Architectural Evaluation of Collaborative Agent-Based Systems

Steve G. Woods
Mario R. Barbacci

October 1999

TECHNICAL REPORT
CMU/SEI-99-TR-025
ESC-TR-99-025

Carnegie Mellon
Software Engineering Institute

Pittsburgh, PA 15213-3890

Architectural Evaluation of Collaborative Agent-Based Systems

CMU/SEI-99-TR-025
ESC-TR-99-025

Steve G. Woods
Mario R. Barbacci

October 1999

Architecture Tradeoff Analysis Initiative

This report was prepared for the

SEI Joint Program Office
HQ ESC/AXS
5 Eglin Street
Hanscom AFB, MA 01731-2116

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

FOR THE COMMANDER

Norton L. Compton, Lt Col., USAF
SEI Joint Program Office

This work is sponsored by the SEI FFRDC primary sponsor and the Defense Advanced Research Projects Agency. The Software Engineering Institute is a federally funded research and development center sponsored by the U.S. Department of Defense.

This work was created in the performance of Federal Government Contract Number F19628-95-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 52.227-7013.

For information about purchasing paper copies of SEI reports, please visit the publications portion of our Web site (http://www.sei.cmu.edu/publications/pubweb.html).

Table of Contents

List of Figures

List of Tables

Abstract

The Architecture Tradeoff Analysis Method[SM] (ATAM[SM]) is an architecture evaluation technique currently evolving at the Software Engineering Institute (SEI). ATAM has been applied to a number of command and control, real-time, and information systems. As collaborative, autonomous agents become a significant software technology, the demand for evaluating the quality attributes of the architectures of agent-based systems will increase. Very broadly, agents may be thought of as software entities that have the ability to undertake action autonomously in their particular embedded environment, according to a typically general set of requests or desired goals, and that are able to communicate with other agents as determined by their own initiative. Given an agent-system architecture, we need scenarios that could be applicable for conducting ATAM evaluations on instances of that agent architecture. This report identifies a few features in agent-based systems that could be used to classify agent-system architectures and to guide the generation of scenarios applicable to these architectures.

.

[SM] Architecture Tradeoff Analysis Method and ATAM are service marks of Carnegie Mellon University.

1 Introduction

In large software systems, the achievement of qualities such as performance, availability, security, and modifiability is dependent not only upon code-level practices (e.g., language choice, detailed design, algorithms, data structures, and testing), but also upon the overall software architecture. Quality attributes of large systems can be highly constrained by a system's software architecture. Thus, it is in our best interest to try to determine at the time a system's software architecture is specified whether the system will have the desired qualities.

A variety of qualitative and quantitative techniques are used for analyzing specific quality attributes [Barbacci 95]. These techniques have evolved in separate communities, each with its own vernacular and point of view, and have typically been performed in isolation. However, the attribute-specific analyses are *interdependent*; for example, performance affects modifiability, availability affects safety, security affects performance, and everything affects cost. In other words, achieving a quality attribute can have side effects on other attributes. These side effects represent dependencies between attributes and are defined by parameters that are shared among attribute models. If we can identify these side effects, the results from one analysis can feed into the others.

Quality-attribute goals, by themselves, are not definitive enough either for design or for evaluation. They must be made more concrete. Using modifiability as an example, if a system can be easily adapted to have different user interfaces but is dependent upon a particular operating system, is it modifiable? The answer is that it depends on what modifications are expected to the system over its lifetime. That is, the abstract quality goal of modifiability must be made concrete. The same observation is true for other attributes.

Agent-based systems are emerging from the artificial intelligence (AI) research community and are being used for increasingly more sensitive applications in which economic, privacy, safety, and other concerns are of paramount importance. Even though achieving the desired functionality was challenging in the past, developers now have to contend with additional nonfunctional (i.e., quality) attributes. The new situation calls for a more deliberate look into the architecture of agent systems.

The Architecture Tradeoff Analysis Method^{SM} (ATAM^{SM}) is an architecture evaluation technique currently evolving at the Software Engineering Institute (SEI).[1] The input to ATAM

1. Architecture Tradeoff Analysis Method and ATAM are service marks of Carnegie Mellon University.

consists of a system architecture and the perspectives of stakeholders involved with that system; the output is an understanding of the architectural decisions that are used to achieve particular quality goals and the implications of those decisions.

This report is intended to serve as the background for the application of the ATAM process to agent-based systems. The report will focus on a the coordination model of selected agent systems and a collection of quality attributes as previously discussed. It is important to point out that the ATAM does not provide an absolute measure of “architecture quality”; rather it serves to identify scenarios from the point of view of a diverse group of stakeholders (e.g., the architect, developers, users, sponsors) and to identify risks (e.g., inadequate performance, successful denial-of-service attack) and possible mitigation strategies. The results of an ATAM evaluation thus reflect the concerns of the particular collection of stakeholders and the scenarios that they consider to be important at the time of the evaluation. This report is concerned with the identification of typical quality attributes, scenarios, and agent-system architectural patterns as a way to expedite evaluations of real agent systems.

2 The Architecture Tradeoff Analysis Method (ATAM)

The Architecture Tradeoff Analysis Method is an architecture evaluation technique currently evolving at the SEI. The input to the ATAM consists of a system architecture and the perspectives of stakeholders involved with that system. The ATAM relies on generating scenarios to assess the architecture, where the scenarios are example stimuli used to represent stakeholders' interests and to understand quality attribute requirements.The scenarios give specific instances of anticipated use, performance requirements or growth, various types of failures, various possible threats, various likely modifications, etc.

After a collection of scenarios is generated by the stakeholders, the evaluation proceeds by applying the scenarios to the architecture and developing an understanding of the architectural mechanisms that are used to achieve particular quality goals and the implications of those mechanisms. Each quality attribute is examined from the point of view of three perspectives: What external stimuli cause the architecture to respond or change; what mechanisms are used within the architecture to control the response; and how is the response to these stimuli measured? For performance, for example, the external stimuli are events arriving at the system; the mechanisms are scheduling, concurrency, and resource management; and the measurements are latency and throughput. For modifiability, for example, the external stimuli are changes to the system; the mechanisms for controlling the cost of changes are encapsulation and data indirection; and the measurement is the cost of a collection of changes.

There are three different types of outputs from an evaluation:

1. a collection of "sensitivity" or "tradeoff" points. A *sensitivity point* is a property of the architecture that is critical for the achievement of a particular quality attribute (e.g., using encryption to achieve confidentiality). A *tradeoff point* is a sensitivity point that is sensitive for multiple quality attributes (e.g., encryption improves security but increases latency).
2. a framework for reasoning about the system. The framework may include a variety of qualitative and quantitative techniques, ranging from a discussion that follows from the exploration of a scenario, to a model or a portion of a model and a discussion of how that model might be analyzed when instantiated, all the way to a formula that represents how to calculate the value of a particular quality attribute.

3. a list of issues not addressed or decisions not yet made. The list of issues not addressed or decisions not yet made arises from the stage of the life cycle of the system at the time of the evaluation. An architecture represents a collection of decisions. Some of these decisions are known to the development team as having not been made and are on a list for further consideration. Others are news to the development team and stakeholders, and the evaluation helps to identify and document them.

The ATAM includes the following activities:

- **Description of the architecture views/styles**: The architecture is presented in as much detail as is currently documented. During this step, the architect also identifies quality attribute-specific architectural approaches and styles.
- **Gathering and mapping of scenarios**: Scenarios are elicited from and prioritized by the stakeholders. The architect then maps the high-priority scenarios onto the architectural description.
- **Identification of risks/sensitivities/tradeoffs**: As a result of the scenario mapping and the ensuing analysis, sets of risks, sensitivity points, and tradeoffs are documented.

We focus in particular on the first two items: agent-based system styles or approaches used in different subsystems, and the generation of scenarios applicable to each style.

Before we deal with specific styles and scenarios, we need to understand the characteristics of agent-based systems that might set them apart from more traditional real-time or embedded applications used in Command and Control (C^2) applications. That is, what quality attributes do we care about in agent-based systems?

2.1 Quality Attributes of Agent-Based Systems

> *Intelligent Software Agents are defined as being a software program that can perform specific tasks for a user and possesses a degree of intelligence that permits it to perform part of its task autonomously and to interact with its environment in a useful manner* [Brenner 98].

A number of survey books and articles on the subject of agent-based systems [Bradshaw 97, Brenner 98, IAG 97, Hayden 99, Nwana 96] outline a set of typical design patterns for collaboration among agents, but they do not delve into the critical aspects of evaluating agent-based systems in terms of software quality attributes.

We see the following set of attributes as particularly relevant to agent architectural evaluation. While this list is by no means complete, it represents a starting point for developing an understanding of the qualities of interest to the deployers of an agent-based system for a particular use.

- **Performance predictability**
  Since agents have (by definition) an usually high degree of autonomy in the way that they undertake action and communication in their domains, it is difficult to predict from a higher level of abstraction individual agent characteristics (such as timeliness and latency) as part of determining the behavior of a system at large.

- **Security against data corruption and spoofing**
  Agents are often free to identify their own data sources (Web agents for example), and they may undertake additional actions based on these sources. In addition to possibly misleading information (deliberate or as a matter of course) acquired by agents, there is the distinct possibility of hostile agents attempting to "spoof" system agents as part of an effort to acquire information normally accorded to only trusted agents of a particular lineage. The protocols of verifying authenticity for data sources by individual agents is therefore an important concern in the evaluation of overall system quality.

- **Adaptability to changes in the environment**
  Agents may be required to adapt to modifications in their environment. Modifications may include changes to the agent's communication language (new versions of KQML for example, roughly analogous to message format changes in traditional distributed architectures), or possibly the introduction of a new "type" of external agent previously unknown that may (or may not) possess capabilities of use to a particular agent system.

- **Availability and fault tolerance**
  Agents that offer services to other agents implicitly or explicitly make promises about the availability of the offered services (e.g., fraction of time the agent or service is working). A related attribute is the ability of the agent system to detect, contain, and deal appropriately with external agents such that erroneous inputs (malicious or otherwise) are not propagated destructively into the agent system itself.

The same quality attributes are of interest to other types of systems, albeit with perhaps different relative emphases depending on the problem domain and architecture style chosen. However, while there is some degree of agreement on the definitions of traditional architectural styles (e.g., pipes-and-filters, client-server), there is less agreement when we attempt to identify agent-system architectures.

2.2 Architectures of Agent-Based Systems

Software agents have evolved from the multi-agent systems (MAS) branch of the distributed AI taxonomy [Bond 88] as shown below:

- distributed artificial intelligence
  - parallel artificial intelligence (PAI)
  - distributed problem solving (DPS)
  - multi-agent systems

While parallel AI focuses on performance and resource utilization, distributed problem solving and multi-agent systems focus on the cooperation and coordination of agents to solve a problem, albeit using different approaches. DPS follows a top-down approach consisting of three steps: problem decomposition and assignment, independent solutions of subproblems, and combinations of the subproblem solutions. MAS, on the other hand, follows a bottom-up approach and relies on communication and interaction between agents to negotiate tasks and resolve conflicts.

The range of agent types under consideration in the literature is expanding rapidly, and there is no consensus on a definition of what constitutes an agent. It is an umbrella term, covering a range of specific agents. For example, H. Nwana in "Software Agents: An Overview" notes, "some cynics may argue that this strand arises because everybody is now calling everything an agent, therefore resulting inevitably in such broadness [Nwana 96, p. 2]." Further, he offers a typology of agents based on many physical guises and many roles as the axes in a space of agents:

- **mobility**: static, mobile
- **behavior**: deliberative (have an internal model), reactive (have a stimulus/response behavior)
- **desired attributes**: autonomy, learning, cooperation

If we consider additional attributes [such as versatility, benevolent/non-helpful, antagonistic/altruistic, attitude (emotion, belief/desire/intention)], the space expands considerably, although the multiple dimensions can be collapsed into a list of agent types [Nwana 96]:

- **collaborative:** exhibiting learning and cooperation or autonomy and cooperation behavior (The result is greater that the sum of the parts.)
- **interface:** exhibiting autonomy and learning attributes (acting on behalf of their owners)
- **mobile:** roaming on behalf of owners
- **information/Internet**: managing information from distributed sources
- **reactive:** exhibiting emerging behavior (stimulus/response)
- **hybrid**: a mix of previous agent types
- **smart:** exhibiting autonomy, learning, and cooperation
- **heterogeneous:** similar to hybrid but integrated, including legacy systems

A different classification [IAG 97] identifies a different set of defining properties:

- acting on behalf of other entities in an autonomous fashion
- performing actions with some degree of proactivity and/or reactivity
- exhibiting some level of key attributes (learning, cooperation, mobility)

Each of these attributes leads to the definition of three main branches:

- **interactive agents:** Act on behalf of a user, try to acquire knowledge about user behavior, and anticipate/optimize display of information. Interactive agent architectures include information filtering agents, information retrieval agents, and personal digital assistants.
- **distributed agents:** The main attribute is cooperation; agents communicate, coordinate, and negotiate among themselves. Agents use alternative coordination models: organizational (central), by contract/bidding, and by planning ahead (central, distributed). Agents use alternative negotiation models: competitive (independent agents with independent goals) and cooperative (all agents have a single global goal).
- **mobile agents:** Agents combine multiple models: life-cycle model, computational model, security model, communication model, and navigation model. Most models are implemented by the environment, which also provides services of various types.

These taxonomies focus on functional properties of the agents. As functionality increases, proliferation of agent "types" mushroom; confusion is inevitable because of unavoidable overlaps and subtle distinctions. These taxonomies might serve marketing needs but do not help developers or users in evaluating quality attributes such as security, availability, or performance. Additional taxonomies are cited in the introductory chapter of [Bradshaw 97], but we do not need to belabor the point.

Instead of attempting to conduct attribute tradeoffs based on ever-changing taxonomies of styles, we take a different approach: First, we do not look at the functionality of the agents in the system, rather we concentrate on external properties such as communication, cooperation, and coordination actions performed by the agent-system components and connections. This defines our "architecture" domain. Second, we look at the instantaneous architecture (i.e., components and connections and the patterns of behavior among the components). Third, we identify a "central" component and conduct the analysis from the point of view of this component as the focus of the interaction between the other components. As we shall see in the next chapter, this simple approach seems to describe a large variety of agent systems.

Specifically, we look at the number of inputs/outputs to/from the focus agent, the message processing actions (bundling, unbundling, translation) of the focus agent, and other actions specific to the focus agent. S. Hayden makes a good first step in this direction [Hayden 99], but unfortunately that report still focuses on a "catalog" of coordination models with the inevitable overlaps and subtle distinctions that we have observed in other taxonomies.

3 The Space of Architectural Choices

To differentiate the various types of agent systems we define a space of seven "binary" choices, arranged as columns in a "table of styles," augmented with an additional column describing the main purpose of the focus agent.[1] Using this simple scheme, the problem with the usual names of agent systems used in the literature become readily apparent. In some cases, the names are too broad and encompass a whole range of behaviors that are worthy of separate consideration, as "subtypes." In other cases, the behavior of different agent systems are more closely related than the difference in their names might suggest.

The seven binary choices are described below:

- **communication choices** (four columns): one-one, one-many, many-one, many-many. The normal case is identified by "Y," a disallowed case is identified by "N." A blank cell indicates a "don't care" situation, where either choice (Y/N) is applicable to the situation. We follow the convention of associating the left side with the initiator of the operation and the right side with the responder. The focus agent is in the middle. For example, "1-N" suggest one (1) initiator and multiple (N) responders.
- **processing choices** (three columns): bundling of input messages, unbundling of input messages, translation of input messages.

3.1 Wrapper Family

Table 3-1 illustrates the choices involved in a wrapper, an agent type useful in evolving systems. A wrapper agent allows a legacy application to be incorporated into a multi-agent system. The legacy code is extended with agent capabilities by agentifying it [Hayden 99].

The wrapper allows agents to communicate with the legacy system using an agent communication language (ACL) by acting as a translator between the agents and the legacy system. This ensures that agent communication protocols are respected and the legacy system remains decoupled from the agents.

1. We do not suggest that all 2^7 =128 rows make sense or deserve to be recognized as a different type of agent style. Rather, we expect that groups of rows in the table will collapse into coordination models. We further suggest that a number of existing coordination models have a hierarchical relationship based on size/nesting of regions.

References		Agents-Legacy				Internal Behavior of Focus Agent			
		1-1	1-M	M-1	M-M	bundling	unbundling	translating	Action
	wrapper [Hayden 99]		N	Y	N	Y	Y	Y	Provides an agent-like interface to a legacy system

Table 3-1: Space of Architectural Choices in a Wrapper Agent

The wrapper is usually domain or system specific and wraps around a single legacy system, although there might be any number of agents on the other side of the wrapper. This is indicated in the cells under "Number of external participants," where the normal case [i.e., many agents, one legacy system (M-1)] is marked Y and the degenerate case of one single agent (1-1) is left blank, indicating a don't care situation.

The advantages of using a wrapper are that the life cycle of the legacy system is extended while it is redeveloped and that the replacement will not require any changes to the interface implemented by the wrapper; the disadvantages are that the legacy system might lack newer functionality and that the wrapping adds performance overhead.

The legacy system should be capable of supporting any of the requests, commands, or activities possible in the agent communication language or provide a graceful denial. The wrapper can examine the traffic and filter the messages if necessary—inappropriate requests can be turned down by the wrapper. The wrapper illustrated in Table 3-1 has a rather rich set of capabilities: (1) it can bundle multiple agent messages into one message to the legacy system (e.g., the wrapper understands the purpose of the individual requests and "packages" them to match the functionality of the legacy); (2) it can unbundle agent messages into multiple messages to the legacy system (e.g., the agent request might be too complicated for the legacy system to handle and the wrapper breaks it into pieces matching the real functionality of the legacy); and (3) typically there is message translation between the ACL formats and the format of the commands understood by the legacy system.

Not surprisingly, one could imagine wrappers with more modest capabilities. But aside from cost and development time, why would anyone want a simpler wrapper? The answer is directly related to the quality attributes of the architecture—a wrapper that does not bundle agent messages might generate extra requests of the legacy system, but the latency of the individual responses to the agent might be shorter; a wrapper that does not unbundle agent messages would not leave the legacy system in some inconsistent state should the wrapper crash in the middle of the sequence; a wrapper with limited translation capabilities might have to reject complicated agent messages but will run faster for the acceptable messages.

As we shall see, as the role of the wrapper is extended, it acquires characteristics of other patterns such as monitor (watching and filtering the traffic) and broker (forwarding requests from the legacy system to an appropriate agent).

The wrapper "family" is described by Table 3-2. The importance of breaking down a family is not that one might need such fine agent specializations, but rather because specific quality-attribute concerns that might be masked if one looks at the whole family might become apparent in one of the individual members of the family.[1]

References		Agents-Legacy				Internal Behavior of Focus Agent			
		1-1	1-M	M-1	M-M	bundling	unbundling	translating	Action
Wrapper family	wrapper --	Y	N	N	N	N	N	N	Primitive wrapper, handles only one agent and has thin understanding of the legacy application or the agent.
		Y	N	N	N	N	N	Y	Additional variations on the single-agent wrapper. Different quality attributes and constraints in behavior might be implied in each of the alternatives.
		Y	N	N	N	N	Y	N	
		Y	N	N	N	N	Y	Y	
		Y	N	N	N	Y	N	N	
		Y	N	N	N	Y	N	Y	
		Y	N	N	N	Y	Y	N	
		Y	N	N	N	Y	Y	Y	
			N	Y	N	N	N	N	Variations on a multi-agent wrapper. Different quality attributes and constraints in behavior might be implied in each of the alternatives.
			N	Y	N	N	N	Y	
			N	Y	N	N	Y	N	
			N	Y	N	N	Y	Y	
			N	Y	N	Y	N	N	
			N	Y	N	Y	N	Y	
			N	Y	N	Y	Y	N	
	wrapper ++		N	Y	N	Y	Y	Y	Full functionality for a multi-agent wrapper.

Table 3-2: Wrapper Family

1. We can only speculate at present, but after we gain more experience with agent-system architecture scenarios and evaluations, we might begin to recognize patterns of "cells" that raise specific concerns (e.g., availability, denial of service) and call for appropriate scenarios.

3.2 Monitor Family

The monitor family is illustrated in Table 3-3. The intent of the monitor is to detect selected state changes in an agent of interest (the subject) and notify other agents (the subscribers) of their occurrence.

Reference		Subject-Subscribers				Internal Behavior of Focus Agent			
		1-1	1-M	M-1	M-M	bundling	unbundling	translating	Action
	monitor		Y	N	N	N	Y	N	Tracks agent state changes and notifies subscribers

Table 3-3: Monitor Family

The monitoring pattern is a compromise between the consistency that can be achieved in a tightly coupled system with the decoupling and reusability that can be achieved in a distributed system. Agents that must remain current with respect to a subject agent subscribe (via the monitor) to notification of state changes. The monitor in turn requests notification of state changes from the subject agent. When the state of the subject agent changes, the subject notifies the monitor and the monitor in turn notifies the subscribers (thus, the unbundling is just a replication of the agent notification).

The monitor pattern is suitable if an indefinite number of subscribers is interested in the subject or if it is desirable to decouple the subject from the subscribers. Subscribers may be interested in subsets of the subject agent state; the subject agent only needs to notify the monitor of state changes for which there is at least one subscriber.

The advantages are that the subjects are decoupled from the subscribers and there is less work for the subject because it needs to notify only the monitor, not a potentially unbounded number of subscribers. This pattern does not protect from unforeseen consequences (such as system instability) from large numbers of updates or cascading updates when subscribers are themselves the subjects of other subscribers through different monitors. While a single monitor could detect a number of dependencies and cascaded dependencies in the overall application, this choice would have other effects on performance (the single monitor does all the work) and availability (the single monitor is a single point of failure and a rich target for an attacker).

3.3 Matchmaker/Broker/Mediator Family

These three agent patterns are clearly related, as shown in Table 3-4, although they appear under different names in the literature. These agents act as intermediaries between a number of agents providing services (the providers) and a number of agents requesting services (the consumers).

Reference		Consumer-Providers				Internal Behavior of Focus Agent			
		1-1	1-M	M-1	M-M	bundling	unbundling	translating	Action
	Matchmaker [Brenner 98]		Y	N	N	N	N	N	Exports agent maps/ids
	Broker [Hayden 99]		Y	N	N	N	N	Y	Keeps agent maps/ids
	Mediator [Hayden 99]		Y	N	N	Y	Y	Y	Keeps agent maps/ids and active participation

Table 3-4: Matchmaker/Broker/Mediator Family

The matchmaker simply locates a provider corresponding to a consumer request for service, and then hands the consumer a handle to the chosen provider directly. Thus, the negotiation for service and actual service provision are separated into two distinct phases. The broker, on the other hand, directly handles all interactions between the consumer and the provider. Their behaviors are illustrated in Figure 3-1 and Figure 3-2 [Brenner 98, Figure 4.3].

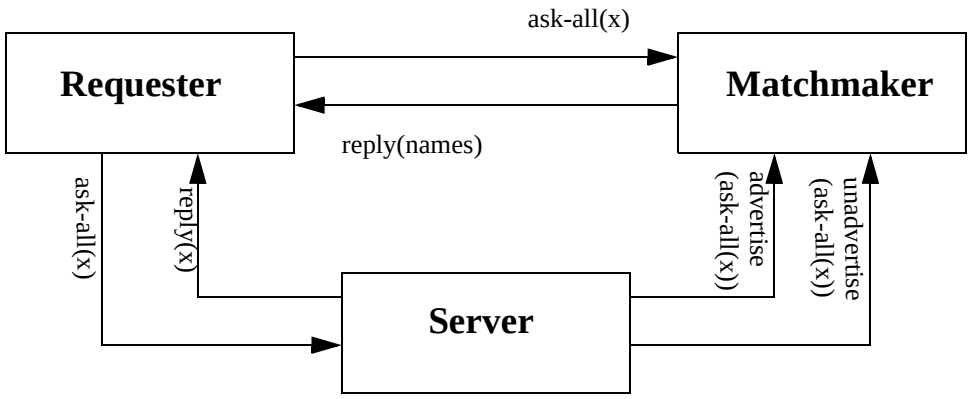

Figure 3-1: Operation of a Matchmaker

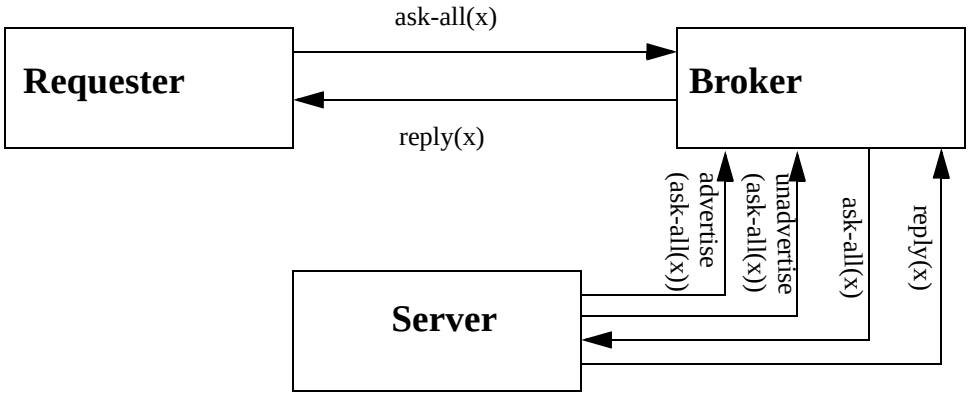

Figure 3-2: Operation of a Broker

The mediator acts like the broker with the following difference: While a broker simply matches providers with consumers; a mediator encapsulates agent interactions and maintains models of behaviors over time; these are the bundling/unbundling actions. [1]

The broker presents a tradeoff with the matchmaker in that the broker insulates the consumer and provider. Thus the broker protects the provider from hostile or unruly consumers at the expense of a doubly-tasked broker component whose failure is crucial to any brokered interactions that are ongoing or in negotiation.

The matchmaker limits its failure impact to the negotiation phase since previously negotiated interactions are undertaken between the consumer and the provider directly; however, this flexibility is provided at the expense of revealing provider identities to consumers. The matchmaker interaction in its simplest form then does not guarantee secure and mediated interactions; however additional negotiation steps can be imagined (key exchanges, for instance) which would further secure these interactions at the expense of additional communication and computation overhead.

Some authors [Hayden 99] extend the standard broker architecture by adding explicit "registration" exchanges between consumers and the broker as well as between providers and the broker. This extension explicitly addresses security concerns indicated in some of the scenarios.[2]

---

1. In "Architectural Design Patterns for Multi-Agent Coordination," S. Hayden states that legacy systems that make requests on their own can not use a "normal" wrapper and must use a mediator agent instead [Hayden 99, p. 7]. These two patterns become confused because proper use of a legacy system might impose limits on interaction patterns. (That is, a wrapper might be "mediator-like" even if the legacy system does not make requests, just to prevent improper use of the legacy system.)

2. Note however that "register" is apparently not a current core performative in the Knowledge Query and Manipulation Language (KQML).

The advantage of a mediator is that it further decouples the agents: Individual agents do not have to maintain models of behavior for all other agents around, and the mediator allows each agent to vary its behavior independently. Agents need to be aware of only the mediator.

The mediator pattern works if the interactions between agents are well defined, even if coordination of distributed behavior is desired (i.e., the mediator must know the valid sequences). Cooperating agents send and receive requests to/from the mediator, which routes the requests in accordance with a specific conversation model. The disadvantage of this pattern is that the mediator becomes a performance bottleneck or a single point of failure. This can be remedied as in the broker pattern, by having multiple mediators that coordinate their activities.

3.4 Contract-Net Family

The Contract-Net family is illustrated in Table 3-5. In Contract-Net [Smith 80], a manager agent (the focus) issues a request for proposal (RFP) or bids for a particular service to all participating agents. The manager then accepts "proposals" to meet the service request at a particular "cost" (or messages indicating the contractors unwillingness to bid). The manager selects among these proposals and indicates acceptance to exactly one bidder. Optionally, the manager indicates to non-successful bidders that their bids have been rejected. The selected contractor performs the contracted work and informs the manager upon completion.

Reference		Client-Contractors				Internal Behavior of Focus Agent			
		1-1	1-M	M-1	M-M	bundling	unbundling	translating	Action
	Contract-Net		Y	N	N	Y	N	Y	Negotiates with contract bidders

Table 3-5: Contract-Net Family

The Contract-Net is particularly interesting because specifications for this protocol "hide" the existence of a client. The interactions as shown in Figure 3-3 are defined entirely in the context of a manager and possible multiple bidders. Implicit in the protocol is the existence of a client requesting the manager to locate a provider of a particular service.

The Contract-Net acts in a manner similar to some patterns examined earlier. It is similar to a broker in that the client and the contractor are not aware of the identity of the other. It is also similar to a mediator in that it is heavily involved in the negotiation with the contractors, and not just in selecting any one at random. In addition, a successful negotiation of the manager and a contractor might involve a direct contractor-to-client interaction that is neither specified

nor prohibited (i.e., the manager is also a matchmaker). The “bundling” and “translating” actions in the Contract-Net represent the packaging of the client’s needs into an RFP.

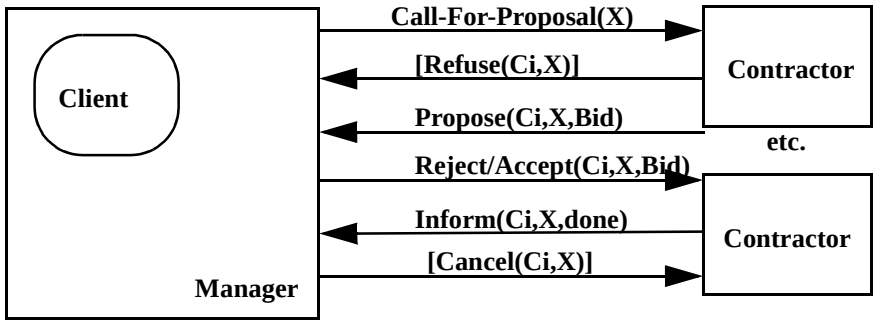

Figure 3-3: Operation of a Simplified Contract-Net

3.5 Embassy Family

In [Hayden 99], a mechanism is described for supporting a foreign agent’s access to the services of an agent system called “Embassy,” which is illustrated in Table 3-6. Embassy interaction is composed of a trusted “local” component (the focus) as the single point of interaction between one foreign agent and multiple local agents.

Reference		Foreign Agent - Local Agents				Internal Behavior of Focus Agent			
		1-1	1-M	M-1	M-M	bundling	unbundling	translating	Action
	Embassy		Y	N	N	Y	Y	Y	Controls access of foreign agent to local resources

Table 3-6: Embassy Family

The embassy single-point-of-entry method is common to networks (gateways), and it limits the scope of violations significantly and the cost of single point-of-failure impacts to local/foreign interactions. Primary interactions for the embassy are described by [Hayden 99] as a generalization of the broker. The embassy is responsible for any semantic translation from foreign performatives into local performatives and vice versa.

Clearly it may not be the case that all performatives are supported. In this aspect, an embassy in general may also act as a mediator. The main phases include a client requesting access, an embassy granting access, the client issuing a performative to embassy, the embassy "passing through" the performative to a local destination after necessary translations, the embassy receiving local responses to the performative, and the embassy translating the local response into the foreign response format and responding to the foreign agent.

4 Example: Scenarios for Matchmaker, Broker, and Mediator Agent Systems

The following outlined scenarios are meant to indicate the possible range of issues applicable to the agent types described in Chapter 3. Although we classify the scenarios by agent types (i.e., matchmaker, broker, mediator), it should be apparent that many scenarios are generic and can apply to multiple agent systems.

The matchmaker differs from the broker primarily in that the matchmaker actually passes responsibility for service provision directly to the matched server and requesting client, whereas the broker completely hides the identity of the server from the client and vice versa.

Agent systems are susceptible to attacks in which an outside party intervenes in an inter-agent exchange. Attacks that require explicit knowledge of the server by the client or knowledge of the client by the server are prevented in the broker agent. On the other hand, the existence of a specific intervenor in the communication between the client and the server might have an impact on performance (increase in latency), availability (single point of failure), and even security of information since there are now two separate communication channels to be protected. While this is unavoidable in some cases, for agent-to-agent communication it is quite possible that public key encryption can help in the positive identification of an agent prior to acceptance of a particular message. Such choices are the nature of architectural analysis. At any rate, the scenarios listed below are indicative of the considerations used in evaluating architectural properties of these negotiation patterns.

We describe the example scenarios using the following pattern:

N. <Scenario Description>

May Affect

Quality attributes that might be affected by the scenario

Implications

A description of the risk or potential problem illuminated by the scenario

Possible Solutions

A description of possible solutions, if any, to the risk

4.1 Attacker (Consumer) Issues Massive Requests for Services

May Affect

- performance
- availability

Implications

- denial of service
- A single broker for all services implies that one successful attack on the broker destroys all service allocations.

Possible Solutions

- A multiple-broker solution would address this issue; however it requires that consumers have a more elaborate understanding of the available brokers and their specialities, if any. This is a clear tradeoff between robustness/availability and complexity and overhead.
- A registration of the consumer with the broker and provider could allow the broker to limit the frequency of consumer requests. For certain services and/or agent systems, this may be appropriate and could alleviate this kind of attack risk.
- Provider is restricted to a single request type to the same consumer in a given time frame.
- All consumer/provider interactions are monitored to limit the consumer actions to expected values, including volume of messages.
- The matchmaker issues only single provider addresses to the consumer in a given time frame.
- The matchmaker notifies the recommended provider of requests by the consumer in order to make service available for a limited time only.

4.2 Provider Fails After Advertising Service

May Affect

- availability of service
- performance of consumer
- reliability of consumer

Implications

- If the matchmaker database contains information about non-available services, a consumer can obtain a handle to an unavailable service. If the consumer blocks while waiting for the service, the system might hang.

Possible Solutions

- The matchmaker regularly pings service providers as assurance of presence.
- The matchmaker requires regular re-advertisement of service.
- The consumer times out and notifies the matchmaker.

4.3 Spoofer (Customer) Acquires Service and then Seeks to Replace It

May Affect

- security of service if the attacker spoofs the provider
- reliability of service
- performance of provider

Implications

- The matchmaker database may contain erroneous provider/service information, and a customer could obtain an apparently valid provider and receive an erroneous result.
  - If the spoofed provider delivers an apparently valid response on contract, the consumer could expect contracted service and not receive it.
  - If the spoofed provider fails to reply to a contract request, the consumer hangs.

Possible Solutions

- Do not provide explicit provider process or port information to the consumer.
- Use public key encryption—for instance:
  - The provider generates private key SPriv and registers public key SPub with the matchmaker.
  - The consumer receives the provider identifier and SPub from the matchmaker.
  - The consumer generates its own RPriv and RPub and sends RPub along with SPub+RPriv encrypted service request to the provider.
  - The provider can decode SPub+RPriv only with RPub+SPriv keys.
  - The spoofing provider cannot possess SPriv key.

4.4 Spoofer (Provider) Unadvertises Legitimate Provider/Service

May Affect

- performance
- availability
- reliability

Implications

- denial of service possible by turning off provider access to all consumers

Possible Solutions

- The matchmaker uses public key encryption to enforce same-key advertise and unadvertise performatives.

4.5 Attacker (Provider) Advertises False Service

May Affect

- availability

Implications

- Provider may respond to service requests with false results.
- Provider may simply not answer requests with contracted response.

Possible Solutions

- The matchmaker periodically verifies service by issuing controlled request and observing response and server behavior.
- The consumer provides report of provider performance to the matchmaker, which may then de-register the offending provider.

5 Example: A Communications Network as a Compound Agent System

A complex agent system would rarely fit neatly into one of the patterns or styles illustrated in Chapter 3. In practice, agent systems are more likely to consist of combinations of these patterns: sometimes nested and sometimes including non-agent systems in their midst. To apply the appropriate scenarios, it is necessary to decompose the system into recognizable building blocks.[1] We illustrate the approach through a simple example of a long-distance communications network shown in Figure 5-1.

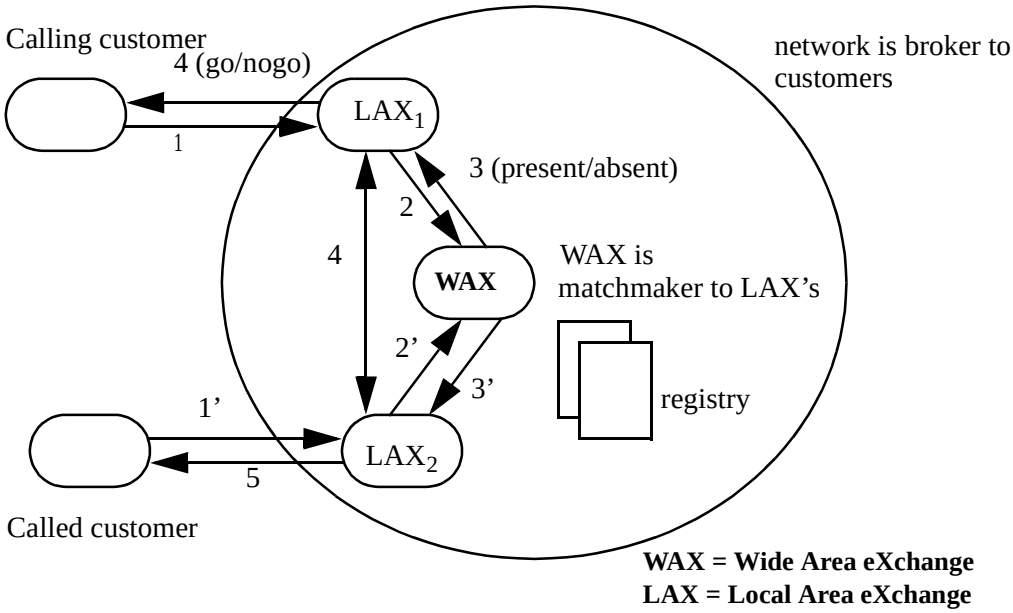

Figure 5-1: The Communications Network as a Combined Broker/Matchmaker

The example shows a communications network that subscribers can use to communicate with each other. In the figure, a "calling customer" (upper left corner) tries to establish a connection

1. The decomposition needs to go down at least to the level where we can identify patterns or styles to which we can apply scenarios similar to those illustrated in this report. The decomposition could go into increasingly more primitive components, all the way down to objects, processes, lines of code, etc., depending on the objective (e.g., code inspection to verify that a risk identified by a scenario has been mitigated).

with a remote customer (the “called customer,” in the bottom left corner). Figure 5-1 illustrates the following sequence:

1. The caller first requests its local area exchange (LAX_1) for directions to the called customer (arrow labelled “1”).
2. The caller’s local area exchange contacts a wide area exchange (WAX) for instructions to reach the called customer (arrow labelled “2”).
3. The wide area exchange maintains a registry of customers and their local area exchanges. Assuming it recognizes the caller customer, the WAX returns to the requesting exchange (LAX_1) the identity of the called customer’s local area exchange, LAX_2 (arrow labelled “3”).
4. LAX_1 can then contact LAX_2 to request establishing a connection with the called customer. The double-headed arrow labelled “4” suggests a protocol between the LAXs. LAX_2 then contacts the callee alerting it that it has a caller (arrow labelled “5”).
5. From then on, the two customers can carry out a dialog through their local exchanges without involving the WAX. Arrows labelled 1’, 2’, and 3’ suggest the symmetric behavior of the network. Any customer can call any other customer.

The complex behavior of the network can be explained in simpler terms by recognizing that the WAX/LAX complex acts as a broker between customers, but the WAX itself acts as a matchmaker between LAXs. A matchmaker (i.e., WAX) acts as a temporary intermediary between a requester (i.e., LAX_1) and a server (i.e., LAX_2). Once the identity of the server is forwarded to the requester, the matchmaker itself drops out of the picture, and the requester and server carry on their exchanges without further intervention of the matchmaker.

A broker (i.e., the WAX/LAX complex) also acts as an intermediary between a requester (i.e., calling customer) and a server (i.e., called customer), but it differs from the matchmaker in a significant way: It never drops out of the picture and remains involved throughout the operation. This behavior could be attributed to a desire for a more robust system—the WAX/LAX complex is presumably run by the communications company, and the matchmaker is more trusting of the local area exchanges. Subscribers, on the other hand, could exhibit erratic behaviors that could tie up valuable resources. The broker watches over their exchanges and takes proper actions if necessary. (For example, a customer refusing to “hang up” could result in extra charges to the other party.) The broker can intervene and break the connection when either party drops out.

In identifying the agent-based architectural “styles,” we are able to readily apply the earlier experiences, reasoning, and models developed in other analyses to this new situation. In particular, the generic scenarios outlined for agents types such as broker and matchmaker can be made more concrete for a specific instantiation of the agent type and we can then determine if the potential risks noted in the generic case are either handled or require further investigation.

We illustrate this with a few examples below—first in light of the external broker style evidenced and then in the internal matchmaker style.

In Section 4.1 we described the generic scenario “Attacker (consumer) issues massive requests for services.” In the instantiation of this scenario in the communications network example, there is an external brokering relationship involving a caller, a callee, and a network “agent,” where one or more callers can flood the network broker agent with requests to connect to callees. As outlined in the scenario, this type of action can seriously affect the performance and availability of the connection brokering service. In the communications network domain, obvious solutions or approaches to this problem include instantiating the generic “multiple-broker solution” by allocating connection-agents to sets of “callers” (local area switching). Other solutions include service limitations on individual request originators by the broker.

Each of the remaining scenarios can be similarly instantiated in the communications network example. The key concept here is that the characteristics of the specific situation (broker or matchmaker in this case) implies the relevance of a set of predefined generic scenarios. In the context of an architectural tradeoff analysis of a specific system, these generic scenarios provide example of where one should start. The applicability of one or more (in this case at least two) architectural styles can suggest specific cases, problems, solutions, and analyses appropriate to the overall analysis.

As has been mentioned earlier, the broker/matchmaker family is susceptible to a similar set of problems since they differ only in the way in which a negotiated interaction occurs—that is, directly or indirectly through an intermediary. Choosing (as in this case) a broker relationship between external agents (in this case, customers) and a matchmaker relationship for internal agents (in this case, local area exchanges) provides an excellent example of careful tradeoffs. A non-brokered interaction avoids the overhead of an intermediary buffering and exchanging messages that would be inappropriate in situations where all agents are trusted. On the other hand, allowing point-to-point unmediated connections might be inappropriate in cases where monitoring quality of content or quantity of service use might be crucial.

6 Conclusions

As part of the ATAM work, we are developing the concept of attribute-based architectural styles (ABAS). An ABAS [Klein 99] is a specific kind of architecture in which the components and their interactions, the attribute models and scenarios, and the attribute tradeoffs have been previously identified and analyzed. Collections of ABAS would make the method more efficient because they tell the architect and the other stakeholders what information is needed, what results will be available, and what tradeoffs are possible. The analysis of architectures built entirely or mostly from ABAS would not have to be carried out from the beginning.

The approach we suggest for the application of scenarios to the architecture of agent-based systems is to decompose the agent-system into more primitive fragments, exhibiting a structure and behavior of previously analyzed agent styles such as wrapper or broker, where wrapper and broker are illustrations of ABAS for which we have previously defined scenarios. In identifying the agent-based architectural "styles," we are able to readily apply the earlier experiences, reasoning, and models developed in other analyses to this new situation. Admittedly, the generic scenarios outlined for agent systems lack specificity because they are inspired by a generic architecture. When applied to a real system, specific domain and application requirements and constraints should be used to refine the scenarios and verify that the potential risks are either handled by the system or present issues requiring further investigation.

References

[Barbacci 95] Barbacci, M.; Klein, M.; Longstaff, T.; & Weinstock, C. *Quality Attributes* (CMU/SEI-95-TR-021, ADA307888). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, December 1995. Available WWW <URL: http://www.sei.cmu.edu/publications/documents/95.reports/95.tr.021.html>.

[Bond 88] Bond, A. and Gasser, L. (eds.). *Readings in Distributed Artificial Intelligence*. San Mateo, CA: Morgan Kaufman Publishers, 1988.

[Bradshaw 97] Bradshaw. J. M. (ed.). *Software Agents*. Cambridge MA: AAAI Press/MIT Press,1997.

[Brenner 98] Brenner, W.; Zarnekow, R.; & Wittig, H. *Intelligent Software Agents, Foundations and Applications*. Berlin, Germany: Springer-Verlag, 1998.

[Hayden 99] Hayden, S. C.; Carrick, C.; & Yang, Q. "Architectural Design Patterns for Multi-Agent Coordination." *Proceedings of the International Conference on Agent Systems '99. (Agents'99)*. Seattle, WA, May 1999. Available WWW <URL: http://www.cs.sfu.ca/~isa/pubs/index.html>.

[IAG 97] Intelligent Agents Group. *Software Agents: A Review.* Dublin, Ireland: A collaborative effort of Broadcom Ireland and the Computer Science Department of Trinity College. Available FTP <URL: ftp://ftp.cs.tcd.ie/pub/tech-reports/reports.97/TCD-CS-1997-06.ps.gz>.

[Klein 99] Klein, M. and Kazman, R. *Attribute-Based Architectural Styles* (CMU/SEI-99-TR-022). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, November 1999.

[Nwana 96] Nwana, H. S. "Software Agents: An Overview," 1-40. *Knowledge Engineering Review,* Vol. II, No. 3. Cambridge University Press, September 1996.

[Smith 80] Smith, R. G. "The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver." *IEEE Transactions on Computers* C-29, 12 (December 1980): 1104-1113.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. **AGENCY USE ONLY** (leave blank)	2. **REPORT DATE** October 1999	3. **REPORT TYPE AND DATES COVERED** Final
4. **TITLE AND SUBTITLE** Architectural Evaluation of Collaborative Agent-Based Systems		5. **FUNDING NUMBERS** C — F19628-95-C-0003
6. **AUTHOR(S)** Steve G. Woods, Mario R. Barbacci		
7. **PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)** Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213		8. **PERFORMING ORGANIZATION REPORT NUMBER** CMU/SEI-99-TR-025
9. **SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)** HQ ESC/XPK 5 Eglin Street Hanscom AFB, MA 01731-2116		10. **SPONSORING/MONITORING AGENCY REPORT NUMBER** ESC-TR-99-025
11. **SUPPLEMENTARY NOTES**		
12.a **DISTRIBUTION/AVAILABILITY STATEMENT** Unclassified/Unlimited, DTIC, NTIS		12.b **DISTRIBUTION CODE**

13. **ABSTRACT** (maximum 200 words)

The Architecture Tradeoff Analysis Method ^{SM} (ATAM^{SM}) is an architecture evaluation technique currently evolving at the Software Engineering Institute (SEI). ATAM has been applied to a number of command and control, real-time, and information systems. As collaborative, autonomous agents become a significant software technology, the demand for evaluating the quality attributes of the architectures of agent-based systems will increase. Very broadly, agents may be thought of as software entities that have the ability to undertake action autonomously in their particular embedded environment, according to a typically general set of requests or desired goals, and that are able to communicate with other agents as determined by their own initiative. Given an agent-system architecture, we need scenarios that could be applicable for conducting ATAM evaluations on instances of that agent architecture. This report identifies a few features in agent-based systems that could be used to classify agent-system architectures and to guide the generation of scenarios applicable to these architectures.

14. **SUBJECT TERMS** agent-based system, Architecture Tradeoff Analysis Method^{SM} (ATAM^{SM}), communications network, Contract-Net, quality attributes, software architecture, system scenarios			15. **NUMBER OF PAGES** 40 16. **PRICE CODE**
17. **SECURITY CLASSIFICATION OF REPORT** UNCLASSIFIED	18. **SECURITY CLASSIFICATION OF THIS PAGE** UNCLASSIFIED	19. **SECURITY CLASSIFICATION OF ABSTRACT** UNCLASSIFIED	20. **LIMITATION OF ABSTRACT** UL