
Architectural Evaluation of Collaborative Agent-Based Systems

Steve G. Woods, Mario R. Barbacci
Software Engineering Institute
Carnegie Mellon University

August 9, 1999

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 and 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. This is the principal difference between an architecture trade-off analysis and other software analysis techniques—that it explicitly considers the *side-effects* between multiple attributes, and permits principled reasoning about the trade-offs that inevitably result from such dependencies.

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 AI research community and are being used for increasingly more sensitive applications in which economic, privacy, safety, and other con-

cerns are of paramount importance. While in the past, achieving the desired functionality was challenging enough, 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 (ATAM) is an architecture evaluation technique currently evolving at the SEI. The input to ATAM 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 specific architectural view — the coordination model — of selected agent systems, and a collection of quality attributes as previously discussed. It is important to point out that 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 they consider 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 Trade-off Analysis Method (ATAM)

The Architecture Trade-off Analysis Method (ATAM) is an architecture evaluation technique currently evolving at the SEI. The input to ATAM consists of a system architecture and the perspectives of stakeholders involved with that system. ATAM relies on generating scenarios to assess the architecture.

- **Use cases** represent operational conditions that the architecture should support, and demonstrate the effectiveness of the architecture to satisfy these operating conditions.
- **Change cases** represent expected system modifications that may cause architectural changes, and demonstrate how efficiently the architecture can be modified.

ATAM utilizes stakeholder perspectives to derive a collection of scenarios giving specific instances for usage, performance requirements or growth, various types of failures, various possible threats, and various likely modifications. 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 causes 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 or 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 “trade-off” 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 **trade-off point** is a sensitivity point that is sensitive for multiple quality attributes (e.g., encryption takes time and affects latency).
- 2.a framework for reasoning about the system. The framework for reasoning about the system may take a variety of forms. It may be the discussion that follows from the exploration of a scenario, it may be a model or a portion of a model and a discussion of how that model might be analyzed when instantiated, or it may be a formula that represents how to calculate a 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 identify and document them.

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/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: The 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 what are 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 [e.g., Bradshaw 97, Brenner 98, IAG 97, Hayden 98, Nwana 96] outline a set of typical design patterns for collaboration among agents, but do not delve at all into the critical aspects of evaluation of 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 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 may undertake additional actions based on these sources. The protocols of verification of authenticity for data sources by individual agents is therefore an important concern in the evaluation of overall system behavior.

In addition to possibly misleading (deliberate or as a matter of course) information 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.

- **Resilience to modifiability of the environment**

Whereas in traditional software architectural evaluation, modifiability refers to the ability to change salient features of a software system, modifiability here refers to the ability of agent systems to adapt to modifications in their environment. Modifications may include such aspects as base communication-language changes (new versions for example) which are 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**

Possibly overlapping “Performance predictability”, Availability here refers to the relative degree of dependence of external agents on the behavior or services offered by the agent system while fault tolerance describes 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 C^2 system developers 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), the situation is more complicated when we attempt to identify agent-system architectures.

2.2 The Architectures of Agent-Based Systems

Software agents have evolved from the multi-agent systems (MAS) branch of the distributed AI taxonomy: [Bond 88] — *Cited by Brenner 98, check reference*

- Distributed Artificial Intelligence
 - Parallel Artificial Intelligence (PAI)
 - Distributed Problem Solving (DPS)
 - Multi-agent Systems (MAS)

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. [Nwana 96, p.2] notes, “[that] some cynics may argue that this strand arises because everybody is now calling everything an agent, therefore resulting inevitably in such broadness.” and 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 an 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** (the result is greater than the sum of the parts, exhibiting learning and cooperation or autonomy and cooperation behavior)
- **interface** (act on behalf of their owners, exhibiting autonomy and learning attributes)
- **mobile** (roaming on behalf of owners)
- **information/Internet** (managing information from distributed sources)

- **reactive** (stimulus/response, emerging behavior)
- **hybrid** (mix of previous)
- **smart** (exhibiting autonomy, learning, and cooperation)
- **heterogeneous** (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
- Exhibit some level of key attributes (learning, cooperation, mobility)

Where 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, 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, 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/users in evaluating nonfunctional, 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 trade-offs based on ever changing taxonomies of styles, we take a different approach. (1) We do not look at the functionality of the agents, rather we concentrate on external properties such as communication, cooperation, and coordination actions performed by the agents’ components and connections. This defines our “architecture” domain. (2) We look at the instantaneous architecture i.e. components and connections and the patterns of behavior of the components, (3) We identify a “central” component and conduct the analysis from the point of view of this component as the focus.

Specifically, we look at the number of inputs/outputs, the message processing actions (bundling, unbundling, translation), and other actions specific to the focus agent. [Hayden 98] makes a good first step in this direction but unfortunately that report still focuses on a “catalog” of coordination models with the inevitable overlaps and subtle distinctions we have observed in other taxonomies.

3 The Space of Architectural Choices

We define a space of 7 “binary” choices, arranged as columns in a “table of styles”, augmented with an additional column describing the main purpose of the focus agent.¹ The seven binary choices are:

- **Communication choices** (4 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 sits in the middle.
- **Processing choices** (3 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, a useful agent type in evolving systems. A wrapper agent allows a legacy application to be incorporated into a multiagent system (MAS). The legacy code is extended with agent capabilities by agentifying it [Hayden 98].

References		Agents-Legacy				Internal behavior of focus agent			Action
		1-1	1-M	M-1	M-M	bundling	unbundling	translating	
	wrapper [Hayden 98]		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 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.

The wrapper is usually domain- or system-specific and wraps around a single legacy system, although there might 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,

1. We do not suggest that all $2^7 = 128$ rows make sense or deserve to be recognized as a different type of agent styles. 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.

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 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 ACL 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); 3) typically there is message translation between the ACL formats and the format of the commands understood by the legacy.

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 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.

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

As we shall see, the role of the wrapper can be extended and acquire characteristics of other patterns like Monitor (watching and filtering the traffic) and Broker (requests from the legacy system will be forwarded to an appropriate agent).

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. .

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.

References		Agents-Legacy				Internal behavior of focus agent			Action
		1-1	1-M	M-1	M-M	bundling	unbundling	translating	
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	
	wrapper ++		N	Y	N	Y	Y	Y	Full functionality for a multi-agent wrapper.

Table 3-2: Wrapper Family

Reference		Subject-Subscribers				internal behavior of focus agent			Action
		1-1	1-M	M-1	M-M	bundling	unbundling	translating	
	monitor [Hayden 98]		Y	N	N	N	Y	N	Tracks agent state changes and notify subscribers.

Table 3-3: Monitor Family

The monitoring pattern is a compromise between the consistency that can be achieved in a tightly couple 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 subject state changes, it notifies the monitor and the monitor in turn notifies subscribing agents of the state change (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 only needs to notify the monitor, not a potentially unbound number of subscribers. The pattern does not protect from unforeseen consequences (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 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 98]		Y	N	N	N	N	Y	Keeps agent maps/ids
	Mediator [Hayden 98]		Y	N	N	Y	Y	Y	Keeps agent maps/ids and Active participation

Table 3-4: Matchmaker/Broker/Mediator Family

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. 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.

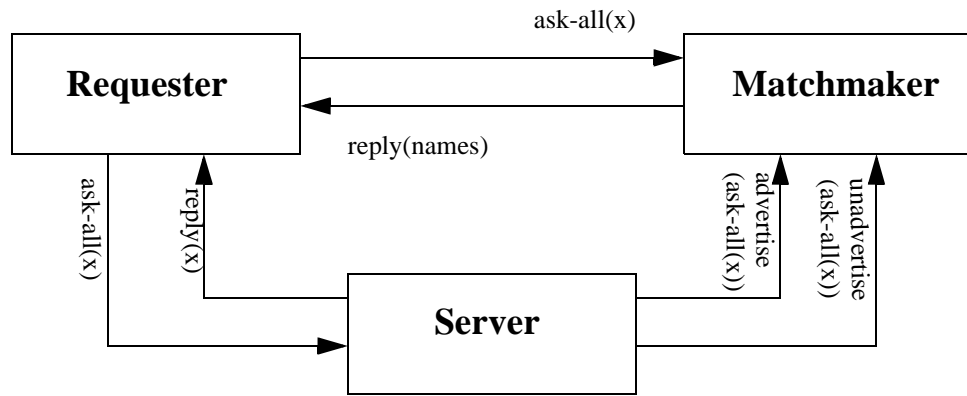


Figure 3-1: Operation of a Matchmaker [Brenner 98, Figure 4.3/17]

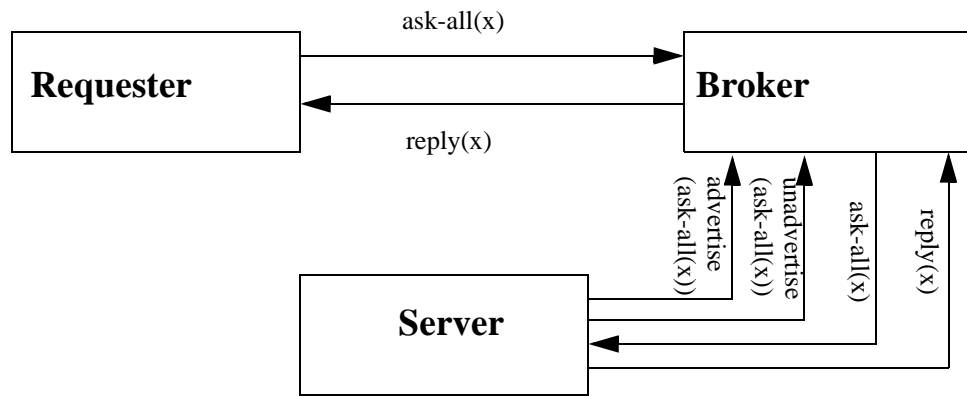


Figure 3-2: Operation of a Broker [Brenner 98, Figure 4.3/18]

The Mediator acts like the Broker with the difference that while a Broker simply arranges the Matchmaking between providers and consumers; a Mediator encapsulates agent interactions and maintain models of behaviors over time; these are the bundling/unbundling actions¹.

Broker presents a trade-off with Matchmaker in that the Broker insulates consumer and provider and thus 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 ongoing as well as in negotiation.

1. Hayden 98, page 7, says that a mediator is the appropriate “wrapper” for legacy systems that make requests of their own. The two patterns become confused because proper use of a legacy might impose limits on interaction patterns, i.e., a wrapper might be “mediator-like” even if the legacy does not make requests, just to prevent improper use of the legacy system.

Matchmaker limits its failure impact to the negotiation phase since previously negotiated interactions are undertaken between the consumer and the provider directly, but this flexibility is at the expense of revealing provider identities to consumers. 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 98) extend the standard Broker architecture by adding explicit “Registration” exchanges between consumers and Broker as well as between providers and Broker. This extension explicitly addresses security concerns indicated in some of the scenarios.¹

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 allows each agent to vary its behavior independently. Agents only need to be aware of 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 sequences).

Colleagues send and receive requests to/from the Mediator which routes the requests in accordance with a specific conversation model. The disadvantage of the pattern is that the Mediator becomes a 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

In Contract-net [Smith 80], a Manager (the focus) issues a request-for-proposals (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 bid has been rejected. The selected contractor performs the contracted work and informs the Manager upon completion.

Contract-net is particularly interesting as specifications for this protocol “hide” the existence of a client. The interaction (Figure 3-3) is defined entirely in the context of a Manager and possible multiple bidders. Implicit in the protocol is the existence of a client requesting a Manager locate a provider of a particular service.

1. Note however that “Register” is apparently not a current core KQML performative.

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

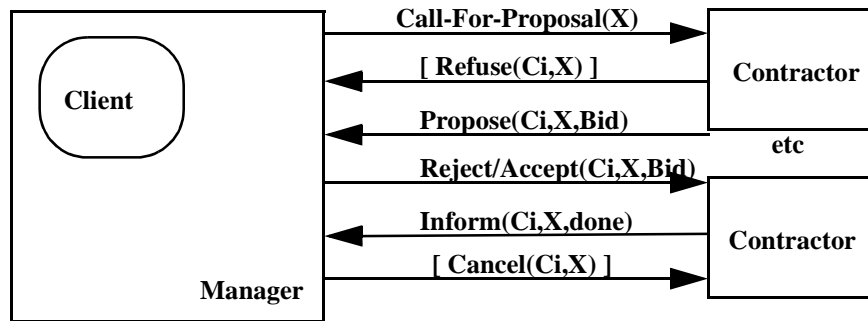


Figure 3-3: Operation of a simplified Contract-net

The Contract-net acts in a manner similar to 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 (the “bundling” and “translating” actions in the Contract-net represent the packaging of the client’s needs into an RFP).

3.5 Embassy Family

[Hayden 98] describes a mechanism for supporting a foreign agent’s access to the services of an agent system called “Embassy”, 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. .

The Embassy single-point-of-entry method is common to networks (gateways), and limits the scope of violation significantly and the cost of single-point-of-failure impacts to local/foreign interactions. Primary interactions for Embassy as described by [Hayden 98] as a generalization of the Broker — the Embassy is responsible for any semantic translation from foreign performative languages into local performatives and vice versa.

Reference	Foreing agent - Local agents				internal behavior of the focus agent			
	1-1	1-M	M-1	M-M	bundling	unbundling	translating	Action
Embassy [Hayden 98]		Y	N	N	Y	Y	Y	Controls access of foreign agent to local resources.

Table 3-6: Embassy Family

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: client requesting access, an embassy granting access, a 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 Scenarios for Agent-Based Systems

The following outlined scenarios are meant to indicate the possible range of issues applicable to the agent types described in Chapter 3. For brevity, we only discuss scenarios for Matchmakers, Brokers, or Mediators, although it will be apparent that many of the scenarios are generic and can apply to multiple agent systems.

Matchmaker differs from Broker primarily in the Matchmaker actually passes responsibility for service provision directly to the matched server and requesting client whereas Broker completely hides the identity of server from client and client from server. Contract net is a two-agent protocol, and the existence of an implied client for the Manager suggests a wide rang of possible variations involving this third party. For example, a successful negotiation of Manager and contractor might involve a direct contractor-to-client interaction that is neither specified nor prohibited (i.e., the Manager is also a Matchmaker).

Agent systems are susceptible to attacks in which an outside party intervenes in an inter-agent exchange. Clearly scenarios which involve hostile actions which require explicit knowledge of server by client or client by server are obviated in the Broker pattern. On the other hand, the existence of a specific intervenor in the communication between client and server might impact performance (increase in latency), availability (single point of failure) or even security of information since there are now two separate communication channels to be considered. 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:

Scenario description

A short explanation of the scenario...

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 Massive requests by consumers to broker

May Affect

- (performance, availability)

Implications

- Denial-of-Service
- Single-Broker solution for all services implies that one successful attack on Broker destroys all service allocation

Possible Solutions

- A multiple-broker solution would address this issue, however it requires that consumers have a more elaborated mode of retaining consumer-provider relationships - a clear trade-off between robustness/availability and complexity and overhead.
- A registration of consumer with 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 restricted to a single request type to same Consumer in a given time frame
- Monitor all Consumer/Provider interactions to limit Consumer actions to expected values, including volume of messages
- Matchmaker issues only single Provider addresses to Consumer in a given time frame
- Matchmaker notifies recommended Provider of request by Consumer in order to make service available for a limited time only

4.2 A provider fails after issuing service advertisement

May Affect

- (availability of service)
- (performance of Consumer if not handled)
- (reliability of Consumer if not handled)

Implications

- If Matchmaker database contains information about non-available service, a Consumer can obtain a handle to an unavailable service handle. If the Consumer times out after asking for the service may not pose a critical error, however, if the Consumer blocks, the system might hang.

Possible Solutions

- Matchmaker regularly pings service providers as assurance of presence
- Matchmaker requires regular re-advertisement of service
- If the Consumer times out, Matchmaker should be notified

4.3 A spoofer (customer) may acquire service and then seek to replace this service.

May Affect

- (security of service if attacker spoofs Provider)
- (reliability of service)
- (performance of Provider)

Implications

- Matchmaker database may contain erroneous Provider/service information and a Customer can obtain apparently valid Provider and receive erroneous result.
 - If Spoofer Provider delivers apparently valid response on contract the Consumer could expect contracted performance and not receive it
 - If Spoofer Provider fails to reply to contract request

Possible Solutions

- Do not provide explicit Provider process or port information to Consumer
- Use Public Key Encryption - for instance:
 - 1.Provider generates private key SPriv and registers public key SPub with Matchmaker.
 - 2.Consumer receives Provider identifier and SPub from Matchmaker.
 - 3.Consumer generates own RPriv and RPub and sends RPub along with SPub+RPriv encrypted service request to Provider.
 - 4.Provider can only decode SPub+RPriv with RPub+SPriv keys.

5.Spoofing Provider cannot possess SPriv key.

4.4 A spoofer (provider) issues “unadvertise” for valid provider and service

May Affect

- (performance, availability, reliability)

Implications

- Denial-of-Service possible through turning off Provider access to all Consumers

Possible Solutions

- Matchmaker using Public Key Encryption to enforce same-key Advertise and UnAdvertise

4.5 An 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

- Periodic Matchmaker verification of service by issuing controlled request and observing response and Server behavior
- Consumer report of Provider performance to Matchmaker which may decide when to de-regist

5 Example: A Communications Network

Figure 5-1 shows a long distance 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 with a remote customer (the “called customer”, in the bottom left corner).

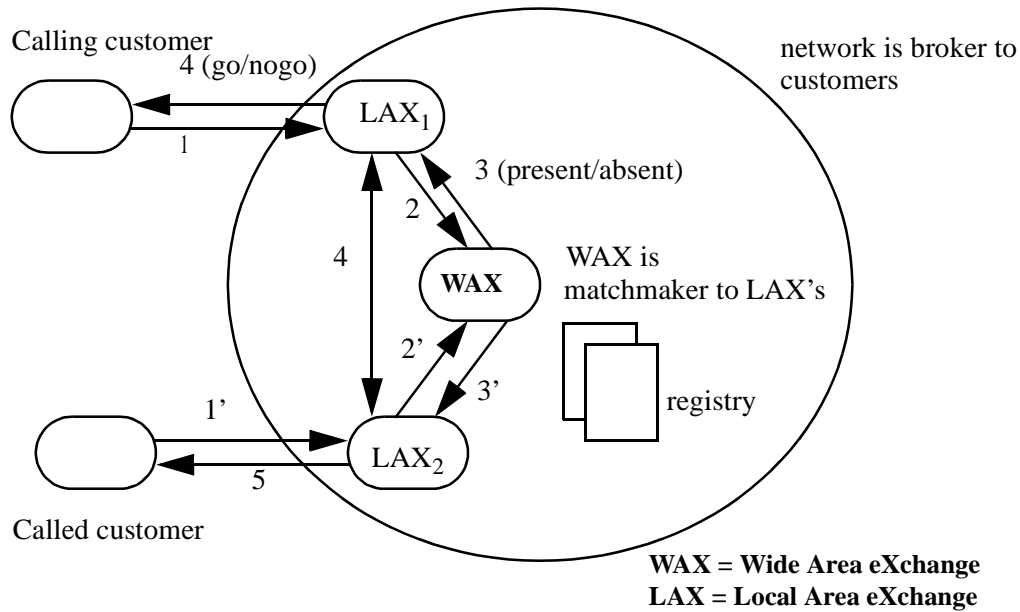


Figure 5-1: The communications network as a combined broker/matchmaker

- The caller first requests its local area exchange (LAX₁) for directions to the called customer (arrow labelled “1”).
- The caller’s local area exchange contacts a wide area exchange (WAX) for instructions to reach the called customer (arrow labelled “2”).
- The wide area exchange maintains a registry of customers and their local area exchanges. Assuming it recognizes the caller customer, WAX returns to the requesting exchange (LAX₁) the identity of the called customer local area exchange, LAX₂ (arrow labelled “3”).
- LAX₁ can then contact LAX₂ to request establishing a connection with the called customer (the double headed arrow labelled “4” suggests a protocol between the LAX’s). LAX₂ then contacts the callee alerting it that it has a caller (arrow labelled “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 LAX’s]. A matchmaker (i.e., WAX) acts as a temporary intermediary between a requester (i.e., LAX₁) and a server (i.e., LAX₂). Once the identity of the server is forwarded to the requester, the matchmaker itself drop 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 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 (e.g., 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 agent collaborations such as broker and matchmaker can be instantiated against the particular domain reflections of the generic mechanisms, and one can essentially verify that the potential problems are either handled or present issues for 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 4.3.1 the scenario “Massive requests by consumers to broker” is described. In the instantiation of this scenario in the communications network example to the external brokering relationship involving a caller, a callee and a network “agent”, on or more callers essentially flood the network broker agent with requests to connect to callees. As the scenario outlines, this type of action can seriously impact performance and availability of the connection brokering service. In the call-domain, obvious solutions or approaches to this problem include instantiating the generic “multiple broker solution” through a factoring of connection-agents among 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 pre-defined 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 as 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 handing back and forth 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 References

- Barbacci 95 Barbacci, M.; Klein, M.; Longstaff, T.; & Weinstock, C. “*Quality Attributes*” Technical Report CMU/SEI-95-TR-21 ADA307888. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1995
<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*, Morgan Kaufman Publishers, San Mateo, California, 1988.
- Bradshaw 97 Bradshaw, J.M. (ed.), *Software Agents*, AAAI Press/MIT Press, Cambridge Massachusetts, 1997
- Brenner 98 Brenner, W.; Zarnekow, R. & Wittig, H. *Intelligent Software Agents, Foundations and Applications*, Springer-Verlag, Berlin, Heidelberg, 1998
- Hayden 98 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.
<http://www.cs.sfu.ca/~isa>
- IAG 97 “*Software Agents: A Review*”, Technical Report ?, Intelligent Agents Group, a Collaborative Effort of Broadcom Ireland and the Computer Science Department of Trinity College, Dublin, Ireland.
<http://www.cs.tcd.ie/Brenda.Nangle/iag.html>
- Nwana 96 Nwana, H.S. “*Software Agents: An Overview*”, Knowledge Engineering Review, Vol. II, No. 3, pp. 1-40, September 1996, Cambridge University Press.
hyacinth@info.bt.co.uk
- Smith 80 Smith, R.G. : “*The Contract Net Protocol: High-level Communication and Control in a Distributed Problem Solver*”. IEEE Trans. Comput., Vol.C-29, No.12, pp.1104 -1113, 1980.