8. Brokering and Matchmaking for Coordination of Agent Societies: A Survey

Matthias Klusch[1] and Katia Sycara[2]

[1] German Research Center for Artifical Intelligence, Multiagent Systems Group, D-66123 Saarbrücken, Germany.
[2] Carnegie Mellon University, Robotics Institute, Pittsburgh, USA.
E-Mail: klusch@dfki.de, katia@cs.cmu.edu

8.1 Introduction

A vast majority of enterprises continue to operate in the physical world, doing business as usual, but increasingly adopt the Internet in every aspect of their business operations. Internet-based coordination and collaboration between supply chain partners is expected to create new value and increase the productivity and efficiency of both digital and physical product companies. To date, the Internet has already created a complete electronic economy which is rapidly growing and comprising, in particular, an open and globally accessible networked environment; interconnected electronic markets, online consumers, producers, and electronic intermediaries; and legal and policy frameworks.

Intermediaries in the Internet economy provide market-maker services, domain expertise, trust, visibility, assurance, and certification that enable buyers to choose sellers and products. They also provide the search, retrieval, and aggregation services that lower online transaction costs for businesses. Like intermediaries in the physical economy, intelligent middle-agents can be considered as electronic intermediaries in the digital economy[4]. These agents provide means of meaningfully coordinating activities among agent providers and requesters of services (information, goods, or expertise) in the Internet [8]. Their main task is to locate and connect the ultimate service providers with the ultimate requesters in open environments, that is to appropriately cope with the connection problem.

Various distributed and centralized settings exist to solve this problem. In general, we roughly distinguish three agent categories, service providers (P-agents), service requester (R-agents), and middle-agents. The basic process of capability-based mediation by middle-agents has the following form: (1) P-agents advertise their capabilities to middle-agents, (2) middle-agents store these advertisements, (3) a R-agent asks some middle-agent to locate and connect to P-agents with desired capabilities which may include complete transaction intermediation and other value-added services, and (4) the respective middle-agent processes this request against its knowledge on capabilities of registered P-agents and returns the result. The result may be either a subset of the stored advertisements with names of respective provi-

ders to contact, or the result of the complete transaction for the most suitable service.

While this process at first glance seems very simple, it is complicated by the fact that the Internet considered as an open social, information, and business environment is in a continual change. The driving force which precipitate or demand change may arise anywhere at any time. Dynamic changes concern, for example, location and heterogeneity of available resources and content as well as different user communities and societies of agents each of which is pursuing its goals that may conflict with the goals of others. In addition the number of different agents and multiagent systems that are being developed by different groups and organizations significantly excarbates the connection problem. Thus, the essential capabilities of any kind of middle-agents are to facilitate the interoperability of appropriate services, agents and systems, building trust, confidence and security in a flexible manner, and to comply to regulatory and legal frameworks when available. This depends on the specific requirements implied by given specific application, information, and system environment. Given the fact that different types of middle agents provide different performance trade-offs what types of middle-agents are appropriate depends on the application. The overall challenge of a middle-agent based solution is to fit in with the considered situation most effectively, efficiently and reliably.

The structure of the remainder of this chapter is as follows. Section 8.2 provides a general skill-based characterization of middle-agents. In particular, we will give a compact guide to the basic concepts of matchmaking and brokering for coordination of agent societies in the Internet by respective types of middle-agents. Section 8.3 gives examples of both coordination techniques as they are available in several multiagent systems which have been developed by different research labs and universities so far. Finally, section 8.4 concludes this chapter by briefly summarizing the main results.

8.2 Coordination of Agent Societies via Middle-Agents

8.2.1 Middle-Agents

The notions of middle-agents, matchmakers, brokers, facilitators, and mediators are used freely in the literature on the subject without necessarily being clearly defined. In the following we address this terminological issue. In general, middle-agents are agents that help others to locate and connect to agent providers of services [8]. Furthermore, any middle-agent can be characterized by its core skills of

1. providing basic mediation services to the considered agent society,
2. coordinating these services according to given protocols, conventions, and policies, and

3. ensuring reliable service mediation in terms of leveled quality of services as well as trust management within and across multiagent systems borders.

Provision of mediation services. In open information and trading environments in the Internet a middle-agent has to provide basic mediation services for the

1. processing of agent capability and service descriptions
2. semantic interoperation between agents and systems,
3. management of data and knowledge, and
4. distributed query processing and transactions.

We will briefly discuss each of these service types in the following and propose a respective service-oriented classification of middle-agents in section 8.2.2.

Processing of agent capability and service descriptions. Basic requirement to understand and automatically process requests for and descriptions of services and capabilities of agents is the agreed use of a (set of) common agent capability description language such as LARKS [41, 40], CDL [7], or XML-based service description frameworks, like RDF(S)[34] or eCo system's server and common business library (CBL) of generic XML document models for e-commerce [14]. In general, the middle-agent has to in real time parse, validate, understand and respectively process capability and service descriptions it receives. This is in order to efficiently determine which of the advertised services and capabilities of currently registered P-agents are most appropriate for a given request of an R-agent. The choice of a suitable matching mechanism certainly depends on the structure and semantics of the descriptions to be matched as well as on the desired kind and quality of the outcome of the matching process; it may rely, for example, on simple keyword and value matching, use of data structure and type inferences, and/or the use of rather complex reasoning mechanisms such as concept subsumption and finite constraint matching. Semantically meaningful matching requires the matching service to be strongly interrelated particularly with the class of services enabling semantic interoperation between agents. We describe selected approaches for specifying and matching capability and service descriptions with given requests in more detail in section 8.2.3.

Semantic interoperation. One main obstacle of a meaningful interoperation and mediation of services is the syntactic and semantic heterogeneity of data and knowledge the middle-agent does access and receive from multiple heterogeneous agents, information systems and sources. The functional capability of amiddle-agent to resolve such structural and semantic heterogeneities refers to the knowledge-based process of semantic brokering [36].

- *Knowledge-based reconciliation of heterogeneities.* Most methods to resolve semantic heterogeneities rely on using partial or global ontological knowledge which may be shared among the agents. This requires a middle-agent to provide some kind of ontology services for statically or dynamically creating, loading, managing, and appropriately using given domain-specific or common-sense ontologies as well as inter-ontology relations when it processes requests and data from different agents. For example, the agreed use of some common metadata catalogue and XML-based frameworks like Dublin Core and RDF(S)[34], respectively, a standardized ontology exchange language (OEL) such as XOL[46] or OIL[31] could be used by different agents in a multiagent system to represent and exchange individual ontological knowledge of given domains.
  The transformation into a unified knowledge representation supports the middle-agent to (a) understand the semantics of requests and advertised capability or service descriptions it receives from heterogeneous agents and (b) to reconcile discovered semantic heterogeneities. The type of application and mediation of the middle-agent requested determines which kind of ontology and reconciliation service is most appropriate. A comprehensive survey of methods and techniques for knowledge-based resolution of different types of heterogeneities can be found, for example, in [11].
- *Integration of information.* A middle-agent may create and maintain an integrated partially global view on available information for internal processing and restricted inspection by other agents and users on demand. Such an information model can, for example, take the static form of a federated schema in a multidatabase system [37] or a dynamically updated and categorized collection of capability and service descriptions annotated with relevant ontological information. The latter can be stored, maintained and queried like in XML-based data warehouses.

Management of data and knowledge. Basic internal services of the middle-agent include the efficient storage and management of data and knowledge about itself and other agents. Such data are, for example, sets of capability advertisements of registered P-agents, past and current requests of known R-agents, ontological knowledge shared among agents and other auxiliary data for internal processing. The corresponding databases, repositories and knowledge base of a middle-agent may be inspected by other agents and users for different purposes such as, for example, the specification of appropriate requests in the domain or monitoring activities according to given valid access restrictions and security policies.

Distributed query processing and transactions. In some cases the middle-agent has to put forward queries to multiple relevant external databases or other sources to gather information on behalf of its requesters. This requires the middle-agent to perform distributed query planning and execution of

transactions in collaboration with respective systems and agents.

Coordination of mediation services. The coordination of mediation by a middle-agent requires in particular its ability to meaningfully communicate with agents, systems, and users which are involved in the overall mediation process according to given protocols, conventions, and policies. As a basic additional means agent naming and registration is needed to enable the middle-agent to locate and identify its providers and requesters within and across multiple agent societies.

Agent registration and naming. Location of relevant agents by the middle-agent presumes that its clients are registered and can be named at any instant; this implies utilization of basic services of agent registration and naming for example agent name servers within the network domain covered by the middle-agent. Regarding scalability the middle-agent may also have to be able to dynamically cross register agent capabilities from one agent society to another so as to maintain maximum accessibility of involved societies to each other's capabilities. R-agents within one agent society may benefit from services and data provided by P-agents in another one without being aware of the fact that each of them belong to a different multiagent system. This can be achieved by the middle-agent which is initially contacted by the R-agents by exploiting its semantic interoperation services and appropriate collaboration with relevant middle-agents of and interoperators between heterogeneous agent societies [13].

Inter-agent interactions. Any coordinated interaction between the middle-agent, registered R-/P-agents and other middle-agents basically relies on the agreed use of one or more standardized agent communication languages (ACL) like FIPA ACL or KQML, given conversation protocols, and policies [23]. In communicating information among different agents it is essential to preserve the desired semantics of utterances transmitted via messages of an ACL. Both the intention of the message as well as its data content has to be understood correctly by the middle-agent and its clients to perform and support required service mediation, respectively.

Accessing sources of data and information. In case the middle-agent has to access database systems, knowledge bases or other sources of information by itself it can do so either via standardized APIs or transparent remote access methods such as JDBC/ODBC, OKBC[32], Java RMI, and CORBA, respectively. In a different approach the middle-agent may retrieve the required data in collaboration with so-called wrapper agents which are encapsulating the relevant systems and sources. The middle-agent forwards appropriate subqueries to the wrappers each of which transforming the received request to a query in the proprietary query language of the particular system and

returns the result of its execution in the desired format to the middle-agent. The middle-agent then has to merge all of these partial results for further processing. Such a scenario is in compliance with the paradigm of a mediator for intelligent information systems introduced by Wiederhold in 1992 [43].

Interfacing with users. In addition to the requirement of a middle-agent to appropriately communicate with its clients and relevant systems, it may also provide interface services to human users to let them, for example, browse available information they are interested in such as parts of a local domain or shared global ontology, sets of actual service descriptions, or registered service providers. These interface services have to be restricted according to given security policies and requirements.

Mediation protocols and policies. Any activities related to service mediation within an agent society can be organized and coordinated according to given protocols, conventions, and policies of interaction between agents. The middle-agent by definition plays a central role of initializing and coordinating the interaction among agents in collaborative or competitive settings. Most common forms of mediation protocols include matchmaking, brokering, and arbitration in negotiations between P- and R-agents, for example, at virtual marketplaces or auctions. The general protocols for brokering and matchmaking are described in section 8.2.3 below.

Reliability of mediation. Each client requires the contacted middle-agent to be reliable in terms of trust and quality of service. That means that the middle-agent should perform its designated mediation services at any given instant in a trustworthy manner and in compliance with given security and quality policies of individual agents or the considered agent society as a whole. This particularly requires the middle-agent to cope with different kind of policies in a coherent and consistent manner.

Quality of service. A middle-agent has to assure that the data it stores, processes and provides to the agents in the society comply with desired data quality requirements, standards, and policies. In this context data policy is the overall intention and direction of an agent with respect to issues concerning the quality of data products and services it will receive either from the middle-agent or the respective P-agents. The specification, management and evaluation of quality of mediated data and services may follow corresponding methods, standards and metrics used for data warehouse quality management [18] and software implementation quality such as the ISO 9126 standard.

Regarding the usage of mediated services and data, common quality requirements are that they are accessible and useful to the agents in terms of, for example, secure availability, interpretability and timeliness in order to detect changes and reasons for them. Other quality factors include correctness,

completeness, consistency, and redundancy of data as well as functionality, efficiency, maintainability and portability of the implementation of mediated services. Latter factors particularly also concern the quality of the mediation services of the middle-agent itself. For example, the provided mediation should be accurate, fault-tolerant to potential service failures, adaptive, time and resource efficient, and stable. It also requires the middle-agent to guarantee against loss of data or revenue as a result of its mediation which in turn, may increase the level of trust in its mediation services.

Trust management. Besides the client-sided demand to the middle-agent of providing quality of services there is also the essential requirement of a middle-agent to behave in a trustworthy manner to its clients in terms of guaranteeing desired data privacy, anonymity, and verification of claimed agent capabilities [45, 27]. In this sense, a middle-agent acts as a trusted intermediary among the agents of the considered agent society according to given trust policies of individual agents and the whole society.

Both internal data and knowledge of, and the computational processes of mediation executed by the middle-agent should be robust against external manipulation or attacks of malicious agents, systems, or users. On the other hand, clients of a middle-agent should have no incentive to misuse any information which has been revealed by the middle-agent during mediation. In this respect most common trust actions are authorization and verification of credentials of all parties which are involved in the mediation.

These actions have to take different trust relationships into account which may exist among the R-agents, middle-agents, and the P-agents within one or more connected agent societies. Can a contacted middle-agent, for example, be trusted by R- or P-agents to not resell (parts of) private profile information, copyrighted data, and so forth, to other middle-agents or P-agents it collaborates with potentially across multiagent systems borders? R-agents also might want to verify certain facts about relevant P-agents before contracting them either by itself or the middle-agent as a trusted intermediary.

In summary, there is an obvious need for all agents involved in the mediation process to use an appropriate trust model to analyse and assess the risks of and methods to prevent and counteract attacks against their data and knowledge. Models, methods and techniques supporting the establishment and management of mutual trust between a middle-agent and its clients in an open environment include

- the use of expressive trust establishment certificates useful for, e.g., binding agent identity to public key infrastructures such as IETF's SPKI [15, 39], and other standard security mechanisms and protocols, the use of mechanisms to bind agent names to their human deployers, so that the human would bear responsibility in case his/her agent misbehaves.
- the formal specification of agent trust policies, and

– the respective update, propagation, and transitive merge of trust matrices to calculate an overall trust relationship that accounts for the trust values in each and every individual trust relationship which is relevant to the considered mediation process [26]. An option is the application of distributed history-based reputation mechanisms to agent societies [47].

8.2.2 Types of Middle-Agents: Mediators, Brokers, and Matchmakers

Decker et al. [8] investigated different roles of middle-agents in the solution space of the connection problem from the standpoint of privacy considerations. The authors particularly examined knowledge about R-agent preferences, and P-agent capabilities; in this view specific requests and replies or actions in service of a request are interpreted as instances of preferences of R-agents and capabilities of P-agents, respectively. Both preference and capability information can initially be kept private at the R-agent, be revealed to some middle-agent, or be known by the P-agent. This leads to a categorization of some middle-agent roles as shown in the following table.

Preferences initially known by	Capabilities initially known by		
	P-agent only	P-agent + middle-agent	P-agent + middle-agent + R-agent
R-agent only	(Broadcaster)	"Front-agent"	Yellow-Pages, Matchmaker
R-agent + middle-agent	Anonymizer	Broker	Recommender
R-agent + middle-agent + P-agent	Blackboard	Introducer	Arbitrator

Table 1: Middle-agent roles categorized by initial privacy concerns [8].

For example, a broker agent understands but protects the initial privacy of capability and preference information of both the R-agent and P-agent; neither the R-agent nor the P-agent ever knows directly about the other in a transaction which is intermediated by the broker agent. In contrast, any R-agent can query capability information of registered P-agents at a matchmaker or yellow-page server such that this information is revealed initially to both requesters and providers. However, this categorization of middle-agent roles does not reflect the possibility of learning of preferences and capabilities over time.

In addition to this perspective on middle-agents restricted to initial privacy concerns we propose a more general skill-based, means service-oriented, classification of middle-agents. This classification is implied by the extent a middle-agent is providing mediation services as they are described in section

8.2.1. Figure 8.1 summarizes these service classes for middle-agents in general and for some common types of middle-agents in particular, namely mediators, brokers, and matchmakers. Latter is illustrated by means of object-oriented inheritance and instantiation of service classes for each of the agents. The corresponding classification matrix based on the differences in services is given in figure 8.2.

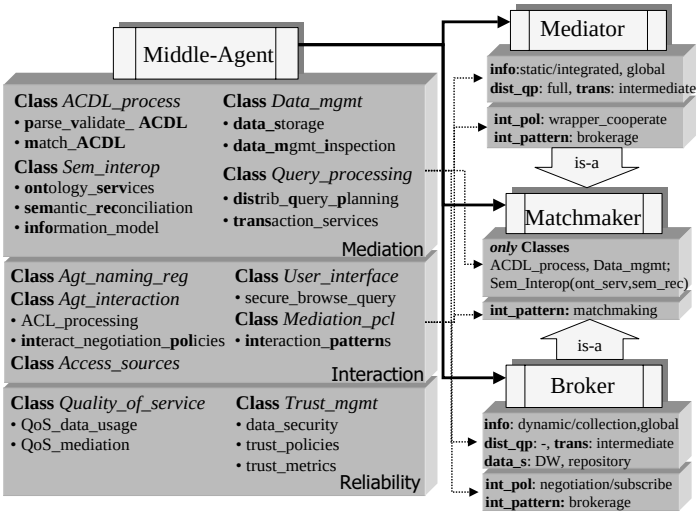


Fig. 8.1. Derived service classes of mediator, broker, matchmaker agents

In the following we briefly discuss mediator, broker and matchmaker agents in general, then focus on particular enabling techniques of brokering and matchmaking, and finally survey some of the prominent examples.

Mediator. A mediator agent in its original definition by Wiederhold (1992) [43] is dynamically and actively interfacing users (R-agents) to relevant data and knowledge resources. In this context, the term 'mediation' includes the processing needed to make the interface work, the knowledge structures that drive the transformation needed to transform the data to information, and any intermediate storage that is needed. In general, a mediator focuses on the provision of services devoted to semantic information brokering. That encompasses services to (1) dynamically determine information services and products; (2) arbitrate between consumer and provider agents by means of resolving different world-views (information impedance), and (3) dynamically create or compose information products by correlating or assembling components and services from various providers.

As mentioned above, the most common scenario of a mediator in an intelligent information system is that it acts as a central unit collaborating

Mediation Services	**Matchmaker**	**Broker**	**Mediator**
Class *ACDL_process*			
• **p**arse_**v**alidate_ **ACDL**	X	X	X
• **m**atch_**ACDL**	X	X	X
Class *Sem_interop*			
• **ont**ology_**serv**ices	X	X	X
• **sem**antic_**rec**onciliation	X	X	X
• **info**rmation_model		dynamic collection	static/integrated global
Class *Data_mgmt*			
• **data_s**torage/**m**gmt/**i**nspect	X	data warehouse repository	federated db/repository
Class *Query_processing*			
• **dist**rib_**q**uery_**p**lanning		X	X
• **trans**action_services		intermediate	intermediate
Class *Agt_interaction*			
• **int**eract_negotiation_**pol**icies	register/matching only no negotiation	negotiation subscribe, coop.	wrapper cooperation
Class *Mediation_pcl*			
• **int**eraction_**pattern**s	matchmaking	brokerage	brokerage
Class *Access_sources*		X	X

Fig. 8.2. Service-oriented classification of mediator, broker, matchmaker

with a set of wrapper agents (P-agents) each of which is providing access to and data of some available information source in a common data model. A mediator is creating and managing a partial global information model on the environment and other agents through knowledge based integration of received information from different sources, and offering domain and application specific value-added services. It takes requests from user agents (R-agents) and answers them either using its global information model or forwarding appropriate requests to the relevant wrapper agents based on respective distributed query planning and processing. In this sense information is gathered following a pull mechanism initiated by the mediator on demand.

In contrast to broker agents mediators do not simply collect information from different providers similar to data warehouses but integrate them into a global information model by resolving inconsistencies and conflicts. This model is typically hidden in the definition of the mediator, and is either statically or dynamically generated. In addition, a broker agent usually is considered not to be able to do any distributed query planning and processing. Examples of mediator-based information systems are Infomaster [12], TSIMMIS [9], Observer [38], MIX [3, 28], and SIMS/Ariadne [1].

Broker. A broker agent may actively interface R-agents to P-agents by intermediating requested service transactions. All communication between paired R- and P-agents has to go through the broker. It typically contacts (a set of) the most relevant P-agent, negotiates for, executes and controls appropriate transactions, and returns the result of the services to the R-agent.

This is in contrast to the functionality of a matchmaker agent but is subsumed by that of a mediator. But unlike mediators, broker agents typically do not provide a global, semantically integrated, consistent information model to their clients but store collected information maybe together with associated ontological annotations in some standardized data (structure) format in one (or multiple) appropriate repositories like in a data warehouse. In addition, a broker typically does not have a comparably extensive set of value-adding mediation services like a mediator; in this respect a broker may rely on different types of task-oriented agents it is able to collaborate and negotiate with. Broker may subscribe to certain P-agents if needed. Unlike a mediator agent, the interaction of brokers with other agents is neither restricted to wrappers or mediators nor committed to a fixed number of agents. This ability is particularly useful since brokers are usually acting in dynamic environments in which resources and agents may continually enter and leave the agent society. In such environments it turns out to be inappropriate to maintain a static pre-integrated global model which is valid for a rather closed society of agents and systems.

However, it is noteworthy that the semantics of the terms matchmaker and broker as well as mediator and broker are often used interchangeably in the literature; brokers are also often called facilitators. Examples of broker agents and broker-based systems include XML-based GMD Broker [16], OntoBroker [33], SHADE, and COINS [22].

Matchmaker. A matchmaker agent just pairs R-agents with P-agents by means of matching given requests of R-agents with appropriate advertised services of registered P-agents. In contrast to the functionality of both the broker and mediator it simply returns a ranked list of relevant P-agents to the requesting agent. As a consequence the R-agent has to contact and negotiate with the relevant P-agents itself for getting the services it desires. This direct interaction between R-agent and selected P-agents is performed independently from the matchmaker. It avoids, for example, data transmission bottlenecks or single point of failure at the matchmaker but increases direct communication overhead between matched R-/P-agents.

One could consider matchmaking as subsumed by brokering but it also extends it in the sense of giving the R-agent the full choice of selecting a (set of) P-agents a-posteriori out of the result of service matching. In settings with brokers instead, any R-agent choice a-priori by specifying respective constraints in its request for service to the broker. Examples of matchmaker-based multiagent systems and platforms include IMPACT [2], InfoSleuth [30], and RETSINA/LARKS [41, 40] each of which is described in section 8.3.

8.2.3 Coordination Techniques for Middle-Agents: Brokering and Matchmaking

The coordination of mediation activities within and across agent societies can be performed by a middle-agent, for example, via brokering or matchmaking. Both types of mediation imply different requirements and protocols for interaction among agents which are involved in it. According to the skill-based classification of middle-agents as described in section 8.2.1 any method and technique for capability and service matching can be utilized for implementing both the brokering and matchmaking. In addition, ontology services are typically used to perform meaningful automated reasoning on capability and service descriptions specified in a given ACDL.

In the following we briefly summarize the high-level interaction patterns for both the matchmaking and the brokering protocol. Additionally we adopt the approach by Wong and Sycara [44] to give a formal specification of the respective protocols by means of state-transition relation based input/output (IO) automata [25]. An IO automaton performs an action in a given state such that it may transition to a new state according to a given transition relation. This relation is described in a precondition-action-effect style to specify the changes that occur as a result of an action in the form of a simple pseudo-code that is applied to a pre-state to yield another state. Each agent is modelled as a process which, in turn, is specified as an IO automaton.

Matchmaking. When a service-providing agent registers itself with the middle-agent together with a description of its capabilities specified in an agreed ACDL, it is stored as an advertisement and added to the middle-agent's database. Thus, when an agent inputs a request for services, the middle-agent searches its database of advertisements for a service-providing agent that can fill such a request. Requests are filled when the provider's advertisement is sufficiently similar to the description of the requested service (service matching). As mentioned above the requesting agent has to directly interact with the relevant provider to obtain the service and data it desires (service gathering). Figure 8.3 gives an overview of the interaction pattern of this two-parted matchmaking process.

According to this interaction pattern an IO automaton for a matchmaker agent can be specified as follows.

Definition 1: *(Matchmaker automaton)*

Variables $provID, reqID, req, pref, capDescr, servParam, transID, match$ are drawn respectively from the set of P- and R-agent ids, requests, preferences, capability descriptions, service parameters, transaction ids, and an ordered list of service-request matches. The set of states and the signature of automaton A is denoted by $states(A)$ and $signature(A)$, respectively. The set of actions $acts(A)$ includes receive(msg), send(msg), and match(db, $(req, pref, reqID)$, k) denoting functions for receiving and sending messages of type 'advertise-capability', 'request-for-service', 'result', and returning a decreasingly ordered list of k tuples

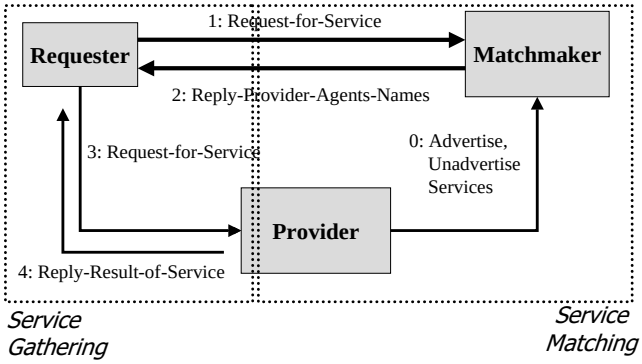


Fig. 8.3. Interaction pattern of capability and service matchmaking between agents

$((provID, capDescr), req, reqID)$ denoting the k best matching P-agents $provID$ (with respective capability description $capDescr$), respectively.
The set of transitions $trans(A) \subseteq states(A) \times acts(signature(A)) \times states(A)$. A middle-agent performs actions triggered by its input which transition A into a valid state and produce some output. Input and output are defined in terms of message types the middle-agent can receive and send, respectively.

Signature:
input: (advertise-capability $(provID, capDescr, servParam)$)
(request-for-service $(reqID, req, pref, k)$)
output(msg), $msg \in \{success, fail\}$
(result $(match)$), $match = [((provId, capDescr), req, reqID)]$

States:

- $capDB$: database of capability descriptions.
- $ackQ$: queue of acknowledgements containing tuples ($provID$, success/fail) each of which is indicating the result of an advertisement.
- $delQ$: queue of delegations containing tuples ($reqID$, $(req, pref, input)$) each of which is representing a request which has not yet been processed by the middle-agent.
- $matchQ$: queue of matching pairs of R-/P-agents containing tuples (($provID$, $reqID$), (req, $pref$)) with $provId$ indicating the P-agent that best matches the request req of R-agent $reqID$.

Transitions:
receive((advertise-capability $(provID, capDescr, servParam)$))
Precondition: -
Effect: $capDB$:= add($capDB$, $(provID, capDescr, servParam)$);

if add($capDB$, ($provID, capDescr, servParam$)) succeeds
then $ackQ$:= append($ackQ$, ($provID$,success)) else
$ackQ$:=append($ackQ$,($provID$,fail))
send(msg, $provID$)
Precondition: head($ackQ$) = ($provID, msg$),
Effect: $ackQ$:=tail($ackQ$).
receive(request-for-service ($reqID, req, pref$))
Precondition: -
Effect: $delQ$:= append($delQ$, ($reqID, req, pref$)).
match($capDB$, ($req, pref, reqID$), k)
Precondition: head($delQ$) = ($reqID, req, pref$).
Effect: $delQ$:= tail($delQ$);
$matchQ$:= append($matchQ$,
($reqID$, match($capDB$, ($req, pref, reqID$), k))).
send((result ($match$)), $reqID$)
Precondition: head($matchQ$) = ($reqID$, $match$)
Effect: $matchQ$:= tail($matchQ$);

□

Brokering. Based on the functional capability of broker agents informally described in section 8.2.2 the interaction pattern of brokering is shown in figure 8.4 below.

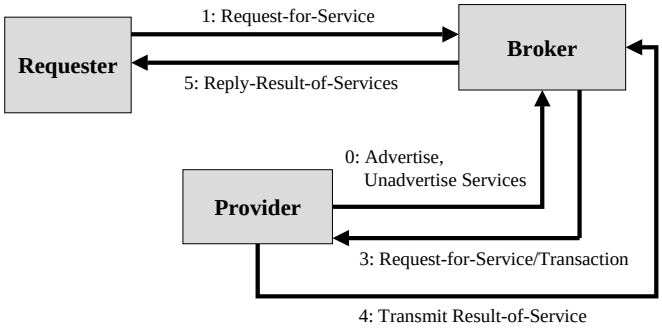


Fig. 8.4. Interaction pattern of capability and service brokering between agents

According to this basic interaction pattern an IO automaton for a broker agent can be defined as follows. Due to space constraints we omitted to include the option of the broker to individually negotiate services execution

with the relevant P-agents on behalf of its clients.

Definition 2: *(Broker automaton)*

See definition 1 for definition of states, variables, actions, message types. In addition, variables $input$, $ResultOfService$, z are drawn from the set of inputs needed for service executions, results they generate, and integer, respectively. The message type 'brokerage-RFS' and 'perform-SE' indicate that the middle-agent has to broker the request for service, and that each of the contacted P-agents has to perform requested service execution (transaction) given the id, and preferences of the requesting R-agent, respectively. It is assumed that the additional input needed for intermediation of transactions (such as payment information) is known to the broker agent. The states are meant to be in addition to the states already described in definition 1.

Signature:

input: (advertise-capability $(provID, capDescr, servParam)$)
(brokerage-RFS $(reqID, req, pref, input, k)$)
(result $(ResultOfService, transID)$)
output: (msg), $msg \in \{success, fail\}$
(perform-SE $((req, pref, input)$, $reqID, transID)$)
(result $(ResultOfService, req)$)

States:

- $AsyncResWait$: set of tuples (reqID, transID) indicating that the middle-agent is still waiting for the result of the transaction transID to be sent to the respective requester reqID.
- $resQ$: queue of service execution (transaction) results containing tuples ($reqID$, $ResultOfService$, $transID$) each of which includes the result of the transaction $transID$ ready to be transmitted to the respective R-agent $reqID$. set of tuples $(req, transID)$ each of which denotes the mapping of a request to its transaction id.

Transitions:

receive((advertise-capability $(provID, capDescr, servParam)$))
Precondition: -
Effect: $capDB$:= add($capDB$, $(provID, capDescr, servParam)$);
if add($capDB$, $(provID, capDescr, servParam)$) succeeds
then $ackQ$:= append($ackQ$, ($provID$,success))
else $ackQ$:= append($ackQ$,($provID$,fail))
send($msg, provID$)
Precondition: head($ackQ$) = $(provID, msg)$
Effect: $ackQ$:= tail($ackQ$).
receive(brokerage-RFS $(reqID, req, pref, k)$)
Precondition: -
Effect: $delQ$:= append($delQ$, $(reqID, req, pref)$), z:=k;
match($capDB$, $(req, pref, reqID)$, k)
Precondition: head($delQ$) = $(reqID, req, pref)$.
Effect: $delQ$:= tail($delQ$);
$matchQ$:= append($matchQ$,
($reqID$, match($capDB$, $(req, pref, reqID)$, k))).
send(perform-SE $((req, pref, input)$, $reqID, transID)$, $provID)$

Precondition: head($matchQ$) =
($reqID$, [(($provID, capDescr$), $req, reqID$)]); z¿0
Effect: $matchQ$:= tail($matchQ$); z–;
$AsyncResWait$:= $AsyncResWait \cup \{(reqID, transID, req)\}$;
receive((result ($ResultOfService, transID$)), $provID$)
Precondition: -
Effect: $resQ$:= append($resQ$,
($reqID, ResultOfService, transID$));
$AsyncResWait$:= $AsyncResWait \setminus \{(reqID, transID, req)\}$;
TR:= $TR \cup \{(transID, req)\}$
send((result ($ResultOfService, req$), $reqID$)
Precondition: head($resQ$) = ($reqID, ResultOfService, transID$);
($transID, req$) $\in TR$
Effect: $AsyncResWait$:= $AsyncResWait \setminus \{(reqID, transID)\}$;
TR:= $TR \setminus \{(transID, req)\}$;
$resQ$:= tail($resQ$).

□

8.3 Examples of Coordination via Service Matchmaking and Brokering

In the following we describe prominent examples of implemented multiagent systems and platforms which are coordinated by middle-agents via the mediation protocols for service matchmaking and brokering. These examples are IMPACT, RETSINA/LARKS, and InfoSleuth, respectively. Other related work is surveyed in section 8.3.4.

It is noteworthy that mediation across multiagent systems boundaries has still not received much attention though it is a crucial issue in open environments like the Internet. First steps in this direction include the RETSINA interoperator [13] between heterogeneous agent societies and multi-agent systems.

8.3.1 InfoSleuth

InfoSleuth [30] has been developed by MCC Inc. in Austin, Texas, USA. It is an agent-based system that can be configured to perform different information management activities in distributed applications such as in an evironmental data exchange network and competitive intelligence system. InfoSleuth offers a set of various agents with different roles as follows.

User agents act on behalf of users and interface them to the rest of the agent system, resource agents wrap and activate databases and other information sources; "broker agents" perform syntactic and semantic matchmaking; ontology agents collectively maintain a knowledge base of the different ontologies used for specifying requests and return ontological information on

demand; multi-resource agents process complex queries that span multiple heterogeneous resources, specified in terms of some domain ontology. Further agent roles concern task planning and execution, monitoring of data streams and system operations, and other specialized functions.

Any P-agent in a given InfoSleuth system announces itself to one or multiple matchmaker agents (called "broker agents") by advertising to it, using the terms and attributes described in a common shared ontology (called "infosleuth ontology") attributes and constraints on the values of those attributes. This special ontology is shared by all agents to use for specifying advertisements and requests. It contains concepts useful to define (a) what kind of content is accessible by an agent, (b) what services an agent can do for whom, (c) what are the interfaces to those services, (d) how well can an agent perform those services at the moment, and (e) other properties of an agent such as location, used communication protocols, etc.

Agent capabilities are represented in InfoSleuth at following four levels:

1. the agent conversations that are used to communicate about the service,
2. the interface to the service,
3. the information a service operates over, and
4. the semantics of what the service does.

Figure 8.5 shows an example of the specification of a pair of matching query and capability description in InfoSleuth adopted from [6].

Advertisement

capability env-subscription
ontology conversation
 class conversation
  slot conversation-type **in** {subscribe}
  slot language **in** {kqml}
ontology sql
ontology services
 class data-response
  slot language **in** {tuple-format}
  slot delivery **in** {http,inline}
 class subscription
  slot computation **in** {direct}
ontology environment
 class site
  slot contaminant
  slot remedial-response
  slot city
  slot state **in** {Texas}

Query

capability subscribe-to-capability
ontology conversation
 class conversation
  slot conversation-type **in** {subscribe}
  slot language **in** {kqml}
ontology sql
ontology services
 class data-response
  slot language **in** {tuple-format}
  slot delivery **in** {inline}
 class subscription
  slot computation **in** {direct}
ontology environment
 class site
  slot contaminant
  slot city **in** {Austin}
  slot state **in** {Texas}

Fig. 8.5. Example of query and capability descriptions in InfoSleuth

In this example a resource agent named 'tx-env-resource' is able to provide environmental contamination information related to contaminated sites

in Texas. In this respect it advertises generic capabilities such as the ability to be monitored, to be queried and subscribed to. Latter capability consists of a set of related ontology fragments according to the levels of capability representations mentioned above: the conversation ontology fragment specifies that the agent accepts conversations of the form used by subscriptions, using the language KQML. The sql ontology fragment specifies that queries have to be specified in the relational database query language SQL. The environment ontology fragment specifies classes and slots which contain data in the agent. Finally, the service ontology fragment specifies that the advertised service is accessed locally within the agent as well as other properties.

R-agents formulate request for services like the one shown in the example above in terms of the infosleuth ontology. The matchmaker then matches the request to P-agents whose advertisements correspond to the constraints specified in the request, and returns a recommendation containing those P-agents to the R-agent. Resource agents whose databases do not maintain data in terms of the common ontology access special mapping agents to perform appropriate ontology mapping actions.

Given a query over a single domain-specific ontology, a query agent coordinates the processing of the request by (a) collaborating with one or multiple matchmakers to identify relevant resource agents, (b) decomposing the query into a collection of subqueries each addressed to these agents, and (c) fusing the subquery results into an integrated answer to the original global query. Issues of subquery translation and execution being encapsulated with the respective resource agent. The resource agents may continually enter and leave the system, such that the query agent does not create and maintain a global integrated information model of the agent society. In summary, in InfoSleuth the query agent and the "broker" agent acts as a broker and a matchmaker, respectively.

Unfortunately, the exact and complete process of query decomposition for general queries (not just data base queries) or the process of matching in the InfoSleuth system has not yet been revealed in the literature. However, what is known from the literature is that service matching occurs at different levels: Syntactic, semantic, and pragmatic matching. In InfoSleuth, syntactic matching refers to the process of matching requests on the basis of the syntax of incoming messages which wrap the query; semantic matching refers to the process of matching the internal data structures of and constraints used in request and capability descriptions; pragmatic matching includes considerations such as the performance of the machine the agent is running on and security requirements.

8.3.2 IMPACT

The IMPACT (Interactive Maryland Platform for Agents Collaborating Together) platform [2, 17] has been developed at the University of Maryland,

College Park, USA. It supports multiagent interactions and agent interoperability in an application independent manner. For this purpose IMPACT provides following set of connected servers:

1. *Yellow-pages server* performs basic matchmaking among R- and P-agents based on two weighted hierarchies it maintains - a verb and a noun hierarchy of synonyms - and retrieval algorithms to compute similarities between given service specifications,
2. *registration server* for agent registration and maintaining an agent service index used by the yellow-pages server,
3. *type server* maintains a hierarchical ontology of standard data types and concepts, as well as
4. a *thesaurus* and a *human interface* allowing a human to access all the above servers in an IMPACT system.

The IMPACT software provides the infrastructure upon which different IMPACT agents may interact; multiple copies of it may be replicated across the network and get synchronized when needed.

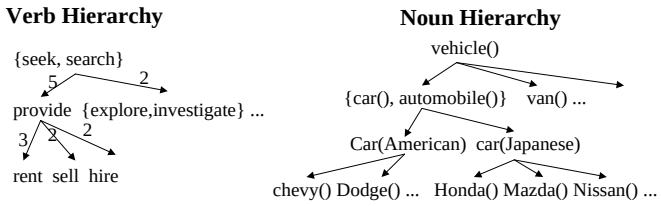
Any service provided by an IMPACT agent is specified in a special markup service description language. A service specification consists of (a) a service name in terms of a verb-noun(noun) expression such as rent:car(Japanese), (b) typed input and output variables, and (c) attributes of services such as usage cost, average response time to requests for that particular service, etc. Services may be either mandatory or discretionary. Agents may request services in one of the following forms.

- *k-nearest neighbor request*: Find the k-nearest service names (pairs of verb and noun term) such that there exists an agent who provides that service and identify this agent
- *d-range search*: Find all service names within a specified distance d.

Searching of appropriate services in IMPACT essentially relies on the exploitation of given weighted verb hierarchy and noun hierarchy each of which are special cases of the general concept of a term hierarchy. Similarity between verbs and nouns in the verb and noun hierarchy, respectively, is computed via a given distance function on paths of weighted edges in the hierarchies. A composite distance function then combines both distance functions to calculate the combined similarity value for two word pairs (verb,noun) of service names. The authors suggest to take addition as composite distance function. If a word cannot be found in the respective hierarchy a synonym will be searched in the thesaurus instead.

Figure 8.6 gives an example of the data structures used in IMPACT to process requests. Regarding these example data structures a request "Find (k =) 2 nearest agents that provide a service (rent, car()) of renting cars" will be processed as follows.

Firstly, an agent table is searched to find at most 2 agents that provide the exact service (rent, car()). Since this search returns the empty set the service



(Service) Agent Table

Verb	NounTerm	Agent
rent	car(Japanese)	ag1
rent	car(American)	ag3
rent	vehicle()	ag6
hire	Honda()	ag4
rent	car(Japanese)	ag2
sell	Mazda()	ag2
rent	vehicle()	ag3

Thesaurus

Word	Synonyms
vehicle()	car,carrier,van, ...
car(American)	chevy,neon,dodge,...
car(Japanese)	Honda,Mazda,Nissan,..
Seek	search,lookup
provide	supply,hand-over
rent	hire,lease
....	....

Fig. 8.6. Example of data structures used for service matchmaking in IMPACT

related word pair (rent, car()) is now relaxed to (rent, vehicle()) by taking an immediate neighbor of the verb 'rent' and the noun 'car' in the verb and noun hierarchy, respectively. The type car() is now instantiated using values specified in the data type hierarchy of the IMPACT type server which leads in this example to noun terms car(Japanese) and car(American). As an intermediate result we get the following list of information on relevant services in the form ((verb,noun),composite distance): ((rent, car(Japanese), 1), ((rent, car(American)), 1), ((provide, car()), 3). Searching the agent table for exact matching service names rent:vehicle() and rent:car(Japanese) is successful in this example and returns agent6 and agent1 as relevant P-agents.

8.3.3 RETSINA/LARKS

The RETSINA (Reusable Task Structure-based Intelligent Network Agents) [42, 35] multiagent infrastructure has been developed at the Carnegie Mellon University in Pittsburgh, USA. It consists of a system of three different reusable agent types that can be adapted to address a variety of different domain-specific problems.

Interface agents interact with the user, receive user input and display results. Task agents help users perform tasks by formulating problem solving plans and carrying out these plans through querying and exchanging information with other software agents. Information/resource agents provide intelligent access to a heterogeneous collection of information sources.

A collection of RETSINA agents forms an open society of reusable agents that self-organize and cooperate in response to task requirements. The RET-

SINA framework has been implemented in Java and is being used to develop distributed collections of intelligent software agents that cooperate asynchronously to perform goal-directed information retrieval information integration and planning tasks in support of a variety of decision making tasks.

Similar to the InfoSleuth system mediation in the RETSINA multiagent systems basically relies on service matchmaking. However, the specification of capability and service descriptions differ. The ACDL developed for matchmaking in RETSINA is called LARKS (Language for Advertisement and Request for Knowledge Sharing)[41]. Application domain knowledge in agent advertisements and requests can be currently specified as local ontologies written in a specific concept language ITL or by using WordNet. In the LARKS approach advertisements and requests are expressed using the same language. It is assumed that every LARKS specification is wrapped up in an appropriate KQML-like message by the sending agent indicating if the message content is to be treated as a request or an advertisement.

A service specification in LARKS is a frame comprised of the following slots:

1. *Context*: Keywords denoting the domain of the description,
2. *Types*: User-defined data types found in the signature definitions,
3. *Input* and *Output*: Input and output parameter declarations defining the signature of the operation,
4. *InConstraints* and *OutConstraints*: Logical constraints on input/output variables (pre-/post conditions),
5. *ConcDescriptions*: Descriptions of used potentially disambiguating words in terms of the concept language ITL, or keyword phrase, and
6. *TextDescription*: A free text description of the agent's capabilities.

Table 2 shows an example of both a request and matching capability description. In this example a P-agent advertises its capability to provide a list of deployed AWAC air combat missions which have been launched in a given time interval. This advertisement matches with the shown request for agents that are capable of providing information on any kind of air combat missions launched in a given time interval. The request is also formulated in LARKS by simply specifying a desired agent capability. Constraints on the input and output of a capability description or request are specified as conjunction of finite Horn clauses. Both the request and advertisement contain ontological annotations of disambiguating word 'Mission' by the concepts 'AirMission' and AWAC-AirMission', respectively, each of which is defined in the Conc-Descriptions slot in terms of a common KL-ONE [5] like concept language based on a shared minimal vocabulary.

LARKS is fairly expressive and capable of supporting automated inferences. The implemented matchmaking process for LARKS specifications employs different techniques from information retrieval, AI, and software engineering to compute the syntactical and semantic similarity among advertised and requested agent capability descriptions. The matching engine

contains five different filters for (1) keyword-based context matching, (2) TF-IDF based profile comparison, (3) concept-based similarity matching, (4) type-inference rule-based signature matching, and (5) theta-subsumption based constraint matching of finite Horn clauses. Any user may individually configure these filters to achieve the desired tradeoff between performance and matching quality. In the following we briefly summarize each of the filters of the proposed LARKS matchmaking process; it is described in more detail in [40].

ReqAirMissions	
`Context`	Attack, Mission*AirMission
`Types`	Date = (mm: Int, dd: Int, yy: Int), DeployedMission = ListOf(mType: String, mID:String‖Int)
`Input`	sd: Date, ed: Date
`Output`	missions: Mission
`InConstraints`	sd <= ed.
`OutConstraints`	deployed(mID), launchedAfter(mID,sd), launchedBefore(mID,ed).
`ConcDescriptions`	AirMission = (and Mission (atleast 1 has-airplane) (all has-airplane Airplane) (all has-MissionType aset(AWAC,CAP,DCA,HVAA)))
`TextDescription`	capable of providing information on deployed air combat missions launched in a given time interval

AWAC-AirMissions	
`Context`	Combat, Mission*AWAC-AirMission
`Types`	Date = (mm: Int, dd: Int, yy: Int) DeployedMission = ListOf(mt: String, mid:String‖Int, mStart: Date, mEnd: Date)
`Input`	start: Date, end: Date
`Output`	missions: DeployedMission;
`InConstraints`	start <= end.
`OutConstraints`	deployed(mID), mt = AWAC, launchedAfter(mid,mStart), launchedBefore(mID,mEnd).
`ConcDescriptions`	AWAC-AirMission = (and AirMission (atleast 1 has-airplane) (atmost 1 has-airplane) (all has-airplane aset(E-2)))
`TextDescription`	capable of providing lists of deployed AWAC air combat missions launched in some given time interval

Table 2: Example of an advertised and requested agent capability description in LARKS.

The *context matching* consists of two consecutive steps:

1. For every pair of words u, v given in the `Context` slots compute the real-valued word distances $d_w(u,v) \in [0,1]$. Determine the most similar matches for any word u by selecting words v with the minimum distance value $d_w(u,v)$. These distances must not exceed a given threshold.
2. For every pair of most similar matching words, check that the semantic distance among the attached concepts does not exceed a given threshold.

The *comparison of the profiles* of LARKS specifications treated as documents relies on a standard technique from the information retrieval area, called term-frequency-inverse document frequency weighting (TF-IDF). The similarity $dps(Req, Ad)$ of a request Req and an advertisement Ad under consideration is then calculated by :

$$dps(Req, Ad) = \frac{Req \bullet Ad}{|Req| \cdot |Ad|}$$

where $Req \bullet Ad$ denotes the inner product of the weighted keyword vectors. If the value $dps(Req, Ad)$ does exceed a given threshold $\beta \in \mathbf{R}$ both documents pass the profile filter. For example, the profiles of both specifications in the example are similar with degree 0.65.

The matchmaker then checks if the declarations and constraints of both specifications for a request and advertisement are sufficiently similar. This is done by a pairwise comparison of declarations and constraints in two steps: Similarity and signature matching.

Similarity matching relies on a combination of distance values as calculated for pairs of input and output declarations, and input and output constraints. Each of these distance values is computed in terms of the distance between concepts and words that occur in their respective specification section. These values are computed at the time of advertisement submittal and stored in the matchmaker database. The overall similarity value among two specifications in LARKS is computed as the average of the sum of similarity computations among all pairs of declarations and constraints.

The *signature filter* first considers the declaration parts of the request and the advertisement, and determines pairwise if their signatures of the (input or output) variable types match according to a given set of type inference rules. Both filters, the similarity and signature matching are used to determine if considered LARKS specifications syntactically match.

Finally, the matchmaker checks if both specifications *semantically match* in terms borrowed from the software engineering area. A software component description D_2 'semantically plug-in matches' another component description D_1 if (1) their signatures match, (2) the set of input constraints of D_1 logically implies that of D_2, and (3) set of output constraints of D_2 logically implies that of D_1. In our case the logical implication among constraints is computed using polynomial and sound θ-subsumption checking for Horn clauses [29].

8.3.4 Other approaches to service description and matching

Other approaches on (annotated) specification and matching of descriptions of capabilities and services via standardized markup languages, ontologies, and agent capability description languages include SHOE [24], OntoBroker [33], Osirix [10], GMD XML Broker [16], MIX [3], eCo system [14], and JAT/CDL [7], respectively.

JAT/CDL. The capability description language (CDL) [7] has been developed at the university of Edinburgh. Capabilities of and requests for services are described in CDL either in terms of achievable objectives or as performable actions. Figure 8.7 below shows an example of a request specified as a task description and a capability description in CDL. The latter describes the capability of a hospital to move patients to its location via ambulance and treat them as injured persons. The agent who is representing this hospital also advertises that its problem-solving behavior is complete with respect to this capability, i.e. that, if there is a solution to a problem it will find it. In the context of the particular capability in the example this implies that the hospital will get only injured people.

A request of service is specified in CDL as a task description; for example, the requested capability defined in the task description shown in figure 8.7 is subsumed by the capability of the hospital agent described above. For the purpose of automated logic-based reasoning on capability descriptions the statements are formally specified in a first-order predicate logic based format like KIF [21].

Advertisement:

```
(capability
      :isa                move
      :properties         (complete)
      :state-language fopl
      :input
        ((To Hospital2)(Ambulance ?a))
      :input-constraints
        ((elt ?thing Person)
         (Is ?thing Injured) )
)
```

Query:

```
(task
      :isa                move
      :properties         (complete)
      :state-language fopl
      :input
        ((Thing JohnSmith)
         (From PowerPlant)
         (To Hospital2)
      :input-constraints
        ((elt JohnSmith Person)
         (Is JohnSmith Injured) )
)
```

Fig. 8.7. Example of an advertised capability description and query in CDL

Logic-based reasoning over descriptions in CDL is based on the notion of capability subsumption or through instantiation. *Capability subsumption* in CDL refers to the question whether a capability description can be used to solve a problem described by a given task description in CDL: A capability C subsumes a task T if

1. in the result of performing C all output constraints of T are satisfied, means if the capability C achieves the desired state as specified in T, and
2. if in the situation that precedes the result of performing C, all input constraints of C are satisfied, means that capability C is applicable.

It is assumed that there is a model-theoretic semantics defined for every state language used to express constraints in CDL descriptions. This implies that reasoning relies on *model-based constraint satisfaction*. A constraint is satisfied in a situation if the model corresponding to the situation is a model of the expression representing the constraint. Regarding capability subsumption this means that the input constraints of a task description T define a set of models, one of which is corresponding to the actual situation before the capability is to be applied. Similarly, the output constraints of T define a set of models, all of which correspond to situations in which the objective has been achieved. Therefore a capability C subsumes a task T (a) if every model of T's input constraints is also a model of the C's input constraints (input match condition), and (b) if every model of C's output constraints is also a model of T's output constraints. This definition of capability subsumption resembles that of semantic plug-in matching of LARKS descriptions as described in section 8.3.3 above.

Both the CDL and the proposed matching of CDL descriptions have been implemented in Java and tested in selected application scenarios for brokering in multiagent systems developed in JAT (Java Agent Template)[19]. In summary, CDL is to some extent similar to LARKS but the matching methods proposed for each of the languages differ significantly in terms of the process and quality of reasoning.

eCo CBL. The eCo system's [14] common business library (CBL) consists of business descriptions and forms represented in an extensible, public set of XML building blocks that service providers in the domain of electronic commerce can customize and assemble. Semantics of the CBL are derived through analysis of industry standards, such as EDI X12850 for specification of purchase orders. This led to a base set of common terms, term mappings in the domain of e-commerce, and corresponding XML data elements, attributes, and generic XML document type definitions (DTD). Any request for a service is transformed to an appropriate standard DTD which then can either be used to find similar DTDs of available service descriptions or to produce a stream of information events routed to and processed by business applications.

SHADE and COINS. The SHADE and COINS [22] are matchmakers based on KQML. The content language of COINS allows for free text specification of services and its matching algorithm utilizes the tf-idf information retrieval method. The content language of the SHADE matchmaker consists of two parts, one is a subset of the knowledge interchange format (KIF), another is a structured logic representation called MAX. MAX uses logic frames to declaratively store the knowledge. SHADE uses a frame like representation and the matcher uses a prolog like unifier.

Other techniques and formalisms related to capability descriptions. The problem of capability and service descriptions can be tackled at least from the following different approaches.

1. *Software specification techniques.*
   Agents are computer programs that have some specific characteristics. There are numerous works for software specifications in formal methods, like model-oriented VDM and Z, or algebraic-oriented Larch. Although these languages are good at describing computer programs in a precise way, the specification usually contains too much detail to be of interests to other agents. Besides, those existing languages are so complex that the semantic comparison between the specifications appears to be impossible. The reading and writing of these specifications also require substantial training.
2. *Action representation formalisms.*
   Agent capability can be seen as the actions that the agents perform. There are a number of action representation formalisms in AI planning like the classical one STRIPS. The action representation formalisms are inadequate in our task in that they are propositional and not involving data types.
3. *Concept languages for knowledge representation.*
   There are various terminological knowledge representation languages. However, ontology itself does not describe capabilities. On the other hand, ontological descriptions provide auxiliary concepts to assist the specification of the capabilities of agents.
4. *Database query capability description.*
   The database query capability description technique is developed as an attempt to describe the information sources on the Internet, such that an automated integration of information is possible. In this approach the information source is modeled as a database with restricted querying capabilities.

8.4 Conclusions

The Internet is an open system where heterogeneous agents can appear and disappear dynamically. As the number of agents on the Internet increases,

there is a need to define middle-agents to help agents locate others that provide requested data and services. In this chapter we introduced several different classes of basic services any middle-agent needs to appropriately instantiate to perform a meaningful, effective, and reliable mediation between agents within and across multiple agent societies.

We have proposed a classification of middle-agents with respect to these service classes, presented a compact description of service matchmaking and brokering, and, finally, surveyed some prominent examples of multiagent systems and platforms which are coordinated by special kinds of middle-agents, namely broker and matchmaker agents. A comparative overview of selected matchmaker, broker, and mediator agents according to the service-oriented classification of middle-agents is given in figure 8.8.

Mediation Services	**IMPACT**	**InfoSleuth Broker**	**RETSINA Matchmaker**	**GMD Broker**	**MIX Mediator**
ACDL processing	HTML-like	frame	LARKS frame	XML	XML
• ACDL descriptions • matching	k-nearest neighbor	structural matching	TF-IDF, plug-in, type inference, constraint match, concept subsumption	DTD matching	DTD matching
Semantic interoperation					
• ontology services	verb/noun hierarchy, thesaurus	infosleuth ontology	WordNet, ITL concepts	XML namespaces	
• semantic reconciliation	synonym, word distance	synonym, value matching	synonym, word distance, subsumption		
• information model	global, collection	global, collection	part. integrated collection	global, collection	global, integrated
Data management	X	X	concept base, databases	XML data warehouse, XQL	XML schema, views, BBQ
Query processing					
• distributed query planning • transaction_services					X
				intermediate	intermediate
Agent interaction	register, matching	register, matching	register, matching	searchbots cooperation	wrapper cooperation
Mediation_protocol	matchmaking	matchmaking	matchmaking	brokerage	brokerage
Access sources				(search bots)	(wrapper)

Fig. 8.8. A service-oriented comparison of selected matchmaker, broker and mediator agents

Issues for future research in the domain of mediation via middle-agents include, for example, approaches to cope with the problems of trust establishment and management by middle-agents; scalability in terms of interoperation across heterogeneous agent societies; a thorough analysis and development of efficient methods supporting distributed real-time collaboration between different types of middle-agents [20]; investigations in system load

balancing using middle-agents; standardized compositional design of middle-agents according to the proposed service-oriented classification enabling, e.g., component-based re-use and prototyping of such agents for different application scenarios.

Main application domains of assisted coordination of multiagent systems by middle-agents are intelligent cooperative information systems and agent-mediated electronic business in the Internet and Web. Farther in the future one could imagine middle-agents coordinating different agent societies which are involved in, for example, inter-planetary information networks, UMTS cell phone videoconferences, traffic management, and negotiation of content provided by different information providers for embedded databases in mobile information appliances.

8.5 References

1. Y. Arens, C. A. Knoblock, and C. Hsu. Query processing in the SIMS information mediator. In Austin Tate, editor, *Advanced Planning Technology*. AAI Press, 1996.
2. K. Arisha, T. Eiter, S. Kraus, F. Ozcan, R. Ross, and V.S.Subrahmanian. Impact: Interactive maryland platform for agents collaborating together. *IEEE Intelligent Systems*, 14(2), 2000.
3. C. Baru, A. Gupta, B. Ludascher, R. Marciano, Y. Papakonstantinou, P. Velikhov, and V. Chu. Xml-based information mediation with mix. In *ACM Conf. on Management of Data (SIGMOD'99), Philadelphia, USA*, 1999.
4. A. Barua, A.B. Whinston, and F. Yin. Value and productivity in the internet economy. *IEEE Computer*, May 2000.
5. R.J. Brachman and J.G. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9(2), 1985.
6. A. Cassandra, D. Cassandra, and M. Nodine. Capability-based matchmaking. In *Agents-2000 Conference on Autonomous Agents, Barcelona*. ACM Press, 2000.
7. CDL. Capability Description Language. http://www.aiai.ed.ac.uk/ oplan/cdl/.
8. K. Decker, K. Sycara, and M. Williamson. Middle-agents for the internet. In *IJCAI-97 International Joint Conference on Artificial Intelligence, Nagoya, Japan*, 1997.
9. H. Garcia-Molina et al. The TSIMMIS approach to mediation: Data models and languages. In *Workshop NGITS-95, ftp://db.stanford.edu/pub/garcia/1995/tsimmis-models-languages.ps*, 1995.
10. Rabarijaona et al. Building and searching an XML-based corporate memory. *IEEE Internet Computing*, May/June 2000.
11. W. Kim et al. On resolving schematic heterogeneity in multidatabase systems. *Distributed and Parallel Databases*, 1:251–279, 1993.
12. M.R. Genesereth, A.M. Keller, and O. Duschka. Infomaster: An information integration system. In *ACM SIGMOD Conference*, May 1997.
13. J.A. Giampapa, M. Paolucci, and K. Sycara. Agent interoperation across multiagent system boundaries. In *Agents-2000 Conference on Autonomous Agents, Barcelona*. ACM Press, 2000.
14. R.J. Glushko, J.M. Tenenbaum, and B. Meltzer. An XML framework for agent-based e-commerce. *Communications of the ACM*, 42(3), March 1999.
15. A. Herzberg, Y. Mass, J. Mihaeli, D. Naor, and Y. Ravid. Access control meets public key infrastructure, or: Assigning roles to strangers. In *IEEE Symposium on Security and Privacy*, May 2000.
16. G. Huck, P. Fankhauser, K. Aberer, and E.J. Neuhold. Jedi: Extracting and synthesizing information from the web. In *Conference on Cooperative Information Systems CoopIS'98*. IEEE Computer Society Press, 1998.
17. IMPACT. Interactive Maryland Platform for Agents Collaborating Together. http://www.cs.umd.edu/projects/impact/.

18. M. Jarke, M.A. Jeusfeld, C. Quix, T. Sellis, and P. Vassiliadis. Metadata and data warehouse quality. In M. Jarke, M. Lenzerini, Y. Vassiliou, and P. Vassiliadis, editors, *Fundamentals of data warehouses*. Springer, 2000.
19. JAT. Java Agent Template. http://cdr.stanford.edu/ABE/JavaAgent.html.
20. S. Jha, P. Chalasani, O. Shehory, and K. Sycara. A formal treatment of distributed matchmaking. In *2nd Conference on Autonomous Agents (Agents 98),Minneapolis, MN*, May 1998.
21. KIF. Knowledge Interchange Format. http://logic.stanford.edu/kif/.
22. D. Kuokka and L. Harrada. On using kqml for matchmaking. In *CIKM-95 3rd Conf. on Information and Knowledge Management*. AAAI/MIT Press, 1995.
23. Y. Labrou, T. Finin, and Y. Peng. Agent communication languages: The current landscape. *IEEE Intelligent Systems*, March/April 1999.
24. Sean Luke, Lee Spector, David Rager, and James Hendler. Ontology-based web agents. In *1st International Conference on Autonomous Agents*, 1997.
25. N. Lynch. *Distributed algorithms*. Morgan Kaufman, 1996.
26. D.W. Manchala. E-commerce trust metrics and models. *IEEE Internet Computing*, 4(2), March/April 2000.
27. Y. Mass and O. Shehory. Distributed trust in open multi-agent systems. In *Autonomous Agents 2000 Workshop on Deception, Fraud and Trust in Agent Societies, June*, 2000.
28. http://www.npaci.edu/DICE/mix-system.html.
29. S. Muggleton and L. De Raedt. Inductive logic programming: Theory and methods. *Logic Programming*, 9(20), 1994.
30. M. Nodine, J. Fowler, T. Ksiezyk, B. Perry, M. Taylor, and A. Unruh. Active information gathering in InfoSleuth. *Cooperative Information Systems*, 9(1/2), 2000.
31. OIL. Ontology Interchange Language. http://www.ontoknowledge.org/oil/.
32. OKBC. Open Knowledge Base Connectivity. http://www.ai.sri.com/ okbc/.
33. OntoBroker project. http://ontobroker.aifb.uni-karlsruhe.de/.
34. RDF(S): XML-based Resource Description Framework Schema Specification. http://www.w3.org/TR/WD-rdf-schema/.
35. RETSINA. http://www.cs.cmu.edu/ softagents/retsina.html.
36. A. Sheth, V. Kashyap, and T. Lima. Semantic information brokering: How can a multi-agent approach help? In M. Klusch, O. Shehory, and G. Weiss, editors, *CIA-1999 Workshop on Cooperative Information Agents*, volume 1652 of *LNAI*. Springer, 1999.
37. A. Sheth and J.A. Larson. Federated database systems. *ACM Computing Surveys*, 22(3), 1990.
38. A. Sheth, E. Mena, A. Illaramendi, and V. Kashyap. Observer: An approach for query processing in global information systems based on interoperation across pre-existing ontologies. In *Cooperative Information Systems CoopIS-96*. IEEE Computer Society Press, 1996.
39. SPKI Simple PublicKey Infrastructure. ftp://ftp.ietf.org/internet-drafts/draft-ietf-spki-cert-theory-02.txt.
40. K. Sycara, M. Klusch, S. Widoff, and J. Lu. Larks: Dynamic matchmaking among heterogeneous software agents in cyberspace. *Autonomous Agents and Multiagent Systems*, March 2001.
41. K. Sycara, J. Lu, M. Klusch, and S. Widoff. Dynamic service matchmaking among agents in open information environments. *ACM SIGMOD Record*, 1999.
42. K. Sycara and D. Zeng. Coordination of multiple intelligent software agents. *Cooperative Information Systems*, 5(2/3), 1996.
43. G. Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer Systems*, 25(3), March 1992.

44. H.C. Wong and K. Sycara. A taxonomy of middle-agents for the internet. In *Agents-1999 Conference on Autonomous Agents*, 1999.
45. H.C. Wong and K. Sycara. Adding security and trust to multi-agent systems. In *Autonomous Agents 1999 Workshop on Deception, Fraud, and Trust in Agent Societies*, May 1999.
46. XOL. Ontology Exchange Language. http://www.ai.sri.com/pkarp/xol/.
47. B. Yu and M.P. Singh. A social mechanism of reputation management in electronic communities. In M. Klusch and L. Kerschberg, editors, *CIA-2000 Workshop on Cooperative Information Agents*, volume 1860 of *LNAI*. Springer, 2000.