SYSTEM ENGINEERING PROCESSES ACTIVITIES FOR AGENT SYSTEM DESIGN

Component based development for rapid prototyping

Jaesuk Ahn, Dung Lam, Thomas Graser, K. Suzanne Barber
Leboratory of Intelligent Processes and Systems, University of Texas at Austin, Texas, USA

Keywords: Component based development, agent system design, agent oriented software engineering, repository

Abstract: System designers of agent-based system are challenged by the lack of mature agent software development methodologies, the diversity of agent technologies, and the lack of a common framework for describing these technologies challenges architects attempting to evaluate, compare, select, and potentially reuse agent technology. Leveraging existing work to (1) categorize and compare agent technologies under a common ontology, (2) build a repository of agent technologies to assist system designer in browsing and comparing agent technologies, this paper proposes an architecting process and toolkit support to rapidly prototype an agent-based system by selecting agent technology components in the context of a given high level reference architecture and associated requirements.

1 INTRODUCTION

Agent technology is now being applied to the development of large open industrial software systems (Luck, et al., 2003). Before agent technologies can be used as generic building blocks, a methodology must be defined that guides agent-based system design using agent technology components. Furthermore, these methods and supporting tools must accommodate the construction of software systems that assemble highly flexible technology components written at different times by various developers (Griss and Pour, 2001). As a foundation for defining such methods and tools, Component Based Software Engineering (CBSE) offers an attractive approach for building enterprise software systems (Griss and Pour, 2001) and is currently a well-developed area of research within software engineering (Brown, 1996). CBSE works by developing and evolving software systems from selected reusable software components, then assembling them within an appropriate software architecture. Other approaches to component-based design of agent systems are often restricted to object-oriented implementation environments, usually based on Java (Martin, A et al., 1999), or do not have the ability to incorporate existing agent technologies into the design process (Brazier, Jonker et al., 2002).

In contrast, ongoing research in the Laboratory of Intelligent Processes and Systems at the University of Texas at Austin offers methods and tools for the component-based design of agent systems. Specially, this research concerns four key steps for component-based agent design:

- Step 1: Identifying a core set of agent functionalities known as agent competencies (planning/reacting, modeling, sensing, acting, organizing, coordinating, communicating) that adequately address the demands of the domain's operational requirements. In other words, the designer attempts to determine which competencies the agent must possess to perform the assigned functional requirements.
- Step 2: Constructing an Agent Reference Architecture that specifies technology-independent agent classes that encapsulate agent competencies.
- Step 3: Specifying agent technology (existing, envisioned, or under-development) as reusable components in the context of the agent competencies and providing a clear model to evaluate and compare technologies based on the agent competencies each is capable of delivering.
- Step 4: Constructing an Agent Application Architecture by browsing, selecting, and assembling agent technology components that

Ahn J., Lam D., Graser T. and Suzanne Barber K. (2005).
SYSTEM ENGINEERING PROCESSES ACTIVITIES FOR AGENT SYSTEM DESIGN - Component based development for rapid prototyping.
In *Proceedings of the Seventh International Conference on Enterprise Information Systems*, pages 196-202
DOI: 10.5220/0002527401960202

fulfill the given requirements captured in the architecture constructed in Step 2.

The first and second steps were already addressed in previous work (Barber and Lam, 2003). Barber and Lam defined a functional agent specification in terms of Core and Pluggable Competencies as a means to describe and compare agent interpretations and models. They also developed a method and tool, *Designer's Agent Creation and Analysis Toolkit (DACAT)*, for architecting an agent based on the defined Competencies such that the resulting architecture (Agent Reference Architecture) captures functional domain requirements. The third step was addressed by Barber and Ahn (Barber, Ahn et al., 2004). To address the third process step, the notion of an *Agent Competency Ontology* (Barber, Ahn et al., 2004) was proposed as a common ontology to represent agent technology at an abstract level of functional composition, and a repository, the *Technology Portfolio Manager (TPM)* (Barber, Ahn et al., 2004), demonstrated browsing and comparing agent technologies.

This paper centers on the fourth step, proposing the architecting of an agent-based system through the selection of agent technology components that fulfill functional (Competency) and data requirements captured in the Agent Reference Architecture (step 2). The Application architecture Creation and Evaluation Toolkit (ACET) is proposed for the fourth step. ACET leverages the technology repository in the TPM and the Competency-based Agent Reference Architecture to derive and evaluate agent-based architectures. An important premise of this approach is that every agent technology can be described by agent competencies. Consequently, the architect can build an Agent-based Application Architecture by selecting appropriate agent technologies according to their coverage of and compliance to both the functional (Competency) requirements and structure prescribed by the Competency-based agent Reference Architecture.

The basic definition of the Agent Competency Ontology is described in Section 2. An architecting process and supporting tool to select and assemble agent technology to create the agent system architecture is then presented in Section 3.

2 AGENT COMPETENCY ONTOLOGY

A multi-agent system (MAS) architect is guided by specific desired agent capabilities and system properties, in the context of a particular domain. Thus, agent technologies are developed / selected for a MAS by considering their application to a particular domain and their ability to offer desired capabilities (Competencies). Consequently, the architect must have a means for viewing and comparing agent technologies with respect to both competencies provided and domains supported. However, when attempting to compare various agent technologies or simply understand the breadth of agent technologies, the architect encounters obstacles that include the disparity in how agents are modeled and the lack of separation between domain-dependent functionalities (e.g., determine UAV route) and domain-independent functionalities (e.g., plan generation). As a result of this diversity, agent developers have difficulty comparing different views of agent technology or even different implementations of the same agent technology on some common basis.

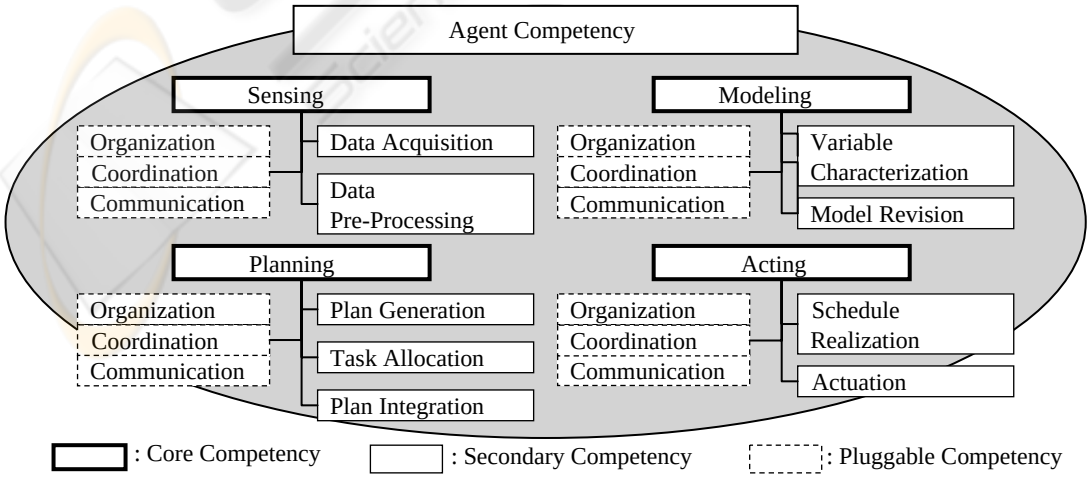

Figure 1: Agent Competency Ontology.

Barber and Lam (Barber and Lam, 2003) proposed Agent Competencies to model agents in a domain-independent manner. The Agent Competency Ontology was proposed as a common representational framework to specify agent technologies (Figure 2) (Barber, Ahn et al., 2004) . The Agent Competency Ontology is used to (1) map domain tasks such as "Generate UAV routing" to domain-independent competencies such as "planning" and, in general, (2) offer a common framework for representing and comparing agent technologies (Barber, Ahn et al., 2004). By specifying agent technologies in terms of these Agent Competencies, the agent technologies can be functionally compared and a common understanding among agent software engineers is promoted. Agent Competencies are based on the essential set of domain-independent functionalities an agent delivers. As seen in Figure 1, there are two types of Agent Competencies that form the framework for specifying agents.

Core Competencies (CCs) define the essential functionalities of an agent. Pluggable Competencies (PCs) are also defined because agents interact with other agents and entities in the system. PCs are not essential in single-agent systems, but are required to describe multi-agent systems (Barber and Lam, 2001). The Core Competencies includes:

Sensing: The agent needs to acquire appropriate data from other agents and the environment.

Modeling: Modeling is the maintenance of the information specified by the developer and/or derived from sensed data.

Planning: In the pursuit of goals, agents need the capability to choose the appropriate action(s) given its situation, and decide when and by whom those actions will be executed.

Acting: Schedules of actions are received and handled by the acting competency of the agent, which executes the appropriate actions at the appropriate times.

Pluggable Competencies: In addition to CCs, when an agent operates in a multi-agent system, it may have the functionality to communicate, to form organization(s), and to coordinate with other agents. Communication, organization, and coordination are Pluggable Competencies (PC) because they work in conjunction with and in the context of CCs.

3 AGENT APPLICATION ARCHITECTURE

The Agent Application Architecture (Agent AA) specifies a system design. Leveraging a well-defined, implementation-independent Agent Reference Architecture (Agent RA) that captures the functional, data, and timing requirements, the Agent AA is a collection of agent technology components selected according to their coverage of and compliance to the structure and requirements prescribed by the Agent RA (Barber and Bhattacharya, 2000). Using the Agent Competency Ontology described in Section 2 to specify agent technologies, a repository of agent technology specification maintained by the Technology Portfolio Manager (TPM) (Barber, Ahn et al., 2004) can be defined that facilitates exploration of potential agent technologies when building an Agent AA. In this section, an architecting process is described for deriving an Agent AA composed of agent technologies.

Section 3.1 describes knowledge acquisition process for this research and section 3.2 describes the Agent RA defined in DACAT and then demonstrates the use of ACET to specify an Agent AA.

3.1 Knowledge Acquisition Process

For this research effort, technologies to be included in this paper were developed as part of the Defence Advanced Research Project Agency - Taskable Agent Software Kit program (DARPA-TASK). The DARPA-TASK program was initiated with the specific intent to advance state-of-the-art agent technology as well as promote tools for easy agent-oriented design and analysis. Numerous universities and companies, developing a wide spectrum of technology, were involved in the program. The process of populating the TPM with DARPA-TASK technology specifications spanned multiple phases beginning with the collection of information available about a technology obtained from filtered presentations and papers posted by the technology providers involved in the DARPA-TASK program. Following initial modeling efforts, every Technology Provider was interviewed to verify the technology models and to obtain additional information which might have been missed from the gathered information. Agent technologies were described/modeled in terms of the domain-specific capabilities of the technology and the domain-independent agent competencies.

3.2 Specifying Agent Application Architecture

Agent RA is specified based on class-based encapsulations and Competency functionality. The Agent RA consists of (1) the classes that were

formed, (2) the functionality that each agent class encapsulates, and (3) the inputs, outputs, and interactions of the agent class as a result of the encapsulated functionality (Barber and Lam, 2003). The Agent RA is constructed in DACAT and output to an XML file.

The Agent Application Architecture process is demonstrated in the following sub-sections, where an Agent AA is derived in the context of an example domain, UAV target surveillance.

3.2.1 Evaluating the Technology Options for the Agent AA

Once an architect imports XML files of an Agent RA from DACAT and a technology repository from the TPM, ACET provides a graphical representation of the Agent RA and a technology repository listing allowing the architect to browse and compare agent technologies for inclusion in the Agent AA. The agent technologies from the TPM that can possibly satisfy the functional (Competency) requirements of each Agent RA class are displayed in a tree structure (upper left panel labeled "Registered Technologies" in Figure 2), and tree structure of the Agent RA is also displayed in lower left panel (labeled "Reference Architecture"). By selecting agent technologies from the tree, an architect can explore all the possible combinations of registered technologies that perform a desired task.

ACET responds by indicating the user-selected technologies in blue (in this case "Alphatech") and colors the related Agent RA classes and tasks based on coverage. If a user selects a technology in the *Registered Technologies* panel (Figure 2), Agent RA classes and Competency tasks (in the *Reference Architecture* pane in the lower part of Figure 2) not performed by the selected technology are colored in red, while Agent RA classes and Competency tasks colored in green and yellow are fully and partially supported by the selected technology, respectively.

3.2.2 Specifying the Agent AA

In this step, the architect selects appropriate technologies to satisfy the functionality (Competency) and data requirements specified in the Agent RA and aligning those technologies to Agent RA classes. The result is an Agent AA. In ACET, The Agent AA building space consists of two topologies: The Reference Architecture Topology and Technology Topology.

The *Reference Architecture Topology* (Figure 3) displays a comprehensive view of the class structures and associated technologies selected by the architect to deliver the Agent RA competencies encapsulated in that class. For each box in the Reference Architecture Topology view (Figure 3),

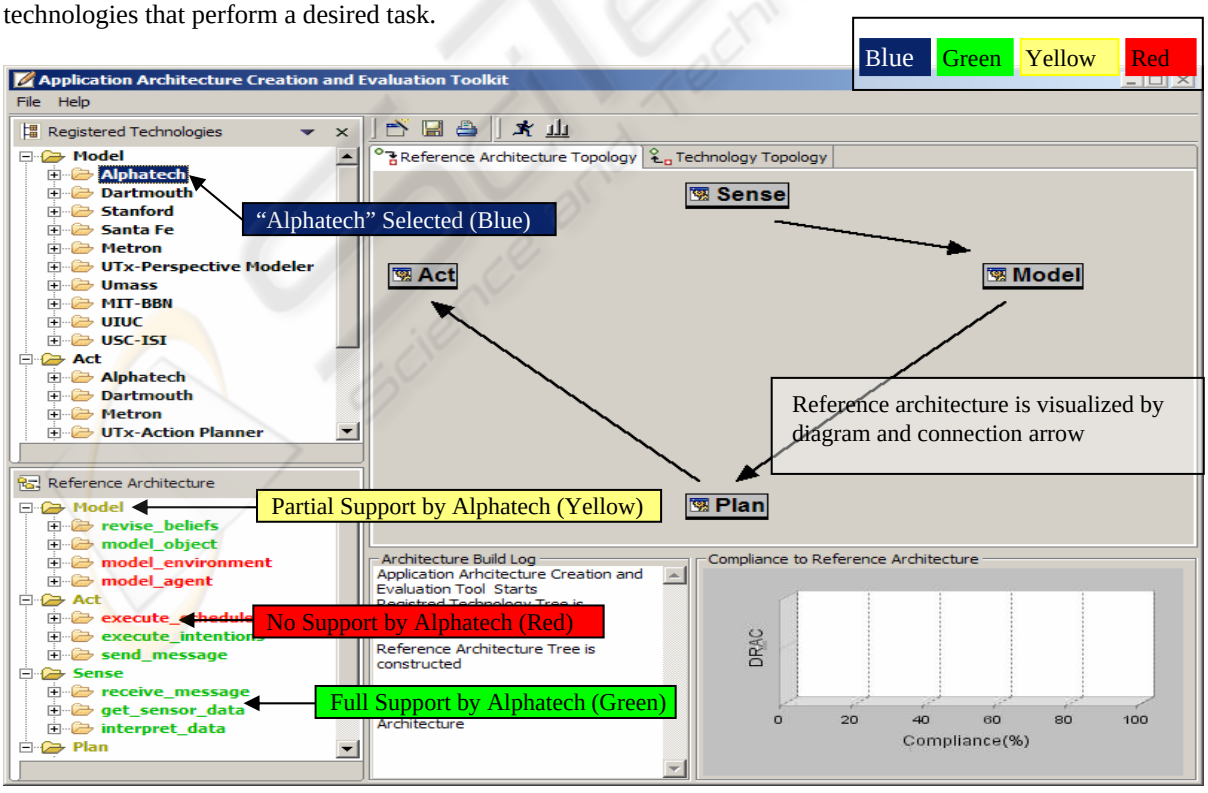

Figure 2: ACET: Building Space for the design.

the upper part of the box displays the name of the Agent RA class and the lower part displays selected technology for the respective Agent RA class. The respective boxes are also colored based on the degree to which the selected technology delivers/implements all the Competency functionality encapsulated in the respective Agent RA. The *Technology Topology* (Figure 4) displays the technology component structures and dependencies exhibited by selected technologies. Dependencies are due to I/O dependencies between functionality (Competencies) delivered by respective technology.

As an architect selects appropriate technologies to satisfy the functionality specified in the Agent RA and aligns those technologies to Agent RA classes, the result is an Agent AA specifying a system design.

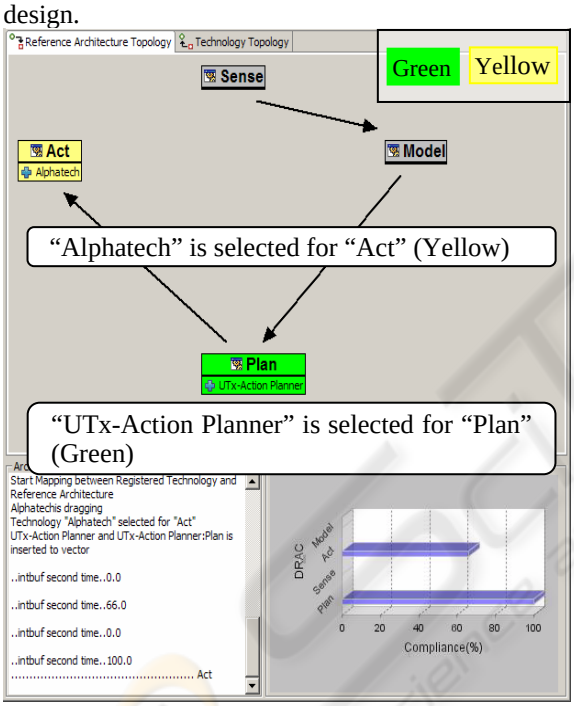

Figure 3: Reference Architecture Topology.

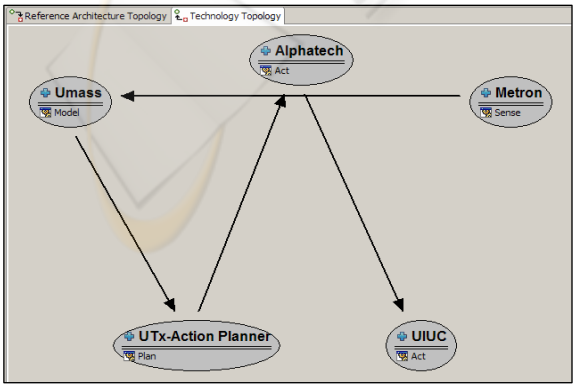

Figure 4: Technology Topology.

To build an Agent AA in ACET, an architect simply drags a technology from the registered technology list and drops it onto the desired agent class in the *Reference Architecture Topology* diagram, ACET then colors the diagram based on how much of the Competency functionality in the Agent RA class are satisfied by the selected technology. For example, "UTx-Action Planner" is selected for the "Plan" class in Figure 3. A green box indicates that the selected technology satisfies the entire set of Competency functionality and dependencies specified in the class (in this case "Plan"). The *Reference Architecture* panel also shows that "UTx-Action planner" performs all of Competency functionality of "Plan" class. To assist the architect in selecting technologies for each class, ACET also provides a compliance graph (lower right part in Figure 3). The compliance graph shows what percentage of the entire set of Competency functionality in an Agent RA class is delivered by the selected technology. In Figure 3, "Alphatech" is selected for the "Act" class. A yellow box indicates that the selected technology satisfies some of the Competency functionality and dependencies specified in the class (in this case "Act"). In this case, one of the three Competency functional tasks in the class "Act" is not satisfied by the selected technology "Alphatech". Therefore, the compliance graph indicates there is a 66% compliance value for the "Act" class.

3.2.3 Evaluating the Agent AA

Given a complete specification of the Agent AA, the evaluation process consists of measuring the coupling and cohesion of technology components.

The architect's objective is to select agent technologies which satisfy all of the Competency functional tasks and input/output requirements in Agent RA, as well as keeping the boundary of functional and input/output structure of the Agent RA.

The coupling for a technology component is defined as the total number of connections with other technology components. Thus, coupling measures the number of dependencies in which a technology component is involved. Since dependencies are directional, coupling is the sum of input and output coupling. Input coupling is the number of dependencies a technology component has on other technology components (i.e., incoming dependencies), and output coupling is the number of dependencies other technology components have on it (i.e., outgoing dependencies).

Cohesion, specifically functional cohesion, is the degree to which Competency functional tasks within an Agent RA class are covered by one or more

technologies; thus, cohesion focuses on the similarity of a technology's boundaries to an respective Agent RA class (i.e., the class task and inputs and outputs). An Agent AA with highly cohesive technology components indicates that selected technology components adhere to the class boundaries prescribed by Agent RA, thereby respecting the vision of the architect who derived the Agent RA structure.

Different combinations of technologies yield different coupling and cohesion values. The Agent AA derivation process involves exploring possible technology selections and observing resulting coupling and cohesion evaluations. Figure 5 illustrates ACET's evaluation space. The left column of Figure 5 shows coupling and cohesion metrics associated with the Agent RA, and the right column shows coupling and cohesion metrics calculated for the Agent AA.

For the illustrative example from the UAV target surveillance domain, both "Alphatech" and "UIUC" have been selected to provide functionality in the "Act" Agent RA class. As a result, the coupling value for "Act" in the Agent AA is greater than the coupling value for "Act" in the Agent RA. In addition, the cohesion of "Alphatech" with respect to the "Act" class is only 66%. The cohesion value of a class is helpful in measuring the similarity between the Agent RR and the Agent AA.

4 SUMMARY

When designing a software system architecture using available technology components (for envisioned, planned, under-development or existing technology), an architect evaluates various technology combinations with respect to the degree to which selected technologies meet stated requirements. For a Multi-Agent System architecture, technologies are evaluated with respect to (1) agent-related "competencies" provided (core capabilities that characterize agency including planning, acting, sensing, modeling, communication, organization and coordination), and (2) domain tasks supported (i.e., the problem domain being addressed by the agent system).

This paper illustrates the Application architecture Creation and Evaluation Toolkit (ACET) for deriving the Application Architecture. ACET supports the architect when performing the types of trade-off and what-if analyses associated which selecting appropriate agent technologies to deliver competencies specified in the Agent RA.

ACET's interface displays (1) a graphical and textual representation of the Agent Reference Architecture (Agent RA), (2) the coverage of respective Agent RA functionality by respective agent technologies from various technology providers, (3) the technologies selected for inclusive in the Agent AA, and (4) the dependencies between selected Agent AA technologies.

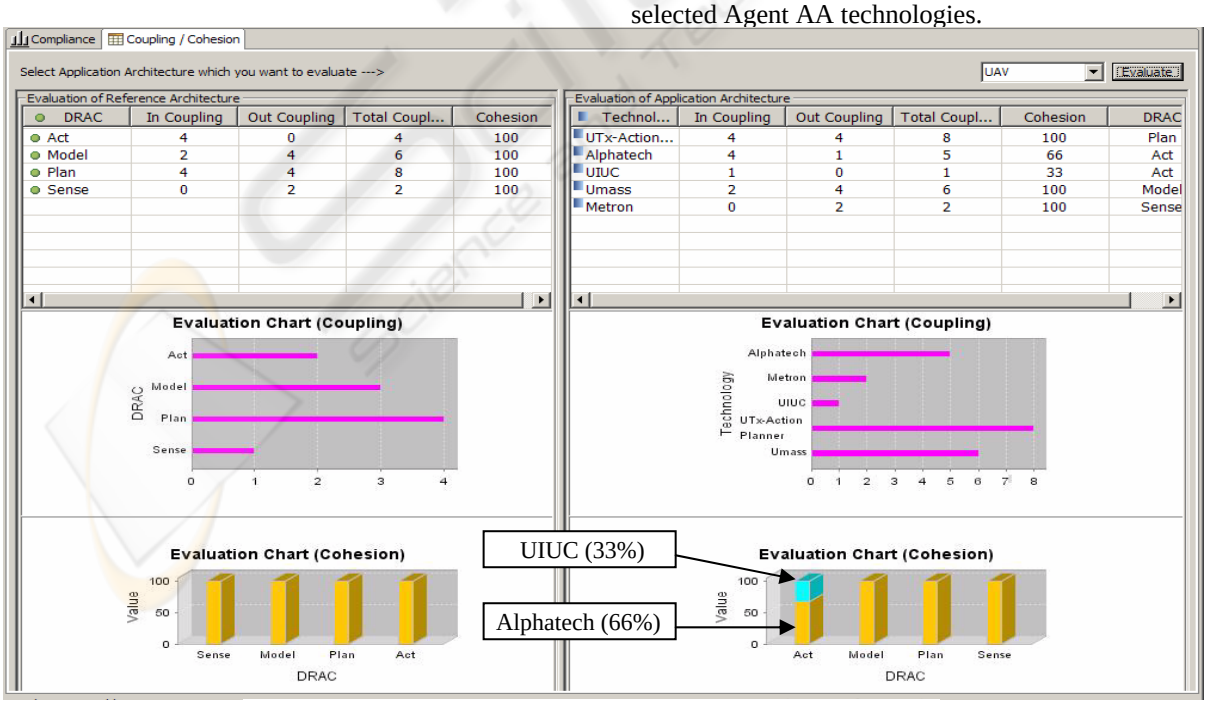

Figure 5: Evaluation of Agent Application Architecture.

The ACET's interface also allows the architect to assess how well selected technologies in the Agent AA comply to the Competency functionality and agent classes specified in the Agent RA. Specifically, ACET allows for the evaluation of the Agent AA with respect to compliance, coupling, and cohesion. Compliance measures the extent to which a set of selected Agent AA technology components satisfies the Agent RA specifications (functionality and data structure). Coupling measures the number of interactions and dependencies a given Agent AA technology component has on other technology components based on inputs required and outputs provided. Cohesion is calculated as the maximum percentage of Competency functional tasks in the Agent RA class covered by a single Agent AA technology among all technologies covering Competency tasks in the class.

The results of this paper help the architect to construct software systems that select and assemble highly flexible agent technology components written at different time by various developers. Specifically, the result enables the rapid prototyping of the complex agent-based systems by offering methods and tools to assist architects in comparing various agent technologies to construct and evaluate the Agent Application Architecture.

ACKNOWLEDGEMENT

This research is sponsored in part by the Defense Advanced Research Project Agency (DARPA) Taskable Agent Software Kit (TASK) program, F30602-00-2-0588. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Project Agency.

REFERENCES

Barber, K. S. and S. Bhattacharya, 2000. A Representational Framework for Technology Component Reuse. *13th International Conference on Software & Systems Engineering and their Applications (ICSSEA 2000), Paris, France*. 285-288.

Barber, K. S. and D. N. Lam, 2001. Architecting Agents using Core Competencies. *1st International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-2002) poster, Bologna, Italy*. 90-91.

Barber, K. S. and D. N. Lam, 2003. Specifying and Analyzing Agent Architectures using the Agent Competency Framework. *15th International Conference in Software Engineering and Knowledge Engineering, San Francisco Bay, USA*. 232-239.

Barber, K. S., J. Ahn, et al., 2004. Agent Technology Portfolio Manager. *16th International Conference on Software Engineering and Knowledge Engineering, Banff, Canada*. 37-44.

Brazier, F. M. T., C. M. Jonker, et al., 2002. *Principles of Component-Based Design of Intelligent Agents*. Data Knowledge Engineering 41(1): 1-27.

Brown, A. W. e., 1996. *Component-Based Software Engineering*, IEEE Computer Society Press.

Griss, M. L. and G. Pour, 2001. *Accelerating Development with Agent Components*. Computer 34(5): 37-43.

Luck, M., P. McBurney, et al., 2003. *Agent Technology: Enabling Next Generation Computing (A Roadmap for Agent Based Computing)*, AgentLink.

Martin, D., C. A, et al., 1999. *The Open Agent Architecture: a framework for building distributed software systems*. Applied Artificial Intelligent 13(1/2): 91-128.