines do not restrict a URLConnection resource in an EJB, but this is not a typical usage.

Servlets, on the J2EE platform, are typically involved in a synchronous, request-response communication using the HTTP protocol. We wanted to design an asynchronous request-response to allow for the possibility of the “Find Room Request” not being fulfilled immediately. To provide an asynchronous solution, both HTTP and JMS were selected for communication between the Experiment Platform and the application hosted by the J2EE framework. HTTP was selected as the protocol for contacting the servlet, rather than JMS. This was for two reasons. First, servlets are typically HTTP servlets. Secondly, the effective use of JMS in a servlet requires session pooling, development of a new servlet engine and potentially modifications to the web container. We did not undertake this level of re-engineering. More recently, a Jakarta Commons Sandbox component, Messenger [18], is addressing this issue. (Note that the Sandbox area of the Commons project contains experimental (Not-Yet-Proposed) components [13].)

The design process involved mapping the entities from the Analysis phase into the proper components in the J2EE framework. UML sequence and class diagrams

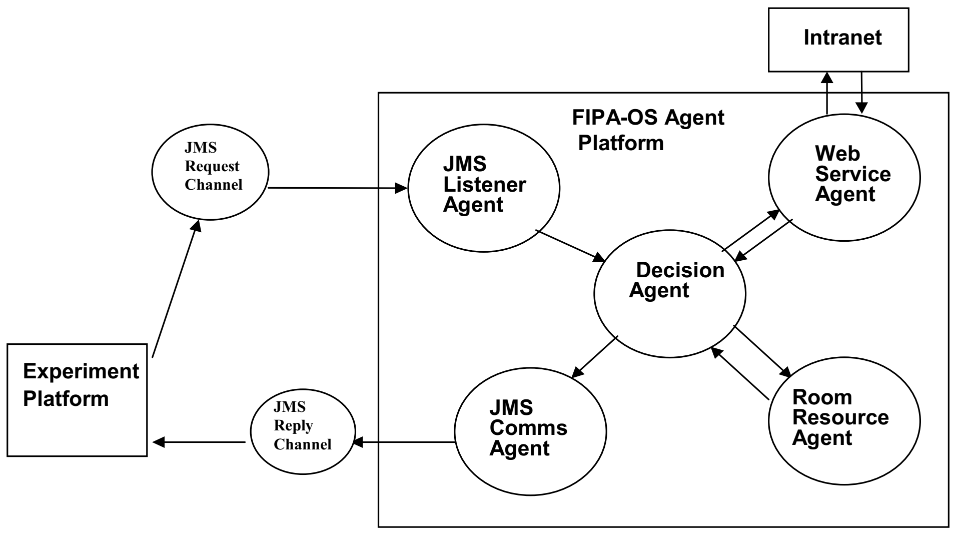

Figure 2. Multi-Agent System Architecture

were created to assist in this process. The final components of the J2EE application (Figure 1) consist of a Servlet and two Enterprise Java Beans. The servlet receives the “Find Room Request”. It contains the logic to satisfy the request, but must invoke methods on the EJBs in order to process it. One EJB acts as a local datastore by holding static room data. It compares the static room resources against the requested resources. The other EJB generates and publishes the response as a JMS message. These operations do not require state, therefore, these components have been implemented as stateless session EJBs.

3.2. Multi-agent system application analysis and design

A similar analysis of communication modes was performed for the multi-agent system. Options presented by sockets, HTTP or JMS solutions presented no issues regarding resource usage restrictions according to the FIPA specifications. However, agents in a multi-agent system inherently communicate asynchronously. We wanted to develop the multi-agent system application according to common usage patterns. Thus, JMS is the optimal choice to use for the asynchronous communications. Previous demonstrations have shown that an agent in the FIPA-OS system can send JMS messages [16]. The JMS Comms Agent developed for that demonstration is reused in this multi-agent system.

We employed a combination of the Multi-agent Systems Engineering (MaSE) methodology [5] and the Gaia methodology [24] for the analysis and design processes of the software agents in the multi-agent system. Note that for this prototype, we are developing a "closed" agent system. That is, all of the agents, with their responsibilities, resource needs, activities, and Interaction protocols, are known at the design phase. We began by capturing the system requirements as goals in a goals-hierarchy tree using the MaSE approach. These goals provided a task-oriented view of the system that laid out a foundation for describing the roles and responsibilities.

Table 4. Roles and Responsibilities Model

Roles	Responsibilities
Message Sender (=> JMS Comms Agent)	• Send “find room” results
Message Receiver (=> JMS Listener Agent)	• Receive “find room” requests and forward to Decision Maker for processing
Decision Maker (=> Decision Agent)	• Parse the “find room” request and obtain information to process it • Analyze the information to determine which rooms meet the request criteria • Generate the “find room” reply
Room Resource Holder (=> Room Resource Agent)	• Contain the list of resources for each room • Provide a list of rooms that meet the resource requirements
Web Service Provider (=> Web Service Agent)	• Search the Intranet for the room schedule • Provide a list of rooms available at the specified time

We followed an approach described by the Gaia methodology to generate the analysis phase products: a Roles model (Table 4) and an Interaction model. Each role in the Roles model has associated responsibilities. These roles evolved into five software agents in the final design (Figure 2): JMS Comms Agent, JMS Listener Agent, Decision Agent, Room Resource Agent, and Web Service Agent. The responsibilities are modeled as tasks. The interactions between the agents are in accordance with the FIPA Request Interaction Protocol.

The Interactions model revealed several design patterns. Roles within the system only interact with the Decision Maker role. This behavior is similar to that of the Mediator pattern as described by Hayden [11]. From the roles Message Receiver and Message Sender, we identify an agent design pattern that allows for an interface to a different framework. These roles, implemented as the JMS Listener Agent and the JMS Comms Agent, act as gateways into and out of the multi-agent system. The Gateway pattern is described in [4].

4. Application of comparison structure to prototype

4.1. Design properties

Design property metrics were applied to each application design solution of the 'Data Access' scenario. The *design size* of each application in the two component frameworks is roughly the same. The J2EE framework solution design size is three, with one servlet and two EJBs. The FIPA-compliant multi-agent solution design size is five, with five individual agents.

The *design patterns* employed in each approach highlight the *coupling* within that framework. In the J2EE framework, the servlet controls processing of the request. It invokes the stateless session EJBs to retrieve information for finding a room. In doing so, the servlet implements a Controller pattern [20]. Similarly, the Mediator pattern and the Gateway pattern organize agent interactions within the multi-agent system. The Decision Agent is responsible for interacting with other agents in the system to fulfill the "Find Room Request". Communication into and out of the multi-agent system is through the gateway agents.

All of the components in each framework use *inheritance/templating*. The servlet in the J2EE framework inherits from javax.http.servlet.HTTPServlet. The EJBs extend the javax.ejb.EJBHome and javax.ejb.EJBObject classes for their home and remote interfaces and implement the javax.ejb.SessionBean class. In the multi-agent system, each agent extends the FIPA-OS Agent class and specific tasks extend the FIPA-OS Task class.

The servlet in the J2EE framework *encapsulates* the logic of processing a "Find Room Request" by controlling the entire process of fulfilling the request. The EJBs *encapsulate* a single business process; one EJB searches the room resources and the other generates and publishes the JMS reply. Within the multi-agent system, each agent *encapsulates* its roles and responsibilities through the use of tasks.

Both the applications developed for the 'Data Access' scenario demonstrate *functional cohesiveness*. The servlet in the J2EE framework provides a single point of entry into the application. The EJBs provide the servlet with the additional functionality required to process the request in its entirety. Likewise, in the FIPA-compliant multi-agent system, the JMS Listener Agent provides a single point of entry into the system. The Decision Agent relies on the other agents within the system in order to complete the "Find Room Request".

4.2. Performance

Basic performance tests were done on the prototype applications. The end-to-end times for the "Find Room Request" from the Experiment Platform to the J2EE framework, and from the Experiment Platform to the FIPA-compliant MAS were measured. The networking for the prototype extends over a subnet. JMS and the J2EE framework are hosted on one machine. A second machine hosts the Experiment Platform and the MAS.

Overall, the application in the J2EE framework processes the request faster than the application in the multi-agent system. Most of the delay experienced using the multi-agent system is due to a 'Wait Task'. An agent discharges its responsibilities through execution of asynchronous tasks. Wait tasks are typically designed for use in situations such as polling, searching or acquiring a resource from another agent.

We used a 'Wait Task' to poll the incoming message queue. We experimented with 'Wait Task' times ranging from 100 ms to 10,000 ms; although it is unlikely that 100 ms would be used in practice. More typical 'Wait Task' times are between 5,000 and 10,000 ms. The 'Wait Task' time must be adequate for the agent system application. Wait tasks designed for polling could result in thrashing with too small a wait time. Conversation timeouts could result from too long a wait time when agents are waiting for search results or resources.

Table 5 contains some of the timing results obtained for the 'Data Access' prototypes. Each set of measurements is the average of ten experimental runs, with standard deviation indicated. The full set of performance results is given in [4].

Table 5. Performance Results

	Experiment Platform to J2EE Hosted Application	Experiment Platform to FIPA-MAS (Wait Task 5,000 ms)
Set 1	177 ms +/- 48 ms	3,021 ms +/- 1,586 ms
Set 2	134 ms +/- 37 ms	3,133 ms +/- 1,714 ms
Set 3	213 ms +/- 30 ms	3,202 ms +/- 1,021 ms
Set 4	306 ms +/- 184 ms	3,017 ms +/- 937 ms
Set 5	229 ms +/- 66 ms	2,932 ms +/- 1,557 ms
Set 6	137 ms +/- 21 ms	2,106 ms +/- 1,543 ms

4.3. Resource usage

Infrastructure resources utilized by the respective applications to process the "Find Room Request" in both the J2EE framework and the FIPA-compliant Multi-Agent System are listed in Table 6.

Table 6. Resource Usage

Resources Used	WebLogic 6.1 J2EE Platform	FIPA-OS Agent System
Comm (transport) protocols from/to user	JMS, HTTP	JMS
Components	Control/Decision Logic Servlet Resource EJB (stateless session bean) JMS-EJB (stateless session bean)	JMS Listener Agent JMS Comms Agent Decision Agent Resource Agent Intranet Agent
Deployment Components	Ejb-xml files (deployment descriptors, dd) Web-xml files (dd) War and EJB jar files (executables)	Bat Files Agent Loader Descriptor File
Framework Infrastructure Elements	Web Container Application (EJB) Container JMS Service	FIPA-OS platform with 2 agents: Agent Management System (AMS) and Directory Facilitator (DF) JMS Service

4.4. Developer experience

One of the authors (M. Casagni) served as the developer of the J2EE components and the software agent components. Her key development experiences are discussed in this section.

She was an experienced Java/J2EE developer, but did not have prior exposure to software agents and FIPA specifications. The detailed design provided a solid baseline, and she found implementation of the J2EE system straightforward. However, developing the prototype revealed some issues that were not apparent at design time. For example, use of a property file for global information such as URLs appeared to be a good design idea. Since guidelines indicate that EJBs should not access files, this required many parameters to be passed to them.

The property file was not an issue in the MAS as FIPA specifications indicate agents can access any resources they need. The Room Resource and Intranet Search components for both systems had similar designs, so she was able to reuse the business logic code in the respective agents. The parallel processing capabilities of a MAS allowed for the subtasks of scheduling a room and comparing the room resources to occur in parallel. In the J2EE system these tasks were performed sequentially.

The developer found the asynchronous, concurrent task structure inherent in agent design to be challenging. In particular, coding of the Decision Agent required more time and effort as she had to determine how to track the completion of multiple subtasks before initiating later tasks. Another implementation challenge was in properly incorporating the JMS resource into the agent task structure. The successful method involved having the Agent implement the MessageListener and execute a WaitTask. Whenever a JMS message was received, it was placed on a local list that was processed each time the WaitTask completed.

5. Related work

We are not aware of other work that performs a side-by-side comparison of a web-centric architecture, the J2EE framework, to that of a FIPA-compliant software agent system. However, assessments have been done on the individual frameworks. Two such efforts were referenced in the Introduction section.

A recent effort due to Eiter and Mascardi [6] serves to compare software agent development environments (ADE). In general, the ADEs selected for this study were not FIPA-compliant. The authors determine a set of criteria for evaluating agent development environments and apply these criteria to several development tools and multi-agent system application scenarios. The end result is a set of guidelines for selection of a multi-agent system development kit.

6. Conclusions and future directions

We have developed a ‘comparison structure’ in order to elucidate the differences and similarities between the two component frameworks in a principled manner. We have applied this ‘comparison structure’ to a ‘Data Access’ scenario. In this section we highlight some of the results of our structured comparison between the two frameworks.

Certain similarities relate to application design. For example, an application should involve communication among its components, but excessive coupling is inconsistent with good design in both frameworks. Details in the realization of good design of components within the two frameworks do differ. For example, EJBs are single threaded, with their lifecycle managed by the container. Software agents are generally multi-threaded to reflect their task structure.

A difference in typical component usage occurs across the two frameworks. In the case of the J2EE framework, the servlet controls the communication to the EJB components, invoking (methods in) them in a sequential pattern. In the MAS, the interaction among components is asynchronous and can be done in parallel. For example, in the 'Data Access' prototype, the communication between the Decision Agent and the Resource Agent, and the Decision Agent and the Intranet Agent are done in parallel. The MAS framework is more suited to parallel processing than is the J2EE framework. (Of course, the application intrinsically must contain elements that can be parallelized.)

The MAS framework is more supportive of an 'open system' configuration, involving new software agents that join an established group at run-time. The FIPA-MAS infrastructure mandates a Directory Facilitator that supports lookup for services based on an ontologically-grounded Service Description. However, the FIPA-MAS framework does not mandate all infrastructure elements that would be useful in an 'open system'. For example, an ontology server is not required.

A 'closed system' MAS is closer to the J2EE framework than the 'open system' MAS. In this case, the software agent components are known at application design time, as are the EJB and servlet components. However, the J2EE framework specifications offer more support to the application developer, especially one attempting to reuse components. The Remote Interface must be specified for an EJB, thus exhibiting the business methods of that EJB. The FIPA specifications do not mandate that a software agent exposes the Interaction protocols that it understands. A developer must resort to the code base in order to determine them.

The J2EE specifications offer more support to the system deployer. These specifications offer a standard structure for component deployment and for component-resource association. Resource usage, such as database connections or JMS topic channels, is managed by the J2EE platform. Neither component deployment nor component-resource association is addressed by FIPA specifications. However, the FIPA-OS agent platform does provide an agent loader for deployment.

There are a number of commercial implementations of the J2EE specifications. As a result, the reliability of the J2EE platform is greater than that of the FIPA-MAS implementations. Performance benchmarks are available for different vendor implementations of the J2EE platform, thus enhancing predictability of performance. In implementations of the FIPA specifications, reliability is the responsibility of the application developer. Performance studies of such implementation platforms are sparse. A recent study investigated the FIPA-OS platform [17]. Performance tuning can involve fine-grained tuning of an individual software agent component's task parameters.

If an application is inherently parallel, or if the majority of communications within the system will be asynchronous, or if the application is conceived to be an open system which will grow dynamically, then the designer is advised to consider the FIPA-MAS framework. In doing this, the system designer will have to provide for reliability. Selecting a FIPA-MAS approach will also require more performance testing. System deployment procedures will also have to be developed.

We plan to apply the ‘comparison structure’ to other scenarios. Also, efforts are planned in the area of MAS infrastructure and reliability.

7. References

[1] Allamaraju, S., K. Avedal, R. Browett, J. Diamond, J. Griffin, M. Holden, A. Hoskinson, R. Johnson, T. Karsjens, L. Kim, A. Longshaw, T. Myers, A. Nakhimovsky, D. O’Connor, S. Tyagi, G. Van Damme, G. van Huizen, M. Wilcox, S. Zeiger, *Professional Java Server Programming, J2EE Edition*, Wrox Press Ltd., Birmingham, UK, 2000.

[2] J. Bansiya and C.G. Davis, “A Hierarchical Model for Object-Oriented Design Quality Assessment,” *IEEE Transactions on Software Engineering*, Vol. 28, No. 1, January 2002, pp. 4-17.

[3] BEA WebLogic web site, http://www.bea.com/

[4] M. Casagni, M. Lyell., L. Rosen, G. Vecellio, J. Warren, and H. Xiao “Architecture Comparison of J2EE Framework and FIPA-Compliant Software Agent Framework Using a Data Access Scenario," MTR 02W0000026, The MITRE Corporation, May 2002.

[5] S. DeLoach, M. Wood, and C. Sparkman, "Multi-agent Systems Engineering," *International Journal on Software Engineering and Knowledge Engineering*, Vol 11, No. 3, June 2001, pp. 231-258.

[6] T. Eiter and V. Mascardi, “Comparing Environments for Developing Software Agents,” INFSYS Research Report 1843-01-02, Institut Für Informationssysteme, Technical University, Vienna, March 2001.

[7] Enterprise Java Beans Specification 2.0 web site http://java.sun.com/products/ejb/docs.html

[8] FIPA specifications, See web site, http://www.fipa.org/

[9] FIPA-OS project hosted at the web site, http://sourceforge.net/projects/fipa-os/

[10] M. Griss and G. Pour, "Accelerating Development with Components", *Computer*, Vol. 34, No. 5, May 2001, pp. 37-43.

[11] S. Hayden,, C. Carrick, and Q. Yang. "Architectural Design Patterns for Multiagent Coordination." *In Proc. of the International Conference on Agent Systems* (Agents'99), Seattle, WA, May 1999.

[12] JADE project web site http://sharon.cselt.it/projects/jade/

[13] Jakarta Commons website http://jakarta.apache.org/commons/components.html

[14] “Java 2 Platform Enterprise Edition Specification v1.3”, Sun Microsystems, July 2001.

[15] A. Liu, L. Bass, and M. Klein, "Analyzing Enterprise Java Beans Systems using Quality Attribute Design Primitives," Technical Note, CMU/ SEI-2001-TN-025,Carnegie Mellon University, 2001.

[16] M. Lyell, L. Rosen, G. Vecellio, H. Xiao , “Interoperability through Messaging: An Investigation of Software Agent Frameworks as Compared with the J2EE Framework,", WN 01W0000022, The MITRE Corporation, May, 2001.

[17] M. Lyell , H. Xiao , L. Rosen, M. Casagni, G. Vecellio, J. Warren , “Performance Study of Software Agent Systems Using the FIPA OS V2.0 Agent Platform," MTR 02W0000032, The MITRE Corporation, May, 2002.

[18] Messenger, Jakarta Commons component, website http://jakarta.apache.org/commons/sandbox/messenger/

[19] Rumbaugh, J., M. Blaha, W. Premerlani, F. Eddy, W. Lorensen, *Object-Oriented Modeling and Design*, Prentice Hall, Englewood Cliffs, NJ, 1991.

[20] Stelting, S. and Maassen, O, *Applied Java Patterns*, Prentice Hall, Palo Alto, CA, 2002.

[21] Sun Java Center J2EE Patterns web site, http://developer.java.sun.com/developer/restricted/patterns/J2EEPatternsAtAGlance.html

[22] S. Woods and M. Barbacci,, "Architectural Evaluation of Collaborative Agent-Based Systems," Technical Note, CMU/ SEI-99-TN- 025, ESC-TR-99-025, Carnegie Mellon University, 1999.

[23] M. Wooldridge and N.R. Jennings, "Intelligent Agents: Theory and Practice," *Knowledge Engineering Review*, Vol. 10, No.2, 1995, pp. 115-152.

[24] M. Wooldridge, N.R. Jennings, and D. Kinny: "The Gaia Methodology for Agent-Oriented Analysis and Design," *Autonomous Agents and Multi-Agent Systems*, Vol. 3, No. 3, 2000, pp. 285-312.