Brokering Services for Cooperative Distributed Systems: An Agent Privacy-Based Architecture

(Spine Title: Brokering Services for Cooperative Distributed Systems)

(Thesis Format: Monograph)

By

AbdulMutalib M. Masaud Wahaishi

Graduate Program
In
Engineering Science
Department of Electrical and Computer Engineering

A Thesis Submitted in Partial Fulfillment
of the Requirements for the Degree of
Doctor of Philosophy

Faculty of Graduate Studies

The University of Western Ontario

London, Ontario

October, 2007

THE UNIVERSITY OF WESTERN ONTARIO
FACULTY OF GRADUATE STUDIES

CERTIFICATE OF EXAMINATION

The thesis by

AbdulMutalib M. Masaud Wahaishi

entitled:

Brokering Services for Cooperative Distributed Systems: An Agent Privacy-Based Architecture

is accepted in partial fulfilment of the
requirements for the degree of
Doctor of Philosophy

ABSTRACT

Cooperative distributed systems (CDS) approach is a promising design paradigm that is suitable for many applications such as healthcare, virtual enterprises, e-business and tele-learning in which entities have some degree of authority to sharing their capabilities. Brokering is a capability-based coordination approach for CDS. A major challenge of brokering in open environments is to support privacy. Within the context of brokering, we model privacy in terms of the entities' ability to hide or reveal information related to its identities, requests, and/or capabilities. In this work we present in-depth analysis of the capability-based coordination and propose a privacy-based brokering framework and interaction protocols that support different privacy degrees. Unlike traditional approaches, the brokering is viewed as a set of services in which the brokering role is further classified into several sub-roles each with a specific architecture and interaction protocol that is appropriate to support a required privacy degree. A formal specification of the privacy-based brokering protocols is represented using an Input/Output Automata model. To put the formulation in practice, a prototype of the proposed architecture has been implemented to support information-gathering capabilities in healthcare environments.

Keywords:

Cooperative Distributed Systems, Capability, Brokering, Interdependency, Coordination, Privacy, Agent Technology, Input-Output Automata.

بسم الله الرحمن الرحيم

قَالُواْ سُبْحَانَكَ لاَ عِلْمَ لَنَا إِلاَّ مَا عَلَّمْتَنَا إِنَّكَ أَنتَ الْعَلِيمُ الْحَكِيمُ

In the name of God, Most Gracious, Most merciful

"They said: Be glorified! We have no knowledge saving that which Thou hast taught us. Lo! Thou, only Thou, art the Knower, the Wise"

(The Noble Quran 2:32)

"اني رأيت أنه لايكتب إنسان كتاباً في يومه إلا قال في غده :لو غُيِّر هذا لكان أحسن, ولو زيد كذا لكان يُستحسن. ولو قُدِّم هذا لكان أفضل, ولو تُرك هذا لكان أجمل. وهذا من أعظم العبر, وهو دليل على استيلاء النقص على جملة البشر "

(العمادالاصفهاني)

"I have found that, whenever one commits his thoughts on a paper or a book, doubt invariably sets in. One thinks: If I were to make an addition, it would become clearer; if I were to remove such-and-such a part, it would become more elegant; if I were to rearrange these sections, it would become prettier. Therein lies a profound lesson; for it is an indication of the imperfection that permeates the actions of all human beings".

(Al-Emad Al-Asfahani, Islamic philosopher)

DEDICATION

To the memory of my late father, the prominent Sheikh "Alhaj Mohamed Masaud-Wahaishi", may Allah the almighty be pleased with him.

ACKNOWLEDGMENT

First and foremost, all praise is due to Allah, the Beneficent, the Merciful. He has guided me; given me the potential, and without his blessings I would not be able to make this degree possible.

In the course of my PhD, I had the great fortune of being supervised by a very supportive and helpful supervisor. His integrity, dedication, insightful advice and encouragement have greatly contributed in fulfilling this degree, my heartfelt gratitude to Dr. Hamada Ghenniwa.

I would also like to thank my examiners, Dr. Sabah Mohammed, Dr. Roberto Solis-Oba, Dr. Abdulla Shami and Dr. Weiming Shen for all their constructive comments and suggestions which surely helped in improving the thesis; members of the CDS-Eng group (especially Raafat and Toufik) at the Department of Computer and Electrical Engineering for their constant efforts and willingness to help.

I remain indebted and grateful to my parents for their prayers, both of whom have always believed in me; my brothers and sisters and my friends back home in Libya for their continuous support.

Finally, but certainly not least, I would like to express my sincere gratitude to my beloved wife Fadwa for all the sacrifices, the patience, and most importantly, for just being with me. It could not be possible without you, for which I will be always indebted. My thanks also to my children Mohamed, Fatma, Shaden, Shayma and Zuhair, who are my pride and joy, for their unconditional love and for understanding that sometimes “dad is extremely busy”.

TABLE OF CONTENTS

List of Tables

List of Figures

Chapter 1

INTRODUCTION

An important class of distributed systems is *Cooperative Distributed Systems* (CDS)**,** in which entities are able to exercise some degree of authority in sharing their capabilities. This characteristic is very desirable in designing systems for many applications, such as electronic business, enterprise integration, manufacturing engineering and virtual environments. In such environments, an application is usually constituted of geographically distributed and decentralized entities. Entities in this paradigm are expected to collaborate and work together to achieve their goals. To enable successful collaboration, the need of coordination and cooperation approaches is an essential necessity. However, a major challenge of coordination in open environments is to enable cooperation under a desired level of privacy protection. This chapter provides an overview of cooperative distributed systems and introduces issues addressed in this dissertation with emphasis on *capability-based coordination* as brokering services.

1.1. Cooperative Distributed Systems - CDS

A distributed system consists of a collection of different entities (such as processes, components, databases, knowledge-base, and so on.) that can perform some functions independently. An important class of these systems is Cooperative Distributed Systems (CDS), in which the entities are able to exercise some degree of authority in sharing their capabilities.

Due to the distributed nature of applications based on the CDS approach, an entity may not be available or known when needed. Although these systems are independently created and administered, they usually need to work together to accomplish individual or social tasks. Nevertheless, in open environments, this becomes a challenge where it is no longer feasible to expect designers or users to hardcode, to determine or to keep track of the entities and their capabilities.

In order to provide agile coordination solutions for the growing complexity of contemporary CDS to share information and capabilities, new coordination approaches and architectures need to be explored and developed.

1.2. Capability-Based Coordination: A Motivation

In developing CDS in open environments, coordination is a major challenge. Entities need to locate and interact with others who posses the capabilities to achieve a particular goal. For distributed systems, fulfilling a request may go beyond the capability of the individual entities, this is known as the *capability-interdependency problem* [32].

In the conventional point-to-point interaction configuration, entities interact directly with each other to provide controlled and directed coordination. However, this configuration is both inflexible and computationally expensive. For instance, there is no separation of concerns between computation and coordination. The absence of a separate medium that deals exclusively with the coordination aspects in the system means that the entities, in addition to other computational activities, have to carry out the "interaction work" themselves to satisfy common or local tasks.

As an alternative, the capability-based coordination approach can be a very effective medium for interaction and coordination. In this approach, the entities need not to be concerned with how the interaction is performed or done. The essential objectives of capability-based coordination solutions are to facilitate the interaction of various entities who continue to operate in open distributed environment and compete to deliver value-rich services.

Brokering is a capability-based coordination which is viewed as an abstraction level at which a distributed system environment can be viewed collectively as a coherent universe. Furthermore, such coordination gives a new dimension of communicating where the involved entities are not required to be known to each other, nor exist in the same place at the same time in order to communicate which relieve the them of the burden of having to handle the coordination concerns, thus providing them with more space and time for other computational activities to improve their profitability and gain a competitive advantages.

Within this context we define brokering for CDS as: "*A capability-based coordination service that provides coordination solutions to a variety of participants in open distributed environments*".

In capability-based coordination, participants can be distinguished by the role they play (for example, a service requester or a provider). Providers specify services they provide in capabilities. For example, a service that provides weather forecasting is an example of a capability. Capabilities are often accompanied by services parameters, which specify the conditions under which services are offered such as cost and quality. Requesters specify services they need in requests. Requests can be accompanied by preferences, which are counterparts of service parameters.

The capacity to coordinate the entities' behavior coupled with the possibility to control different levels of privacy upon the operations they perform is vital. The privacy concerns over the inappropriate use of the CDS resources such as information and services make it hard to successfully take advantage of the gains from sharing, utilizing or accessing the capabilities of these systems. Entities prefer to have authority on controlling the collection, retention and the distribution of information about themselves in such environments. This restricts the willingness of CDS entities to share their capabilities and consequently, distributed systems architects, developers and administrators are faced with the challenge of securing the desired privacy levels.

Thus motivated, the scope of this research focuses on incorporating capability-based coordination solutions, emphasizing on developing a framework that defines an appropriate coordination structure and mechanisms that imply various requirements and protocols for interoperability and interaction to suit desired levels of privacy.

In order to develop a privacy-based brokering framework, we provide thorough analysis and define the structure that represents the patterns of communication amongst involved participants. Then we propose the interaction protocols and the mechanisms of the coordinated control to support various levels of privacy.

To provide deep understanding and formal treatment of the coordination activities, we propose a formal model to describe and represent each privacy-based interaction protocol. Furthermore, these protocols are analyzed and consequently we provide a detailed design and implementation guidelines for a privacy-based brokering model.

Different approaches have been proposed to facilitate capability-based coordination in CDS to deal with relevant privacy issues. However; to our knowledge, none of these approaches have treated privacy as an architectural property of the CDS.

1.3. Privacy in Cooperative Distributed Systems

With the rapidly growing development of applications, user's privacy is becoming a critical issue. As a result, distributed systems architects, developers and administrators are faced with the challenge of securing the user's privacy as well as the services he or she might access. In general, users and service providers are concerned about their personal privacy from different perspectives. For example, they may wish to protect their identities from being used, or decide by whom they will be revealed, and for what purposes, or retain the choice about whether or not to reveal their personal interests or capabilities.

There are numerous privacy risks that create many threats to personal privacy and raise unique privacy concerns in developing CDS, for example identity theft is now becoming an industry in its own right, with massive acquisition of personal data sufficient to do serious damage on a large scale. The dissemination of sensitive information is particularly offensive as a violation of privacy, and dealing with it can be enormously time consuming. Target marketing can turn into spamming, service customization can turn into unfair price discrimination, hackers and insiders can cause systemic denial of access to targeted individuals, and so on. For these and other reasons, many people provide false identity. On the other hand, providers desire to share capabilities and provide services in a manner that does not violate their privacy. In many settings, service providers need to be guarded and prevented from damages resulted form malicious entities and accordingly need to effectively manage and prevent any abuse of the services they provide.

With growing concern about privacy in distributed environments, considerable research has been conducted focusing on various aspects. Solutions and models put forth by this research address specific challenges of the problem. However, within the context of brokering for CDS we view Privacy as "*the ability of CDS entities to decide upon revealing or hiding information related to their identities, requests and capabilities in open distributed environments*".

1.4. Contribution

A major contribution of this work is to define a generic architecture of the brokering interaction protocols that are appropriate to different privacy degrees. It demonstrates how brokering can be used to provide capability-based coordination solutions, particularly to different levels of privacy protection. The work introduces several new original ideas that contribute to the overall thesis. They are:

- **Brokering Architecture,** that enables cooperation under a desired level of privacy protection in CDS.
- **Defining Brokering,** as a capability-based coordination solution for the interdependency problem.
- **Defining Privacy,** within the context of brokering and is viewed as an architectural element of the coordination.
- **Interaction Protocols**, that provides the structure and the suitable mechanisms for capability-based coordination.
- **Formal Structure**, for capturing, describing and representing the privacy-based interaction protocols.
- **Design and Implementation**, guidelines that provide a prominent tool for developers, architects and system designers to develop privacy based CDS applications.

Chapters 3, 4, and 5 will further elaborate on these concepts.

1.5. Organization of the Thesis

This chapter has provided an overview of the entire thesis, which is structured as follows:

Chapter 1: Literature Review. This chapter provides a fundamental literature review to the most relevant works; the review is based on exploring the different approaches that have been proposed in the field to deal with brokering and privacy. This chapter reviews the literature relevant to the thesis. It begins with a general overview of coordination in distributed systems, and then focuses more closely on the two areas directly related to the thesis: capability-based coordination approaches and privacy.

Chapter 3: Privacy-Based Brokering Framework. This chapter represents our view of modeling the cooperative distributed systems and represents the privacy-based brokering framework. The chapter also defines formal specifications and representation for the proposed privacy-based protocols.

Chapter 4: Design and Implementation. Introduces an agent privacy-based brokering architecture and illustrates the detailed design and implementation guidelines of the proposed model.

Chapter 5: Summary and Conclusion. The final chapter of the thesis discusses the conclusions about the research described throughout the dissertation, and recapitulates the contributions, limitation and presents proposals and directions for future work.

Chapter 2

LITERATURE REVIEW

This chapter provides a literature review of integration solutions in cooperative distributed systems with special focus on capability-based coordination approaches that deal with privacy concerns in open environments. In this chapter we survey some of the existing approaches that deal with the coordination solutions and the protection of the privacy attributes of CDS entities.

2.1. Capability-Based Coordination Techniques and Approaches

The coordination activities within and across distributed systems can be performed by a different coordination structures and mechanisms that imply various requirements and protocols for interoperability and interaction. The structure refers to the patterns of communication amongst involved participants (for example, brokering, matchmaking or facilitation). The mechanisms define the coordinated control and the interaction protocols. The mediator-based approaches [96] provide dynamic solutions for coordination in which the focus is on developing software modules that perform value-added activities but keep the information model hidden. Examples include MOMIS [7] and OBSERVER [59]. The information entity (mediator) provides integrated access to various types of heterogeneous information sources and controls the coordination according to the following pattern: (i) the mediator receives, collects and forwards user requests; (ii) locate and direct user's request to the appropriate information provides results to the receiver's request, and (iii) collects query results and delivers an integrated response to the user [33].

2.1.1. Broker-based Approaches

In the broker-based approaches, all the communication between paired participants has to go through an intermediary entity (usually called, a broker) [26][43]. The broker controls all the coordination activities between requesters and providers. The broker interaction pattern involves contacting providers who might have the capability to serve a particular request. The interaction might involve indifferent mechanisms to facilitate the coordination such as negotiation, contract nets and auction protocols. The Broker insulates requesters and provider and thus protects the provider from hostile or unruly requesters. Examples of broker-based systems include MACRON [44] and NZDIS [76]. In MACRON, the architecture does not preclude providers' privacy attributes from being exposed to other agents in the environment. On the other hand, requesters are required to reveal their privacy attributes to their relevant agent (called query manger, QM) when submitting their queries.

The NZDIS (New Zealand distributed information system), introduces a capability-based coordination approach for integrating distributed information systems. The architecture comprises the same pattern of the broker-based approach. Recent approaches distinguish a resource brokering architecture that manages the scheduling of different tasks on a large-scale grid [21][11][47]. The proposed architectures utilize the broker-based approach in which a brokering entity allocates requesters' tasks to different distributed resources. The Grid Service Broker [93] focuses on developing matchmaking solutions for computational and data-grid applications to provide interoperability and accessibility means to distributed resources from different grids. Other approach emphasizes on locating capabilities by obtaining some form of quality-of-service (QoS) offers from different resources, so that offers from providers may be distinguished based on the level of the provided quality [104].

Other frameworks utilize the broker-based architecture [30][60] [39][45][64] to deal with capability-based coordination for Web services [95]. These approaches provide protocols that coordinate the actions of distributed applications and enable existing transaction processing, workflow, and other systems for coordination to hide their proprietary protocols and to operate in a heterogeneous environment. Examples of these approaches are, IRS-II [63], IRS-III [51] and Agent-based semantic Web Services [65][67] [72].

2.1.2. Matchmaker-based Approaches

In the matchmaker-based approaches, an intermediary entity coordinates the activities by identifying the relevant provider(s) for the requester. The interaction pattern involves matching given requests with appropriate capabilities offered by available providers. The matchmaker identifies and proposes a set of potential providers to the requester. The requester contacts the proposed providers and accordingly carries out any further interaction. [89][55]. In contrast with the broker-based approaches, all interactions are undertaken between the requester and the provider directly. Examples of matchmaker-based systems include InfoSleuth [97], RETSINA [88], DECAF [36], IMPACT [5] and COINS [49].

2.1.3. Facilitator-based Approaches

The facilitator-based [15] approaches extend the functionality of the mediator architecture with automatic resource identification and data conversion. This level of automation requires all attribute information to be presented and revealed to the facilitator. The Infomaster [29] is an information integration system that utilizes this approach.

2.1.4. Distributed Databases

Approaches for interoperability across information systems were proposed in the context of database management systems. Global and federated approaches have been proposed for achieving coordination between distributed databases based on the overall system at the architecture level. In the former, all the local schemata may be integrated into a single global schema that represents all the databases in the entire distributed system [84]. Some of the information integration approaches include TSIMMIS [13] and TAMBIS [73]. Both of these systems are domain-specific (static data bioinformatics) and query-centric federated systems based on three layers of mediators and wrappers. Both systems assume that information about the sources is available a priori.

2.1.5. Standards, Platforms and Specifications

The FIPA agent software integration specification defines how software resources can be described, shared and dynamically controlled in an agent community [25]. It includes agent wrappers for software services and an agent resource broker (ARB) service. Wrappers are agents that interface non agent-based software with agent-based systems. An Agent Resource Broker (ARB) allows advertisement in the agent domain and management of their use by other agents, such as the negotiation of parameters (e.g. cost and priority), authentication and permission.

Within the industry arena, there are a number of standards and platforms, which have gained a wide acceptance. Their main purpose is to provide infrastructure tools, platforms and frameworks, to facilitate capability-based coordination at the technology level by allowing different entities to easily communicate with each other to exploit different capabilities.

Some of the well-known standards and platforms include the Object Management Group's CORBA (Common Object Requester Broker Architecture) [69] and JINI [87].

CORBA allows distributed objects to communicate with each other. The interface of the distributed objects is described in a special language, the Interface Definition Language (IDL). Each object that needs a service must access the service using the object request broker (ORB). Locating services (objects) is accomplished through two CORBA services: naming service, in which objects are discovered based on their names and trader services, in which objects are discovered based on their capabilities. Objects need to register their presence (object references and capabilities) with ORB.

JINI is a Java-based platform for service discovery. The main components of a JINI system are *Services*, *Clients* and *Lookup Services*. A service registers a "service proxy" with the Lookup Service and clients requesting services get a handle to the "service proxy" from the Lookup Service.

Several available commercial products enable coordination at the at the technology level. The concern is to provide tools to provide a uniform means for distributed application to interact and use the technological capabilities to produce a desired functionality, some of these products include: Mercator Enterprise Broker [91], NEON eBusiness Integration Servers [66], SeeBeyond EBusiness Integration Suite [82], Actional Control Broker [1] and CrossWorlds [19]. These industrial approaches are built primarily on messaging middleware technology that provides capability-based coordination based on pre-built application adapters, and bi-directional connectivity to multiple applications. The middleware receives requests and identifies the target sources.

2.1.6. The Service-Oriented Semantic Driven Architecture (SOSDA)

The Service-Oriented Semantic Driven Architecture (SOSDA) has proposed integration architecture for CDS [31]. The SOSDA specifications provide the abstraction to support the domain entities and applications independent of any specific technology.

Within this architecture CDS is viewed as a service-oriented environment in which the overall connectivity of the system supports a "*virtual*" point-to-point integration mechanism as shown in Figure 1. SOSDA defines various services and the primary ways in which they interact to support integration.

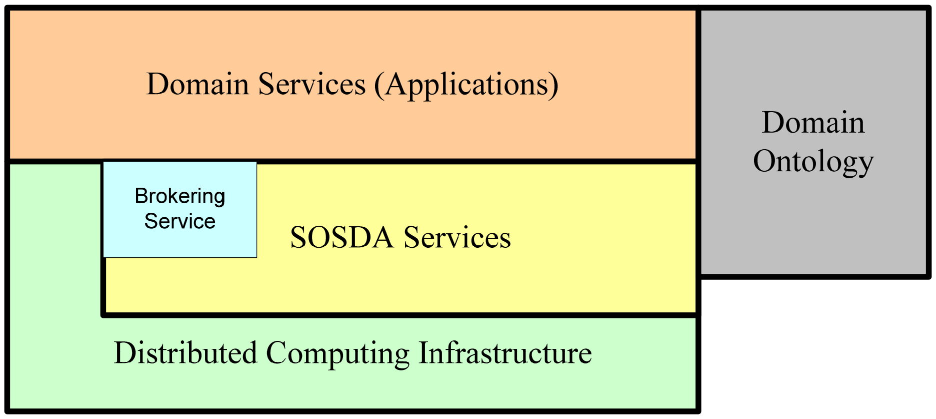
Domain Services (Applications)
Domain Ontology
Brokering Service
SOSDA Services
Distributed Computing Infrastructure

Figure 1: SOSDA as a Layered Architecture

Basically, services in SOSDA are classified into three family-of-services (FOS): (1) Coordination & Cooperation, (2) Ontology & Semantic Integration and (3) Wrapping services. The work presented in this dissertation deals with the brokering as part of the coordination and cooperation FOS which provides ad hoc or automated support for capability-based coordination.

In summary, the objective of the approaches described above was based on extending the capability of a specific class of distributed systems by developing auxiliary tools that facilitate coordination as an aspect of cooperation.

This work addresses the capability-dependency problem and proposes a capability-based coordination framework that provides coordinated access to a collection of different domain entities in open environments. Form the coordination point of view the work focuses on the following issues:

1. Interaction Protocols (IPs)
    - ◙ It is assumed that the environment has the dynamic nature where entities have to accomplish their tasks utilizing an appropriate interaction mechanism.
2. Formulation and Representation
    - ◙ The description of the privacy-based protocols should adhere to a formal representation. A set of specifications for realizing that model need to be presented. The specifications provide prominent foundation for developers, architects and system designers to develop applications that provide capability-based coordination.

2.2. Privacy

Privacy appears as a major challenge is providing coordination solutions in open distributed environments. Recognition and understanding of the privacy problems in CDS and the risks that result from inadequate action are absolutely essential. Tremendous effort has been devoted to deal with privacy and security issues in distributed systems for the last few decades to find technological means of guaranteeing privacy by employing state-of-the art encryption and anonymization technologies [50]. Although, these technologies can provide tools to secure a great deal of protection, personal privacy entails more than just a secret communication and masked identity.

Using privacy as an organizing paradigm, one approach [20] suggested nine roles that can be played by middle agents. These roles are categorized by the preferences and capabilities information that can be kept initially with the requester and provider agents, respectively, and can be later revealed to the middle agent. Other agent-based architectures involving three parties have been suggested, introducing a layer between users and providers, constituted by brokers, mediators or middle agents, an overview is given in [99]. An agent-based management of user profiles, including access control mechanisms, has also been suggested [98], the approach proposes a combination of (XML-based) access control and privacy technologies such as Platform for Privacy Preferences (P3P) to control access to distributed managed user profiles.

P3P [70] is an industry standard that aims to enable web sites to express their privacy policies in a standardized format that they can be automatically retrieved and interpreted. It provides entities with the ability to communicate about privacy preferences in a standard machine-readable format. The specification describe includes a protocol for requesting and transmitting P3P policies which is built on the same HTTP protocol that web browsers use to communicate with web servers. Requesters use standard HTTP requests to fetch a P3P policy reference file from a well-known location on the web site to which a user is making a request. The policy reference file indicates the location of the P3P policy file that applies to each part of the web site. There might be one policy for the entire site, or several policies that each covers a different part of the site. The requesters can then fetch the appropriate policy, parse it, and take action according to the user's preferences. However P3P addresses only a narrow set of privacy issues related to the automation of creating, requesting and reading privacy policies.

In the context of Web services, the Privacy Service [75] defines the requirements that enable privacy protection for the consumer of a Web service across multiple domains and services. However, these requirements focus on the details of data encryption at the data level.

The work in [46] introduced the concept of privacy engineering and accordingly proposed architecture to manage personal data held in Digital Rights Management (DRM) systems [22]. The architecture defines a language to represent and describe DRM rights in terms of permissions, constraints and obligations between users and contents. A repository (Rights metadata) defines the control over data contents. A data controller entity deals with access requests to determine and interprets access rights. The approach emphasizes privacy as a data protection aspect and combines encryption techniques with access mechanisms with implicit trust on the controller entity which manages the coordination activity.

Different approaches to protect the location privacy in open distributed systems [8]. Location privacy is a particular type of information privacy that can be defined as “the ability to prevent other parties from learning one’s current or past location”. These approaches range form anonymity, pseudonymity and cryptographic techniques. Some approaches focus on using anonymity by unlinking user personal information from their identity. One available tool is called Anonymizer [4]. The service protects the Internet protocol (IP) address or the identity of the user who views WebPages or submits information (including personal preferences) to a remote site. The solution uses anonymous proxies (gateways to the Internet) to route user’s Internet traffic through the tool. However, this technique requires a trusted third party, because the Anonymizer servers (or the user’s Internet service provider, ISP) can certainly identify the user. Other tools try not to rely on a trusted third-party to achieve complete anonymity of the user’s identity on the Internet, such as Crowds [77], Onion Routing [35] and MIX networks [14].

The Crowd approach is based on the idea of ‘blending into a crowd”, i.e., hiding one's actions within the actions of many others. The interaction pattern starts when a user joins a “crowd” of other users. Before sending a user’s request to a web server, it has to be first passed to a random member of the crowd who can choose either to submit the request directly to the end server or forward it to another randomly chosen member within the crowd. When the request is eventually submitted, it is submitted by a random member, thus preventing the end server from identifying its true initiator. Although, the user’s identity can be prevented, the coordination model does not address any privacy concerns that might be needed by the end server (service provider).

The goal of Onion Routing (OR) is to protect the privacy of the sender and recipient of a message, while also providing protection for message content as it traverses a network. Similar to the crowd approach, the onion routing encompass the following: messages travel from source to destination via a sequence of proxies ("onion routers"), which reroute messages in an unpredictable path. To prevent an adversary from eavesdropping on message content, messages are encrypted between routers. It is not necessary to trust each cooperating router; if one or more routers are compromised, anonymous communication can still be achieved. This is because each router in an OR network accepts messages, re-encrypts them, and transmits them to another onion router. However, it is possible for a local eavesdropper to observe that an individual has sent or received a message and therefore, onion routing does not provide absolute guarantee of privacy; rather, it provides a continuum in which the degree of privacy is affected by the number of participating routers versus the number of compromised or malicious routers.

A mix network provides anonymous communication facilities. A mix-network can be viewed as a public key cryptographic approach that takes as input a number of ciphertexts, decrypts and shuffles them and finally outputs a random permutation of plaintexts. Each message is encrypted to each proxy using public key cryptography; the resulting encryption is layered with the message as the innermost layer. Each proxy server strips off its own layer of encryption to reveal where to send the message next. If all but one of the proxy servers is compromised by the tracer, privacy protection can still be achieved. However, its weakness lies in its vulnerability of communication between the user device and the service provider and it is possible for an eavesdropper to observe that a proxy has sent or received a message.

Other approaches [100] focus on preserving the user's anonymity by applying cryptographic techniques that focus on encrypting the user's personal information using Private Information Retrieval (PIR) schemas [16]. The emphasis is to allow users to retrieve information from database sources while keeping their queries private from providers. Many of the PIR schemes were proposed under the assumption of accessing a single database source. Some applications or users require services without providing or using any user identifiable information.

Information Space organizes information, resources, and services around privacy-relevant contextual factors [102]. The model defines different boundaries and permissions for information, resources, services, and authorizations management in context--aware systems. For privacy control, an information space boundary acts as a trusted entity to enforce permissions defined by owners of that space. This model protects sensitive information through control of the information through trusted owners under the assumption of trustworthiness of the metadata as well as the software component that processes the metadata.

Other schemas support multiple database sources but assume that these multiple sources would not communicate with each other, which is not realistic in practical applications. Protocols in these schemas are based on the RSA decryption algorithms [78] that are geared only towards the protection of the user's identity. Nevertheless, these schemas focus on a single service environment which makes them impractical for a dynamic distributed environment where requesters (and service providers) need to maintain several keys (private and public) for identification purposes when requesting services from many providers which might lead to practical implementation complexities [94][6][58][54].

Other initiatives proposed the use of privacy policies along with physical access means (such as smartcards) in which the access to private information is granted through the presence of another trusted entity [18], the X.509 [101] and pretty good privacy (PGP) [3]. The X.509 standard defines and specifies the structure and format of digital certificates and credentials. In the X.509 system, an intermediary which is a single trusted network entity with whom other entities are registered, issues digital identity and attribute certificates. The standard accommodates adaptable levels of privacy for users' anonymity, however the need of a trusted third party to protect one's privacy is a necessity. Another approach [9] makes use of a policy language to check the compliance to the required level of privacy. However required privacy attributes (defined as a required level of privacy) need to be revealed to the compliance checker in order to verify credential holders.

Another approach [53] provides access control mechanisms and tools for protecting requesters' personal privacy. Service requesters joining an environment are prompted for the required privacy policies of each service in the environment. A dedicated requester's proxy checks these policies against the user's predefined privacy preferences and accordingly decides upon using or declining the services.

Privacy Enhancing Technologies (PET) [34] are based on eliminating or reducing personal data by preventing unnecessary and/or undesired processing (and storage) of personal and individual data, all without losing the functionality of the system. The term PET is used to describe all types of technologies that provide privacy to a user.

The Privacy Incorporate Software Agent (PISA) targets the creation of privacy enhancing technologies for electronic business applications [74]. The project utilizes PET as a technical solution to protecting the privacy of users when using intelligent agents in E-commerce applications, according to EC-Directives on Privacy. The PISA adopts agent technology for intelligent brokering and matching. However the focus of the project is to develop coordination architecture to achieve information privacy using cryptographic mechanisms.

Several solutions [48] were proposed to deal with different privacy challenges encountered in various application domains such as e-Auctions [68], data mining [56], e-commerce [94] and healthcare [83]. In the e-Auction approach, a privacy-based auction protocol assumes the existence of a trusted auctioneer which evaluates received bids non-interactively. The protocol ensures that no information beyond the result is disclosed, provided that the auctioneer does not collude with any participant. In the e-commerce approach, the privacy model adopts agent-based approach, in which a mobile agent authenticates electronic transactions on behalf of a customer on a remote host. The model does not address any privacy concerns related to remote hosts. These solutions assume the existence of a central trusted entity that has all the information about participants, or assume that each participant of the computation shares all relevant information with others.

In healthcare domain, one approach as described in [62], the focus was on providing management assistance to different teams across several hospitals by coordinating their access to distributed information. The brokering architecture is centralized around a mediator agent, which allocates the appropriate medical team to an available operating theatre in which the transplant operation may be performed. Other approaches attempts to provide agent-based medical appointments scheduling [2][61]. In these approaches the architecture provides matchmaking mechanisms for the selection of appropriate recipient candidates whenever organs become available through a matchmaking agent that accesses a domain specific ontology. Others proposed the use of privacy policies along with physical access means (such as smartcards), in which the access of private information is granted through the presence of another trusted authority that mediates between information requesters and information providers [103]. Web-service based tools were developed to enable patients to remotely schedule appointments, doctor visits and to access medical data [85][12]. With the advent of Information Technology and its obvious surveillance potential, various programs and initiatives have proposed a set of guidelines for secure and private collection, transmission and storage of patients' data. Some of these programs include: the Initiative for Privacy Standardization in Europe (IPSE) [40] and the Health Insurance Portability and Accountability Act (HIPAA) [38]. Yet, these guidelines need the adoption of new technology for healthcare requester /provider interaction.

Privacy Sensitive Information Diluting Mechanism (PSIUM) [14] is a model that tries to eliminate the misuse of requester information by service providers. The model proposes an interaction protocol that enables requesters to send multiple location-based service requests to the service provider in which only one of these requests contains the true location. On return of service's result, the requester links the available result without revealing the correct location and thus preventing the identity form being exposed to the service providers. However, the model increases the cost of linking results from the service provider as the number of queries is increased.

Many approaches identify various patterns to address many information security problems, including privacy. In [80] , the approach proposed criteria for privacy patterns and identified protection solutions for cookies and pseudonymous emails. The criteria describes how a user can configure their web client to control how and when cookies are set and used and provides guidelines for internet users to send and exchange emails without revealing their online identity.

Another approach [17] describes a privacy proxy that informs users of a website's privacy practices and introduces two types of patterns: patterns that shields and preclude personal information from being transmitted to other entities in the environment, and patterns that filter information sent from others to the user.

Building on existing privacy based pattern, another initiative [79] have proposed three privacy patterns for web-based activities that address system architecture issues related to informed consent for web-based transactions and the support for personal privacy protection. These patterns describe how users can protect their privacy by both revealing less about themselves, and acquiring more information from the party with whom they are communicating. In one approach, a privacy patterns that deals with information filtering in collaborative systems was presented [81]. In [37], the work presented four design patterns for building anonymity systems for online interactions. However, the aforementioned privacy patterns do not provide detailed specification of the corresponding protocols, the structure of the coordinated control and the type of messages exchanged in any interaction. Additionally, the patterns do not address the architectural requirements for building systems that enable capability-based coordination under various privacy degrees.

Furthermore, the patterns typically describe the rules for controlling the performance of the entities' actions. For example, patterns associated with privacy policies assume that after an entity accepts the policy, it agrees to enforce these rules when it performs actions and therefore the task of accepting privacy policies and interpreting the definitions rests on the shoulders of the involved entity.

Many of these solutions have the assumption that the computations take place with the existence of a completely trusted third party.

Additionally, none of the above-mentioned approaches have treated privacy as an architectural element within the coordination services. The objective of the work presented here is to develop a brokering architecture that deals with various degrees of privacy as related to the identity, requests and capabilities of the participant entities (requesters and providers) within a cooperative distributed system.

In summary, developing the brokering services comprise the automation of privacy to enhance the overall security of the system and accordingly entities should be able define the desired degree of privacy. The challenge in this context is how to architect the brokering with the appropriate set of services that enable cooperation across the different degrees of privacy. The focus is to provide a mechanism to reduce the costs and risks that might be a result of violating privacy requirements. This work addresses issues and challenges in providing privacy-based brokering that will include:

- Develop a privacy-based model
    - The model will allow building systems that work in an open environment with different roles and behaviors. This model has to take into consideration any degree of privacy that might be needed by entities within a CDS. Within the context of brokering, the degree of privacy is defined towards three privacy attributes; entities' identities, requests, and capabilities.

2.3. Summary

In open environments, where entities may appear and disappear unpredictably, the need of an effective coordination service is essential. Furthermore, in developing cooperative distributed systems, privacy is a desired aspect of providing capability-based coordination in these systems. This chapter highlighted different approaches to develop capability based coordination solutions for distributed systems with special focus on privacy.

Chapter 3

PRIVACY-BASED BROKERING FRAMEWORK

This chapter introduces a privacy-based brokering model for CDS. Here privacy is geared towards preserving the identity, requests and capabilities of the CDS entities. The model represents different brokering scenarios and introduces the interaction protocols associated with various privacy degrees that might be required by the CDS participants. For each privacy degree, an associated interaction protocol defines the basic components, the structure and the pattern representing both message communication and the corresponding constraints on the content of such messages. The chapter provides a formal description and a representation of the interaction protocol which expresses many fundamental and essential characteristics of the proposed privacy-based model.

3.1. Brokering for Cooperative Distributed Systems

In CDS, domain entities are usually required to collaborate and work together to satisfy a request. Moreover, these entities should be able to select an appropriate privacy level and play different roles to achieve their goals and get results to their requests regardless of whether the request can be satisfied at a local or remote location.

A domain entity's role can be categorized as either a service-requester or a service-provider. A service-provider is the role of a domain entity with the capability to meet the needs of another domain entity. A service-requester is the role of a domain entity that attempts to achieve a goal beyond its own capability.

Brokering entities need to interact (on behalf of requesters) with various providers to fulfill a request. The interaction protocols specify the set of allowed message types, message contents and the correct order of messages during any brokering scenario. To facilitate the interaction, entities in CDS need to depend heavily on communication with each other not only to perform requests, but also to advertise their capabilities.

3.2. The Privacy-Based Brokering Model - The Brokering Layer

Architecturally, the brokering model is viewed as a layer of services, each with a specific architecture and interaction protocols. The brokering layer enables entities to solicit help and delegate requests and get results according to interaction protocols that deal with different privacy degrees. The interaction protocols represent the communication sequences, the set of allowed message types, and the message contents during the capability-based coordination activities between various participants in CDS.

The concern of this thesis is to view privacy in terms of three attributes: the entity's identities (*Id)*, capabilities (*Cap)* and requests (*Req*). The brokering enables the entities to participate in the environment with different roles and hence be capable of automating their privacy concerns and select a particular privacy degree. An entity is able to choose whether to reveal or hide a particular privacy attribute. Each role is represented as a special brokering entity (each has a distinguished name) with a specific architecture and interaction protocol that is appropriate to a required privacy degree.

Responsibilities are separated and defined according to the roles played and the required privacy degree. Within the layer, two sets of brokering entities are available to service requesters and providers. The first set handles interactions with requesters according to the desired privacy degree that is appropriate to their preferences. The other set supports privacy degrees required by service providers. Figure 2 shows a logical view of the brokering layer. Each brokering scenario is accomplished by the combination of the requester role, brokering entity role and the provider role. Note that in the figure, a specific privacy attribute variable $\overline{x},\ \ x \in \{Id, Req, Cap\}$ represents that the corresponding privacy attribute is not revealed.

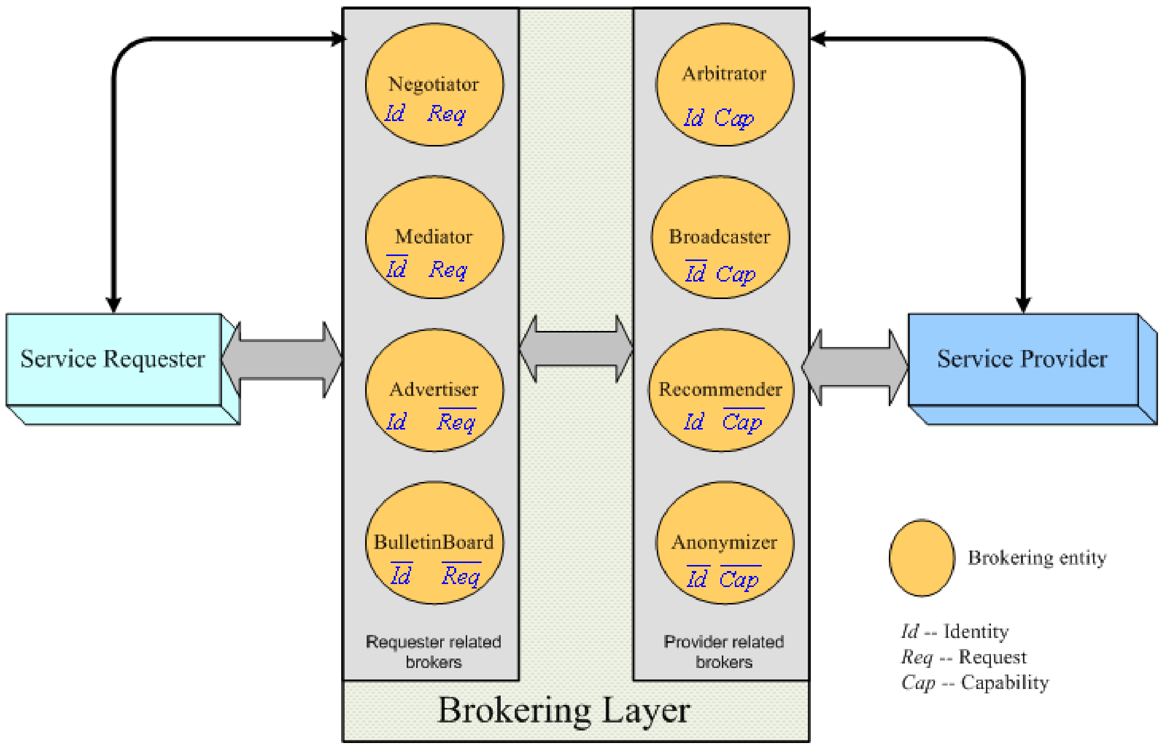

Figure 2: Logical View of the Brokering Layer

The following tables summarize the different scenarios that can be played by the brokering layer categorized by the required privacy degrees of both the requester and the provider entities.

Table 1: Brokering Roles and Interaction Protocols with Requesters

Brokering Role	**Privacy Attributes**		**Brokering Interaction**
	Req	*Id*	
Negotiator	Revealed	Revealed	• Receive service request • Forwards request to broker-provider side • Deliver result to requester
Mediator	Hidden	Revealed	• Retrieve service request posted by a requester • Forwards request to broker-provider side • Store result to be retrieved by requester
Advertiser	Revealed	Hidden	• Post service request to service repository • Requester to search repository and request service • Retrieve a service request that was stored by a requester • Forwards request to broker-provider side • Store result to be retrieved by requester
Bulletinboard	Hidden	Hidden	• Requester to store service request • Retrieve service request that was stored by a requester • Forwards request to broker-provider side • Store result to be retrieved by requester

Table 2: Brokering Roles and Interaction Protocols with Providers

Brokering Role	**Privacy Attributes**		**Brokering Interaction**
	Id	*Cap*	
Arbitrator	Revealed	Revealed	• Assign capable provider • Forwards request • Get service's result • Deliver result to requester-broker side
Broadcaster	Hidden	Revealed	• Post service request to service repository • Providers to access service repository • Providers to evaluate service parameters • Providers to store result • Provider-broker to retrieve stored result
Recommender	Revealed	Hidden	• Forward service request • Provider to evaluate request • Providers to store result • Provider-broker to retrieve stored result
Anonymizer	Hidden	Hidden	• Providers to access repository • Provider to evaluate request • Provider to store service result • Brokering layer to retrieve stored result

Each interaction protocol is described in terms of a combination of the interaction within the brokering layer and the interaction with the domain entities.

To provide a deep understanding and formal treatments of these protocols, we propose that each protocol is modeled using the Input/Output Automata (IOA) model [28][57]. This is further described in a precondition-action-postcondition mode. Each IOA depicts the entities' behavior and is then mapped and represented using Unified Modeling Language (UML) sequence diagrams.

3.3. The Input Output Automata (IOA)

Formally, an IO automaton is presented in terms of the action signature, the set of states and the set of transitions. The set of transitions are presented in a precondition/effect model. That is, the state during which an action is enabled is given as a precondition, and the resulting state is given by the effects of the action.

Each system component is modeled as an I/O automaton with an action labeling each transition. An automaton's actions are classified as either "input", "output" or "internal". An automaton generates output and internal actions autonomously, and transmits output instantaneously to its environment. In contrast, the automaton's input is generated by the environment and transmitted instantaneously to the automaton. An automaton (A) consists of the following components:

1. a set $states(A)$ of states
2. a nonempty set of start states $start(A) \subseteq states(A)$
3. a set $acts(A)$ of actions, and
4. a transition relation which is a set $steps(A) \subseteq states(A) \times acts(A) \times states(A)$ of steps.

The set of $acts(A)$ is portioned into three disjoint sets, $in(A), out(A), and\ int(A)$ which denote input actions, output actions and internal actions respectively. An action signature $sig(A) = (in(A), out(A), int(A))$ is a partition of actions into input actions, output actions and internal actions respectively. A state is to be said reachable if it is the final state in a finite execution of A.

The union of the input actions and the output actions represent the external actions which are visible to the environment. It is to be noted that an IO automaton (A) is a labeled state transition system which consists of a set of actions π (classified as input, output and internal), a set of states S (including a nonempty subset of start states), and a set of transitions in the form of $(s, \pi, s^{'})$ that specify the effects of the automaton's actions.

The following illustrates a simple popular example of candy machines using the IOA model:

Three candy machines CM-1, CM-2 and CM-3 differ only in their transition relations. The CM-1 candy machine has the following action signature:

Input actions:	PUSH1, PUSH2
Output actions:	SKYBAR, HEATHBAR, ALMONDJOY
Internal actions:	none

The state of CM-1 consists of one variable "*button-pushed*", which takes on values: 0, 1 and 2. In the initial state, "*button-pushed*", is set to 0. We describe the transition relation for CM-1 by giving a *precondition* and an *effect* for every action π the triple (s, π, s') is a step of CM-1 exactly if the precondition of π is satisfied by $S^{'}$ and S is the result of transforming $S^{'}$ as determined by the effects of π. We omit the precondition for an action when this precondition is true. The transition relation for CM-1 is as follows:

PUSH1

Effect:	button-pushed = 1

PUSH2

Effect:	button-pushed = 2

SKYBAR

Precondition:	button-pushed = 1
Effect:	button-pushed = 0

HEATHBAR

Precondition:	button-pushed = 2
Effect:	button-pushed = 0

ALMONDJOY

Precondition:	button-pushed = 2
Effect:	button-pushed = 0

When the customer pushes button, CM-1 can dispense a SKYBAR. When the customer pushes button, CM-2 can dispense either a HEATHBAR or an ALMONDJOY, but not both. The choice between H and A is made nondeterministically by CM-1. Candy machine CM-2 is identical to CM-1 except that its HEATHBAR action has “false”, as its precondition. This candy machine never dispenses HEATHBARs, but is able to dispense SKYBARs and ALMONDJOYs. Candy machine CM-3 is identical to CM-1 except that all three candy dispensation actions have “false” as their precondition. It never dispenses candy, which must disappoint a number of its customers.

Three customers CUST-1, CUST-2 and CUST-3 are also quite similar. CUST-1 continues to request candy bars repeatedly, non-deterministically choosing which button to push. Its action signature is the “complement” of the candy machines:

Input actions:	SKYBAR, HEATHBAR, ALMONDJOY
Output actions:	PUSH1, PUSH2
Internal actions:	none

The state of CUST-1 consists of one variable "waiting" which takes on values "yes" and "no". In the initial state, waiting" is set to "no", CUST-1s actions are as follows:

SKYBAR

	Effect:	waiting = no

HEATHBAR

	Effect:	waiting = no

ALMONDJOY

	Effect:	waiting = no

PUSH1

	Precondition:	waiting = no
	Effect:	waiting = yes

PUSH2

	Precondition:	waiting = no
	Effect:	waiting = yes

This customer is very patient, after pushing a button; it waits for a candy bar before pushing a button a second time. The partition *part*(CUST-1) of this customer's locally-controlled actions puts PUSH1 and PUSH2 together in one equivalence class. Customer CUST-2 is somewhat more selective than CUST-1. It pushes button 2 repeatedly just until the machine dispenses a HEATHBAR, and then pushes button 1 forever. Formally, CUST-2 has another variable "*heathbar-received*" in its state in addition to "waiting". This variable takes on values "yes" and "no", initially "no". The actions of CUST-2 that differ from those of CUST-1 are as follows:

HEATHBAR

	Effect:	waiting = no, heathbar-received = yes

PUSH1

	Precondition:	waiting = no, heathbar-received = yes
	Effect:	waiting = yes

PUSH2

	Precondition:	waiting = no, heathbar-received = no
	Effect:	waiting = yes

Customer CUST-3 is similar to CUST-1 except that it may make a transition to a “satiated” state from which it no longer requests any candy bars. Formally, CUST-3s state has an additional “satiated”, variable besides the “waiting” variable of CUST-1. It takes on values “yes” or “no”, initially “no”. CUST-3 has an additional internal action BECOME-SATIATED, denoted as follows:

BECOME-SATIATED

Precondition: satiated = no; waiting = no,

Effect: satiated = yes

Also, each of PUSH1 and PUSH2 has the additional precondition “satiated = no”. Again, *part*(CUST-3) puts all three locally-controlled actions PUSH1, PUSH2 and BECOME SATIATED in the same equivalence class.

UML (Unified Modeling Language) Sequence diagrams enable defining the actions and states of any IOA abstractly. Message exchanges between the entities provide concrete representation for actions’ abstractions. Table 3 represents a mapping for the IOA to the UML sequence diagram elements. The states of the object are represented by the lifeline. IOA transitions can be modeled as precondition-effect messages.

Table 3: Mapping IOA Parameters to UML Sequence Diagrams

IOA Parameters	**Sequence Diagram**	**Comments**
Entity	Object	
Input Actions	Input Messages	Messages received by the entity.
Output Actions	Return Messages	
Internal Actions	Self Messages	An internal invocation or execution of a particular operation.
States ● Start (special state) ● Final (special state)	Lifeline ● Object creation ● object Deletion	Object creation and deletion are special states of the object
Transition	Activation	Activation represents the change of state of an object when performing an operation.
Pre-condition	Guard Condition	
Post (Effect) Condition	State of the object	

A precondition is a predicate on the object state and the parameters of a transition that must hold whenever that transition executes. An effect on the object specifies the result of a transition. One or many transitions may be enabled at any time. However, only one is executed at a time. The selection of which enabled action to execute is a source of implicit non-determinism.

We can model an automaton (A) using object-oriented approach such as UML to include the tuple that consists of the following elements:

$$SD(A) \equiv \langle lifeLine, M, Act \rangle \text{, where}$$

1. $lifeLine$ represents the different states that an entity can be in, this includes the instantiation of an entity (creation $instance$), a state (s) and the final state (fs) of a particular object (might be an idle or a deletion of the instance), in such a way $instance \subseteq lifeLine$ and $fs \subseteq lifeLine$.
2. A set of messages M denotes the input messages $inMessage$, return message $returnMessage$ and a self message $selMessage$. The entire set of messages is denoted as: $inMessage \cup returnMessage \cup selMessage$.
3. An activation relation Act which represents the time during which an entity is performing an operation, where $Act \subseteq lifeLine \times M \times LifeLine$; this means that for every state in the $lifeLine$ and message $m \in M$, the object transit from state (s) to state (s') such as $(s, m, s') \in lifeLine$.

To formally describe the interaction protocols using the IOA, we model the brokering entity in a given brokering role as a unique automaton that generates output and internal actions autonomously, and transmits output instantaneously to its environment.

For example, a brokering entity that supports a requester hiding privacy attribute transmits the output action to the other elements of the environment such as domain entities (requesters and Providers) and other brokering entities. In contrast, the automaton's input is generated by the environment and transmitted instantaneously to the automaton.

The brokering entities perform actions triggered by its input which transition (A) into a valid state and produce some output. The input and output actions are defined in terms of message types the brokering and domain can receive and send, respectively.

In the specifications, we use the following variables and parameters:

- $reqID$, $provID$, $reqBrokID$ and $provBrokID$ are drawn respectively from the identity of a requester, provider, ReqBroker and ProvBroker.
- $request$, is a tuple $\langle reqID, serName, reqPerf \rangle$.
- $serviceRequest$, is a tuple $\langle reqBrokID, serName, reqPerf \rangle$.
- $serviceProposal$, is a $\langle provID, serName, serPar \rangle$, representing service(s) offered by the provider, where $serPar$ is a tuple consisting of the following $\langle input, output, cost, quality, time \rangle$
    - o $serName$, is the service name.
    - o $input$, represents the type of input needed for the service execution.
    - o $output$, represents the type of output that result after the service execution
- $serviceOffer$, is a tuple $\langle provBrokeID, serviceProposal \rangle$.
- $requestStorage$, has a value True when a service request is available, and False otherwise.
- $service$, has a value True indicates an available service offer.
- $delegated$, is a Boolean variable with values in $\{True, False\}$. True indicates that a service request has been delegated, initially is set to False.
- $resStatus$, has a value True when a service's request result is available.
- $serResult$, represents the result of a specific service request.
- $advertised$, has a value True when a service's offering is sent to a requester.
- $accepted$, is set to True when a service proposal is accepted.
- $requestStorageLocation$, is an indexed storage that holds service requests.
- $serviceOfferingStorage$, is an indexed storage that holds service offers.
- $delegatedRequests$, is an indexed storage that holds delegated requests to ProvBrokers.
- $acceptedServiceOffers$, is an indexed storage that holds accepted service proposals (from providers).
- $resultLocation$, is an indexed storage that holds results of services' requests (stored by service providers or by the ProvBrokers)

3.4. Brokering Interaction Protocols

Each brokering scenario encapsulates a set of conversation and message exchanges amongst the *requester-related brokering entities* (called ReqBroker henceforth) and the *provider-related brokering entities* (called ProvBrokers henceforth) as well as the corresponding domain entity which plays a specific role in an interaction protocol. An interaction protocol can be viewed as a set of messages' content and the constraints imposed on the individual roles in different privacy degrees. A role focuses on how the entity in a given state receives a message of a specified type, performs local actions, sends out messages, and switches to another state.

3.4.1. The Requester-Brokering Interaction Patterns

A requester interacts with the environment through sending and receiving messages. In some scenarios (for example, requesters hiding privacy attributes), the ReqBrokers and the domain entities exhibit a proactive behavior to respond to changes in the environment. The following represent the various roles and the associated interaction patterns that can be played by the brokering in supporting requesters with different privacy degrees.

The interaction requires a set of agreed messages, rules for actions based upon reception of various messages and assumptions of the communication channels. These constraints, rules and patterns can be abstracted and formalized as interaction patterns, which are basis for successful capability-based coordination. The interaction protocols range from negotiation schemas to a simple request for a task.

The interaction protocols are viewed as patterns representing both message communication and the corresponding constraints on the content of such messages. They describe an allowed sequence of messages and message content among participant entities. In the proposed model, a protocol is modeled as a set of communicating processes executing concurrently. They express the constraints on the relationship between sending and receiving messages which represent the protocol mechanism. This model emphasizes the entities' collaborative behaviors.

In order to define the messages that are needed to support a specific privacy degree, we first identify the required “message-types” that can satisfy the supporting protocol and next, decide on the possible messages that can be assigned to particular role in a given interaction protocols. Note that messages can be accompanied by guard conditions to describe the constraints on the exchanged messages. To summarize the process, the process will be as follows:

1. Define the possible roles that entities can play is a specific protocol
2. Identify how many types of messages exist in an interaction protocol. Message types are specified as constructors of the actions initiated by the entities.
3. Decide what messages a role can send, check, receive or store
4. Next, we have to figure out the rules and constraints on these messages.

A message consists of a sender, a set of receivers, “type” of message and message “content”. In all the following interaction protocols, we focus only on message semantics, without caring about its implementation details. For readability purposes, we list the interaction protocols using the message type only.

3.4.1.1. The Negotiator

Consider the following scenarios:

- A doctor wants to have information about the number of patients who have Hepatitis B in a specific city. The doctor needs to be assessed without exposing its identity and the pertinent request to others.
- A customer who wants to prevent marketers and service providers from generating user profiles of his/her shopping trends, financial and travel interests while requesting particular services.

The above scenarios exemplifies privacy degrees in which revealing sensitive information can lead to catastrophic discrimination outcomes, knowing the scientist’s identity might lead to a biased and unfair decision; marketing trends can turn into spamming. Therefore, it might be desirable to not be identified when accessing on-line services. Requester should be able to interact with the corresponding brokering entity to request services, receive service's results, and acknowledge the receipt of service's result.

The proposed protocol protects the requester's identity and requests despite revealing them to the Negotiator. The assumption is that the Negotiator is a trusted entity. Figure 3 depicts the protocol that involves the Negotiator's interaction pattern includes interaction with various ProvBrokers. The Negotiator forwards the request to all the ProvBrokers. The Negotiator issues a Call-For-Proposals (CFP) to ProvBrokers (act as potential contractors) with the request specifications which include:

- *Request abstraction:* a brief description of the request represented by the service name that abstract the required capability.
- *Request specification:* a description and the expected format of the request.
- *Expiration time:* a statement of the time interval during which the announcement is valid.

For example, within the healthcare domain, a doctor might request health information related to the mortality rate amongst the newborns in specific region. Accordingly, a request for service is defined as follows: $\langle informationGethering, NewBorn - Mortality, Region - Name, PDF, 30 \rangle$ .The request states that an electronic PDF file is required for newborn mortality data in a defined region within a defined time unit.

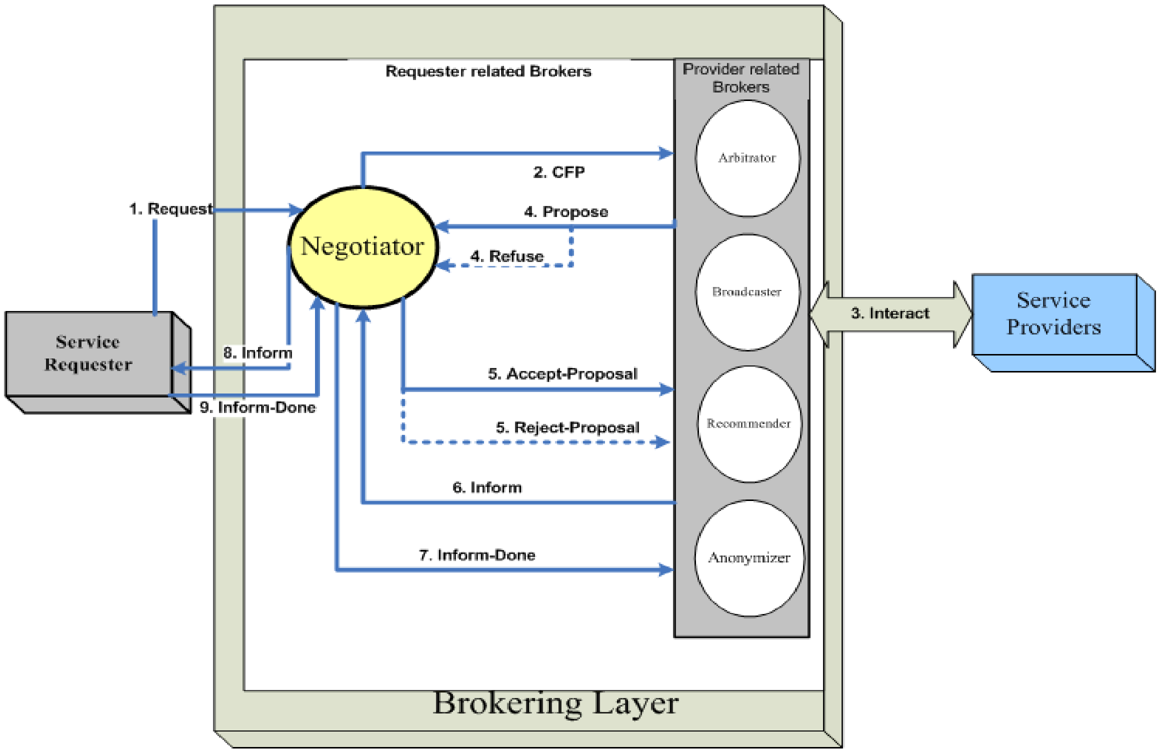

Figure 3: Interaction Pattern for the Negotiator

Each ProvBroker submits an offer on behalf of its providers. The interaction protocol represents the message communication and the corresponding content of such messages.

1. Receive ("Request") – A service request is received by the Negotiator.

8. Send ("Inform") – The Negotiator delivers service's result to the requester.

9. Receive ("Inform-Done") – A message is received from the requester indicating the receipt of the service's result.

The interaction within the brokering layer is represented as follows:

2. Send ("CFP") – Sending a call for proposal message to ProvBrokers.

4. Receive ("Propose") – The Negotiator receives service proposal(s).

4. Receive ("Refuse") – A ProvBroker declines to participate in fulfilling a service request.

5. Send ("Accept-Proposal") – A message is sent to the wining ProvBroker indicating the acceptance of the proposal.

5. Send (“Reject-Proposal”) – A rejection message is sent to those ProvBrokers who do not win.

6. Receive (“Inform”) – The Negotiator receives the service’s result[1].

7. Send ("Inform-Done”) – The Negotiator informs the ProvBroker of the receipt of the service’s result.

The Negotiator Automaton

The action signatures of the Negotiator include the following subsets of actions.

Input Actions: All input actions are referred by the “*receive*” action to represent that the environment is the source of the action.

- *receive(Request(request))* -- A request for service received from a requester.
- *receive(Inform(provBrokID, resStatus))* -- Result of a service request is received from the ProvBroker.
- *receive(Infom - Done(reqID, resStatus))* -- A message received from the requester indicating the receipt of a service’s result.
- *receive(Propose(serviceOffer))* -- A service offering is received from the ProvBroker.
- *receive(Refuse(serviceOffer))* -- A decline message from the ProvBroker pertinent to a particular service request.

Output Actions: All output actions are referred by the “*send*” action to represent that the Negotiator is the source of the action.

- *send(CFP(serviceRequest))* -- A CFP message is sent to the ProvBrokers.
- *send(Accept - Proposal(serviceOffer))* -- An acceptance message is sent to the ProvBroker.
- *send(Reject - Proposal(serviceOffer))* -- A rejection message is sent to the ProvBroker.
- *send(Inform(reqID, serResult))* -- sending a service’s result to the requester
- *send(Infom - Done(provBrokID, resStatus))* -- A message is sent to the ProvBroker indicating the receipt of a service’s result

Internal Actions

The internal actions of the Negotiator are mainly generated to perform operations related to accessing different storage repositories (store and delete). The Negotiator has an

[1] A result is the required format depicted in the service’s request, (for example, the patient’s diagnosis information in PDF as per the patient’s service request specification).

additional internal action pertinent to the evaluation of service offerings received from various ProvBrokers.

- *store(Request, RequestStorage)* .
- *store(serResult, resultLocation, resStatus)*
- *store(serOffer, serviceOfferStorage)*
- *remove(Request, RequestStorage)*
- *remove(serResult, resultLocation, resStatus)*
- *remove(serOffer, serviceOfferStorage)*
- *evaluateOffer(serOffer, serviceRequest)*

States: We capture the set of states as variable labels with instantiation values.

- *Wait* : is the initial state of the ReqBroker which represent two possible values, $Wait := True$ and $Wait := False$.
- *RequestAccessed* : represents the state in which service the ReqBroker has received a service request that need to be served. It represents two possible values $RequestAccessed := True$ and $RequestAccessed := False$.
- *Delegation* : denotes a delegation of specific service request to a particular ProvBroker. It can have two possible values, $Delegation := True$ and $Delegation := False$
- *ResultAccessed* : represent results acknowledgments of service requests that need to be sent to the requester or to be stored in the service. It represents two possible values $ResultAccessed := True$ and $ResultAccessed := False$.
- *cfpInitiation* : represents the state where the ReqBroker has issued CFP message to various ProvBrokers. It represents two possible values $cfpInitiation := True$ and $cfpInitiation := False$.
- *Evalution* : represents the state where the ReqBroker has received service offers from ProvBrokers. It represents two possible values $Evalution := True$ and $Evalution := False$.

The set of Transitions for are represented as action/precondition/postcondition model

```
receive(Request(request))
    precondition :
        Wait := True
    Effect :
        RequestAccess := True
        requestStorage := True
      Wait := False

send(CFP(serviceRequest))
    precondition :
        RequestAccess := True
    Effect :
        cfpInitiation := True
```

```
recieve(Inform(provBrokID, serResult))
    precondition :
        Delegation := True
        resStatus := False
    Effect :
        ReSult := True
        resStatus := True
        Delegation := False

recieve(Infom - Done(reqID, resStatus))
    precondition :
        Result := True
        resStatus := False
    Effect :
        ReSult := True
        resStatus := True
        Result := False
        requestStorage := False
        Wait := True

recieve(Propose(servcieOffer))
    precondition :
        cfpInitiation := True
        delegated := False
    Effect :
        Evaluation := True
        cfpInitiation := False

recieve(Refuse(serviceRequest))
    precondition :
        cfpInitiation := True
    Effect :
        cfpInitaition := True

store(Request, RequestStorage)

    Effect :
        requestStorage := True

store(serOffer, serviceOfferStorage)

    Effect :
        servcie := True

remove(serResult, resultLocation, resStatus)

    Effect :
        resStatus := False
```

```
send(Accept - Proposal(serviceOffer))
    precondition :
        Evaluation := True
        delegated := False
    Effect :
        Delegation := True
        Evaluation := False

send(Infom - Done(reqID, resStatus))
    precondition :
        Result := True
        delegated := True
    Effect :
        ReSult := True
        delegated := False

send(Reject - Proposal(serviceOffer))
    Precondition :
        Evaluation := True
        delegated := False
    Effect :
        Evaluation := True
        delegated := False

send(Inform(reqID, serResult))
    precondition :
        Result := True
        resStatus := Truse
    Effect :
        ReSult := True
        resStatus := True

store(serResult, resultLocation, resStatus)

    Effect :
        resStatus := True

remove(Request, RequestStorage)

    Effect :
        requestStorage := False

remove(serOffer, serviceOfferStorage)

    Effect :
        servcie := False
```

```
$evaluateOffer(serOffer, serviceRequest)$
    $precondition :$
        $Evaluation := True$
        $accepted := false$
    $Effect :$

$1 - if\ serOffer\ is\ accepted\ , then\ accepted := True$
$2 - if\ serOffer\ is\ rejected\ , then\ accepted := False$
```

Figure 4 shows the state-machine representation[2] of the IOA for the Negotiator.

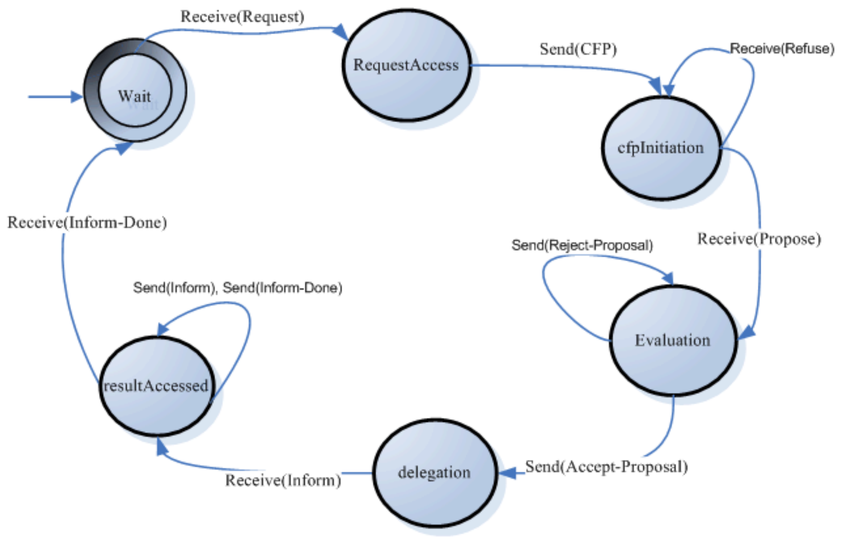

Figure 4: State Transition Diagram representing Negotiator Behavior

The Requester Automaton

In addition to its behaviour, the requester automaton has the following additional variables and is described as follows:

- $waiting$, is a Boolean variable with values in $\{True, False\}$. True indicates that the requester is waiting for the service's result.
- $resultRecieved$, is a Boolean variable with values in $\{True, False\}$ .True indicates the receipt of the result.

The input and output actions and the associated transitions are as follows:

[2] Note that, for readability reasons, the diagram shows only the transitions when the states are True.

```
recieve(Infom(reqID, resStatus))
    precondition :
        waiting := True
        resultRecieved := True
    Eff :
        resultRecieved := False
        waiting := True
```

```
send(Request(request))
    precondition :
        waiting := False
    Eff :
        resultReceived := False
        waiting := True

send(Infom - Done(reqID, resStatus))
    precondition :
        resultRecieved := True

    Eff :
        waiting := False
```

Figure 5 depicts the interaction as exchange of messages to accomplish the desired behavior. A requester interacts with the ReqBroker regarding a service request. The request is stored in a repository through invoking the *add* method.

In brief, available contractors (represented by the various ProvBrokers) evaluate announcements sent by the ReqBroker and submit service offers. The Negotiator stores received offers into the serviceOfferStorage repository by invoking the *add* method. The ReqBroker internally evaluates the bids (invoking the *evaluate* method) and awards contracts to the ProvBroker it determines to be the most appropriate and accordingly invokes the *add* method to store the delegated service requests into the delagtedServiceRequests repository.

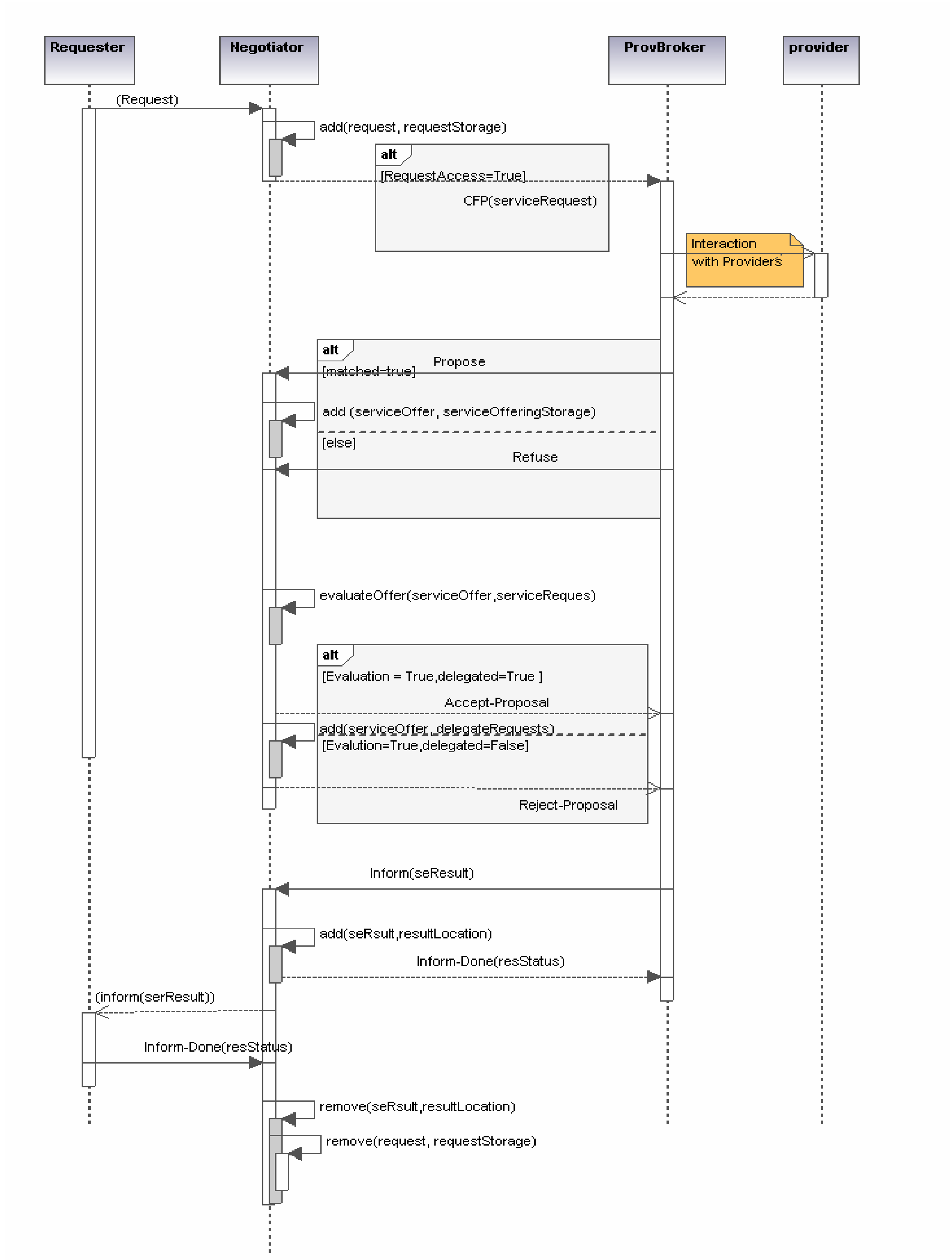
Requester
Negotiator
ProvBroker
provider
(Request)
add(request, requestStorage)
alt
[RequestAccess=True]
CFP(serviceRequest)
Interaction with Providers
alt
[matched=true]
Propose
add (serviceOffer, serviceOfferingStorage)
[else]
Refuse
evaluateOffer(serviceOffer,serviceReques)
alt
[Evaluation = True,delegated=True]
Accept-Proposal
add(serviceOffer, delegateRequests)
[Evalution=True,delegated=False]
Reject-Proposal
Inform(seResult)
add(seRsult,resultLocation)
Inform-Done(resStatus)
(inform(serResult))
Inform-Done(resStatus)
remove(seRsult,resultLocation)
remove(request, requestStorage)

Figure 5: Sequence Diagram for the Interaction Protocol of the Negotiator

In the meantime, the Negotiator sends an output message to those who do not satisfy the desired service request and accordingly delete relevant service offers that have been previously stored (invoking *delete* method on the serviceOfferStorage). Upon fulfilling the service request, the Negotiator receives the result and stores it in a resultLocation repository. Note that the behavior of the Negotiator has to ensure and confirm the receipt of the service's result by the requester prior to the deletion of the stored result or the service's request. In other words, the confirmation signals the fulfillment of the Requester service's request and consequently exemplifies the end of the interaction protocol

The sequence diagram illustrates a distributed control since processing and communication are not focused on a particular entity, but rather every entity is capable of accepting and assigning service requests.

Entities involved in exchanging such messages are assumed to have a prior knowledge of the available operations and methods that fulfill the required functionality. The behavior is controlled wholly by the actions initiated by the external entities. Therefore, objects involved in supporting this privacy degree exhibit a predefined flow of control and could not autonomously have explicit control over initiating messages. In an open environment, more complex forms of message exchanges need to be introduced to facilitate the transferring of information among these objects for which a high-level of abstraction is required to model such interaction.

3.4.1.2. The Mediator

In some cases, such as in healthcare environments, patients with fatal diseases may wish to request services and seek further health related information without the need to reveal their identities.

The requester should have appropriate means that permit requesting services without exposing its identity. Clearly a direct communication link with the Mediator violates this requirement. Therefore, a requester must convey requests and get results exclusive of related identity information. This can be achieved by providing an access to common storage facilities that are publicly available to post requests and retrieve results. The storage facility can be a dedicated repository, or a database. The Mediator is responsible for granting requesters the right to access these facilities either for a limited number of times or only for a limited-time period (for example during the active involvement of the requester in the interaction protocol).

The requester should have a prior explicit consent to access these storage facilities either to post service's request or to retrieve a result (for example, protection guidelines for Sexual Transmission Disease, STD). Retrieving results implies the ability of requesters to link a particular request to its corresponding result. This can be accomplished by assigning a unique identification key for every posted request. Both the Mediator and the requester use this key during the interaction protocol to identify and link the service request to its relevant result.

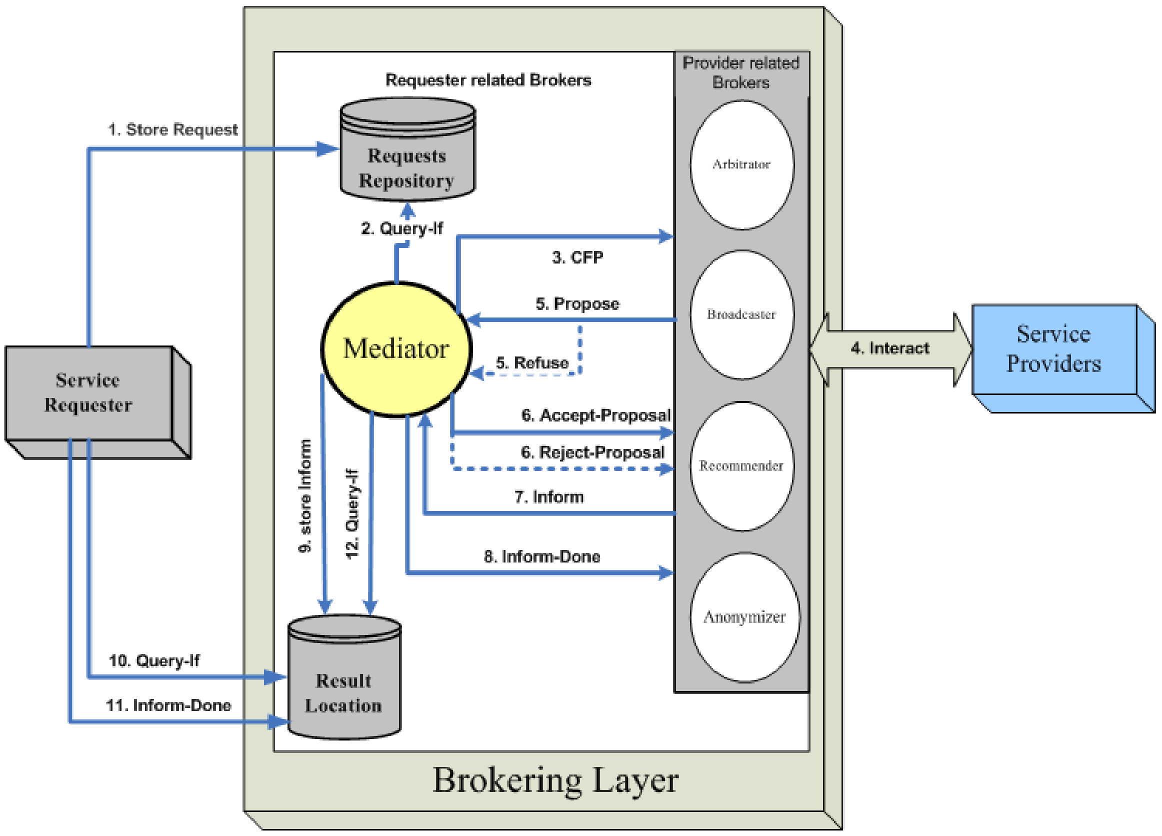

Figure 6: The Interaction Pattern for the Mediator

It is to be noted that, in order to be authorized for online access to such repositories, the requester might be at the risk of exposing its IP (internet protocol) address and hence the privacy requirement will be violated. To overcome such an issue, requesters will be able to hide their IP through the use of a proxy server utilizing cryptographic techniques in which a dynamic IP address is issued from a pool of IP addresses and therefore making the identity anonymous.

The Mediator checks for available requests that have been posted and accordingly forwards the service request to the ProvBrokers. Figure 6 shows the proposed interaction pattern associated with this privacy degree[3]. The corresponding protocol will be as follows:

2. The Mediator checks for ("**Request**") message for any available service requests that were stored by Requesters and need to be served.

3. Send ("CFP") – Sending a call for proposal message to ProvBrokers.

5. Receive ("Propose") – The Mediator receives service proposal(s).

5. Receive ("Refuse") – A ProvBroker declines to participate in fulfilling a service request.

6. Send ("Accept-Proposal") – A message is sent to the wining ProvBroker indicating the acceptance of the proposal.

6. Send ("Reject-Proposal") – A rejection message is sent to those ProvBrokers who do not win.

7. Receive ("Inform") – The Mediator receives the service's result.

8. Send ("Inform-Done") – the Mediator informs the ProvBroker of the receipt of the service's result.

9. The Mediator to store **("Inform")** indicating the availability of a service's result.

12. The Mediator checks for **("Inform-Done")** that has been stored by the Requester (indicating the receipt of the result).

The requester interaction with the relevant Mediator is solely restricted to the following:

1. The requester to store ("**Request**"**)** into a request repository.

10. The Requester checks for ("**Inform**"), indicating the availability of the result and hence retrieves it.

---

[3] Note that in all the interaction diagrams, the Query-If action permits the brokering entity to access the storage repositories and to check stored messages for message types.

12. Upon retrieving the service result, the requester stores **("Inform-Done")** into the result repository.

The Mediator Automaton

The input and output actions for the Mediator are similar to the actions generated during the interaction with the ProvBrokers. Note that a requester hides its identity by setting the value of $reqID$ in $\langle reqID, serName, reqPerf \rangle$ of the stored $request$ to null. As shown in Figure 7, the Mediator exhibits a behavior represented by the internal actions and the transitions described as follows:

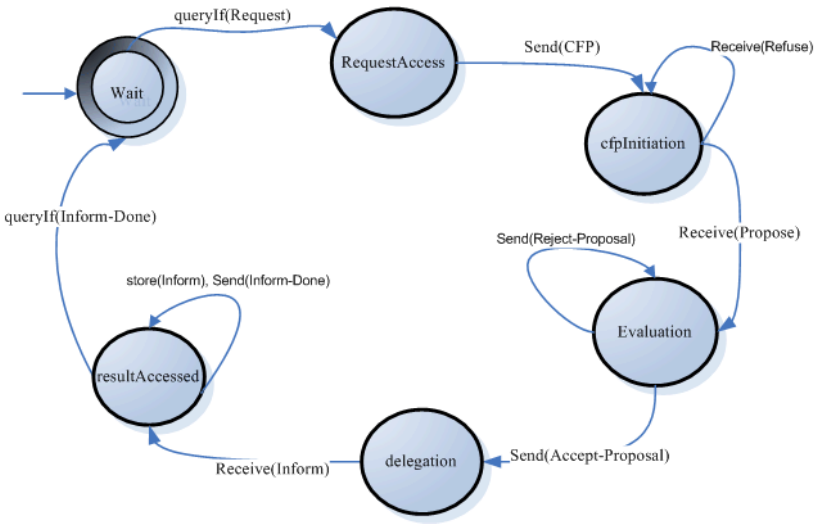

Figure 7: State Transition Diagram representing the Mediator Behavior

```
store(Inform(serResult))
    precondition :
        $resStatus := True$
        $ResultAccessed := True$
    Eff :
        $ResultAccessed := True$

store(Request, RequestStorage)

    Effect :
        $requestStorage := True$

queryIf(Request(request))
    precondition :
        $Wait := True$
    Eff :
        $requestStorage := True$
        $RequestAccessed := True$
        $Wait := False$

store(serOffer, serviceOfferStorage)

    Effect :
        $servcie := True$
```

queryIf(Inform – Done())
precondition :
ResultAccessed := True
Eff :
cfpInitiation := True
requestStorage := False
Wait := True

remove(serResult, resultLocation, resStatus)

Effect :
resStatus := False

store(serResult, resultLocation, resStatus)

Effect :
resStatus := True

remove(Request, RequestStorage)

Effect :
requestStorage := False

remove(serOffer, serviceOfferStorage)

Effect :
servcie := False

evaluateOffer(serOffer, serviceRequest)
precondition :
Evaluation := True
accepted := false
Effect :

1 – if serOffer is accepted , then accepted := True
2 – if serOffer is rejected , then accepted := False

The Requester Automaton

A requester does not exhibit any observable behaviour (i.e. no external actions are generated). The behaviour is only restricted to the internal actions that enable the requester to access the storage repositories. These actions and the associated transitions are described as follows:

store(Inform – Done(resStatus))
precondition :
resultReceived := True
Effect :
waiting := False

queryIf(Inform(serResult))
precondition :
waiting := True
resultReceived := True
Effect :
resultReceived := False

store(Request(request))
precondition :
waiting := False
Effect :
resultReceived := False
waiting := True

The pattern of the exchanged messages shown in Figure 8 indicates a similar sequence as described in the previous case. The only difference lies in the interaction among the requester and the Mediator. The requester stores a request along with desired preferences into a requestStorage repository by invoking the store method. The Mediator checks the desired request, retrieves it and thus initiates the same sequence of the mentioned contract net protocol [86]. In contrast to the protocol described in the previous case, the Mediator stores the service's result which will be retrieved by the requester. Upon retrieving the result, the requester confirms the receipt by invoking the store method on the requestStorage. The protocol comes to an end once the Mediator verifies such confirmation. The confirmation allows the Mediator to remove any relevant information related to this request.

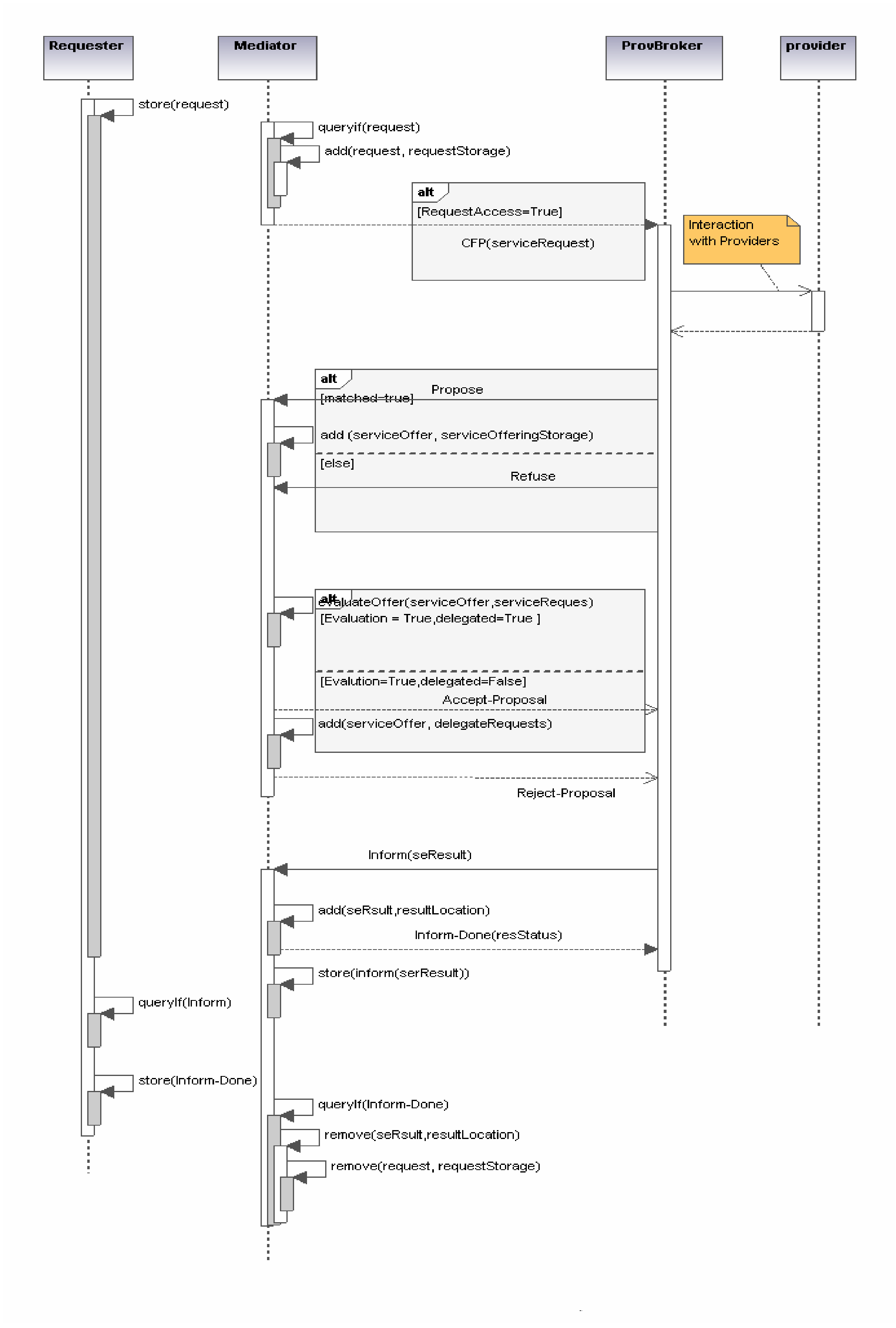
Requester
Mediator
ProvBroker
provider
store(request)
queryif(request)
add(request, requestStorage)
alt
[RequestAccess=True]
CFP(serviceRequest)
Interaction with Providers
alt
Propose
[matched=true]
add (serviceOffer, serviceOfferingStorage)
[else]
Refuse
alt
evaluateOffer(serviceOffer,serviceReques)
[Evaluation = True,delegated=True]
[Evalution=True,delegated=False]
Accept-Proposal
add(serviceOffer, delegateRequests)
Reject-Proposal
Inform(seResult)
add(seRsult,resultLocation)
Inform-Done(resStatus)
store(inform(serResult))
queryIf(Inform)
store(Inform-Done)
queryIf(Inform-Done)
remove(seRsult,resultLocation)
remove(request, requestStorage)

Figure 8: Sequence Diagram for the Interaction Protocol of the Mediator

Requesters hiding their identities need not to worry about or observe this behavior in fulfilling their requests. The sequence diagram represents a predefined flow of control that is completely determined by the current state and the actions executed by the objects (deterministic environment). Requesters need not to react on specific method invocations only, but rather on observable events within the environment as well. Requesters need to poll the environment for events and other messages (available service offerings) to determine what action they should take.

3.4.1.3. The Advertiser

There might be certain situations where requesters prefer to hide their requests. For example, clinicians might benefit form variety of service offerings regarding new medications, tools, medical equipments and health related notifications. The clinicians will be able to check a service's repository for service offerings that have been previously posted and thus decide on choosing an offering that might be of interest.

In order for those clinicians to browse such a repository, an access control should be granted prior to any interaction. The access to this repository provides an appropriate indirect communication channel that allows service requesters to post requests and get results without having to reveal their request to the relevant ReqBroker supporting this privacy degree.

The Advertiser permits requesters to check a service's repository for further information or to search for other service offerings that have been previously posted and accordingly determines services that might be of interest.

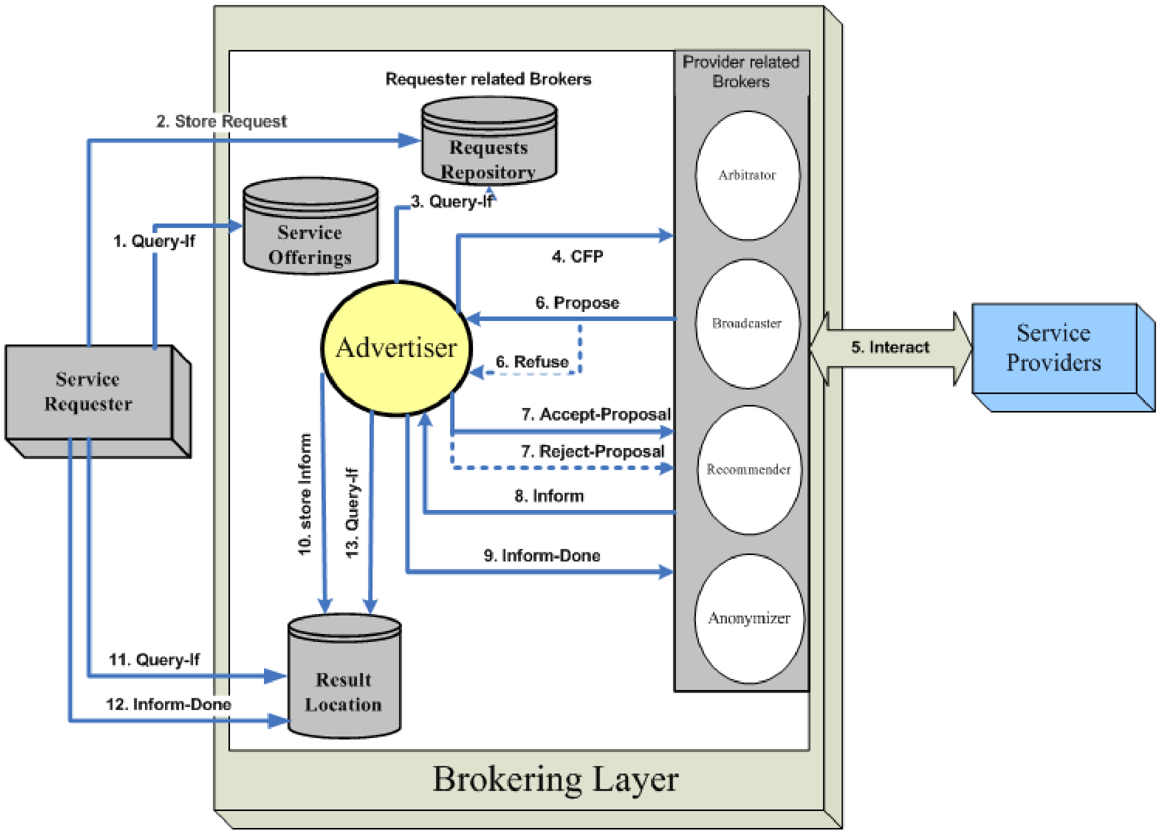

Figure 9: The Interaction Pattern for the Advertiser

Upon selecting a particular offering, the requester informs the Advertiser with the desired a service request as shown in Figure 9. Similarly, the interaction pattern is as follows:

1. The requester to check for (“**Propose”)** for service offerings.

2. The requester to store (“**Request**”**)** into a request repository.

3. The Advertiser to check for (“**Request**”) which indicates the availability of service requests.

4. Send (“CFP”) – Sending a call for proposal message to ProvBrokers.

6. Receive (“Propose”) – The Advertiser receives service proposal(s).

6. Receive (“Refuse”) – A ProvBroker declines to participate in fulfilling a service request.

7. Send (“Accept-Proposal”) – A message is sent to the wining ProvBroker indicating the acceptance of the proposal.

7. Send (“Reject-Proposal”) – A rejection message is sent to those ProvBrokers who do not win.

8. Receive (“Inform”) – The Advertiser receives the service’s result.

9. Send ("Inform-Done”) – The Advertiser informs the ProvBroker of the receipt of the service’s result.

10. The Advertiser to store **(“Inform”)** indicating the availability of a service’s result.

11. The requester checks for (“**Inform**”) for the availability of the result and hence retrieves it.

12. Upon retrieving the service result, the requester stores **(“Inform-Done”)** into the result repository.

13. The Advertiser checks for **(“Inform-Done”)** that has been stored by the requester (indicating the receipt of the result).

The Advertiser Automaton

As shown in Figure 10 and in addition to the input and output actions generated during the interaction with the ProvBrokers, the Advertiser has an additional output action that is related to proposing service offerings to the requester. The Advertiser actions and the transitions are described as follows:

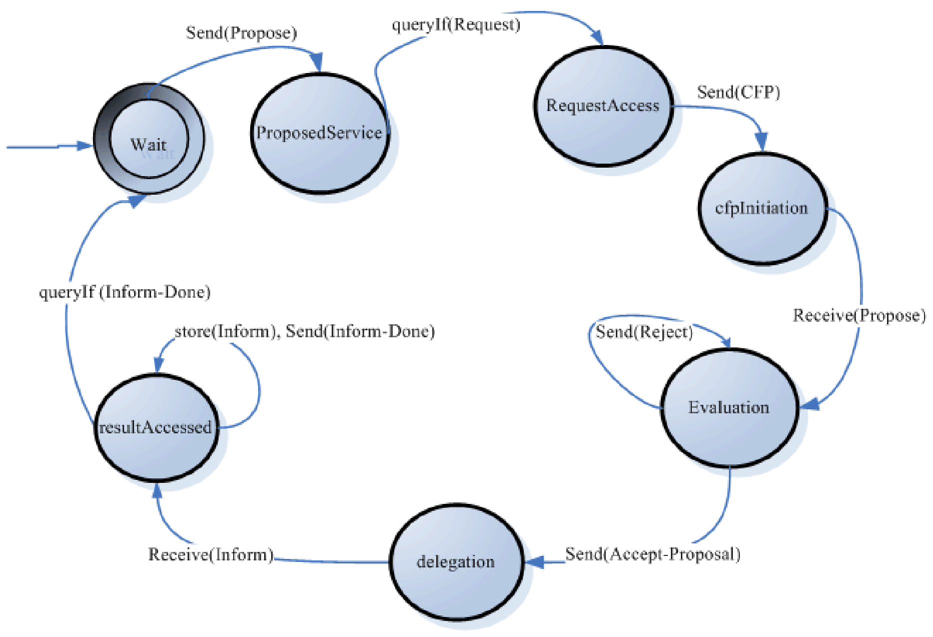

Figure 10: State Transition Diagram representing the Advertiser Behavior

```
send(Propose(reqID, serviceOffer))
    precondition :
        Wait := True
    Effect :
        ProposedService := True
        advertised := True

store(Inform(serResult))
    precondition :
        ResultAccessed := True
        resStatus := True
    Effect :
        ResultAccessed := True

queryIf(Request(request))
    precondition :
        ProposedService := True
    Effect :
        ProposedService := False
        RequestAccess := True
        requestStorage := True

queryIf(Inform – Done())
    precondition :
        RequestAccessed := True
    Effect :
        RequestAccessed := False
        requestStorage := False
        Wait := True
```

The Requester Automaton

Although the requester in this scenario is revealing its identity, it is required to set the value of $reqID$ in $\langle reqID, serName, reqPerf \rangle$ of the stored $request$ to null to prevent further linking of the identity to the request. The requester actions and the associated transitions are described as follows:

```
store(Inform – Done(resStatus))
    precondition :
        resultReceived := True
    Effect :
        waiting := False
```

```
queryIf(Inform(serResult))
    precondition :
        waiting := True
        resultReceived := True
    Effect :
        resultReceived := False

store(Request(request))
    precondition :
        waiting := False
    Effect :
        resultReceived := False
        waiting := True

queryIf(Propose(servieOffer))
    precondition :

    Effect :
        resultReceived := False
```

The requester encapsulates a behavior associated with the evaluation of every received message proposal. Upon deciding on a certain service proposal, the requester responds and engages in an interaction with the Advertiser. The Intra-Brokering interaction follows the same patterns explained in the previous cases. The protocol comes to an end whenever the result of the service is delivered to the requester who has to acknowledge the receipt.

There are situations where requesters might inquire for more information relevant to the proposed service, such as delivery status, change or cancel requests and acknowledge receipt of service's result. The protocol has to automatically respond to such events and even pro-actively provide means to dynamically alert requesters about new events and conditions and therefore unpredictably perform some action at the time a given precondition becomes true. By giving requesters the ability to self-configure and request services autonomously, we allow for the possibility of self-configurable systems; thereby potentially increasing the degree of automation in the construction of software systems. Unfortunately, those requirements cannot be achieved when utilizing the object-oriented approach as the modeling paradigm. The pattern shown in Figure 11 represents a sequence of messages exchanged among the Advertiser and the requester. By abstracting away the internal, low-level behavior of the interactions and concentrating solely on the static and dynamic nature exhibited by the respective entity behaviors, one can establish a means by which these entities can interact and communicate effectively.

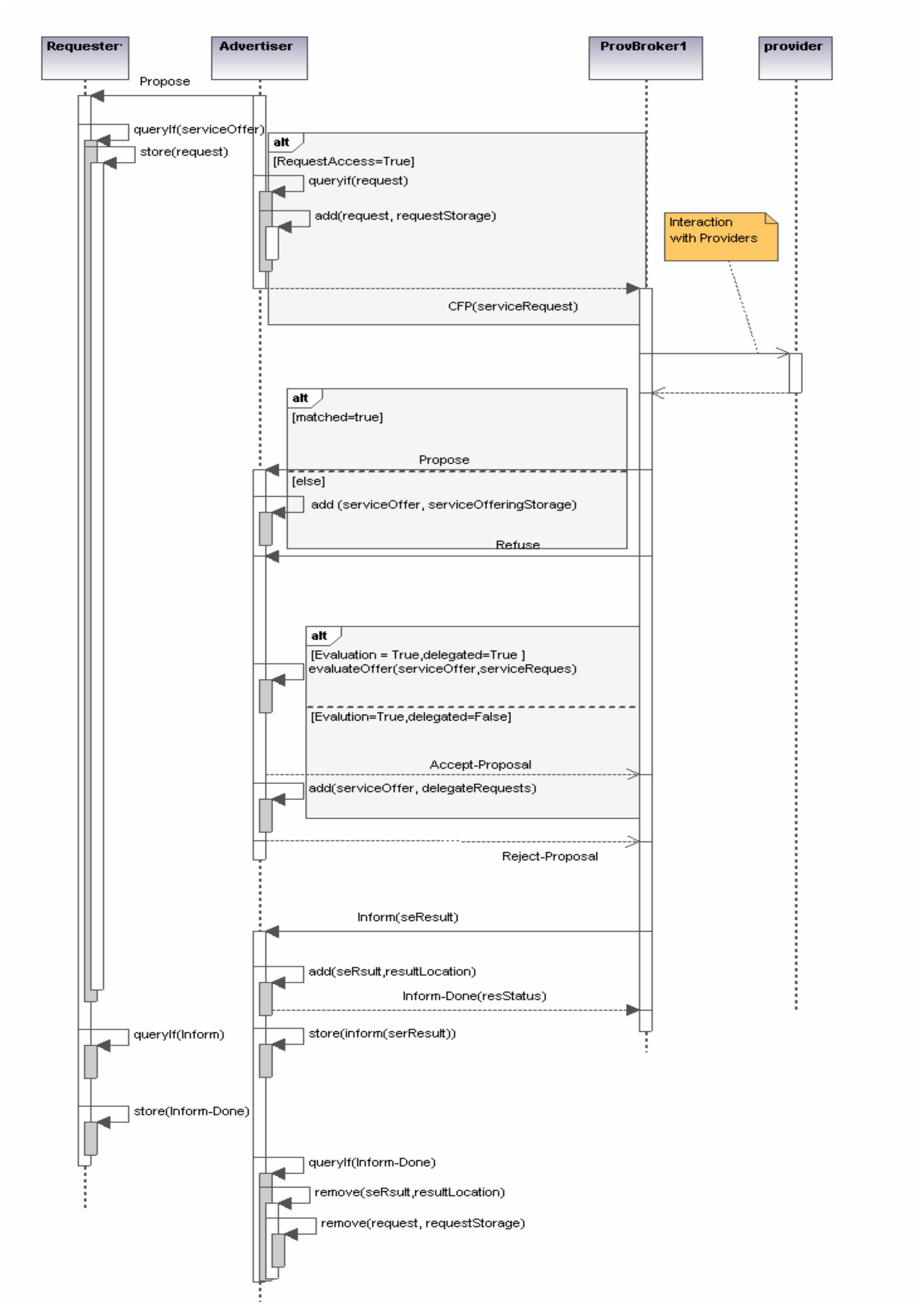
Requester
Advertiser
ProvBroker1
provider
Propose
queryIf(serviceOffer)
store(request)
alt
[RequestAccess=True]
queryif(request)
add(request, requestStorage)
Interaction with Providers
CFP(serviceRequest)
alt
[matched=true]
Propose
[else]
add (serviceOffer, serviceOfferingStorage)
Refuse
alt
[Evaluation = True,delegated=True]
evaluateOffer(serviceOffer,serviceReques)
[Evalution=True,delegated=False]
Accept-Proposal
add(serviceOffer, delegateRequests)
Reject-Proposal
Inform(seResult)
add(seRsult,resultLocation)
Inform-Done(resStatus)
queryIf(Inform)
store(inform(serResult))
store(Inform-Done)
queryIf(Inform-Done)
remove(seRsult,resultLocation)
remove(request, requestStorage)

Figure 11: Sequence Diagram for the Interaction Protocol of the Advertiser

3.4.1.4. The Bulletinboard

In some cases, requesters desire to hide their identities and requests from the entire environment For example, patients with narcotic-related problems (such as drug or alcohol addiction) can seek services that provide information about rehabilitation centers, specialized psychiatrists, or programs that will help overcoming a particular critical situation without revealing either their identities nor the desired information.

As shown in Figure 12, requesters will have the ability to either post their requests into physical storage facility (requests repository) or check the service offerings repository for services that might be of interest. In both cases, the requester stores the request in a special storage location (request repository). The Bulletinboard checks and identifies requests that need to be served and accordingly forwards them to the ProvBrokers.

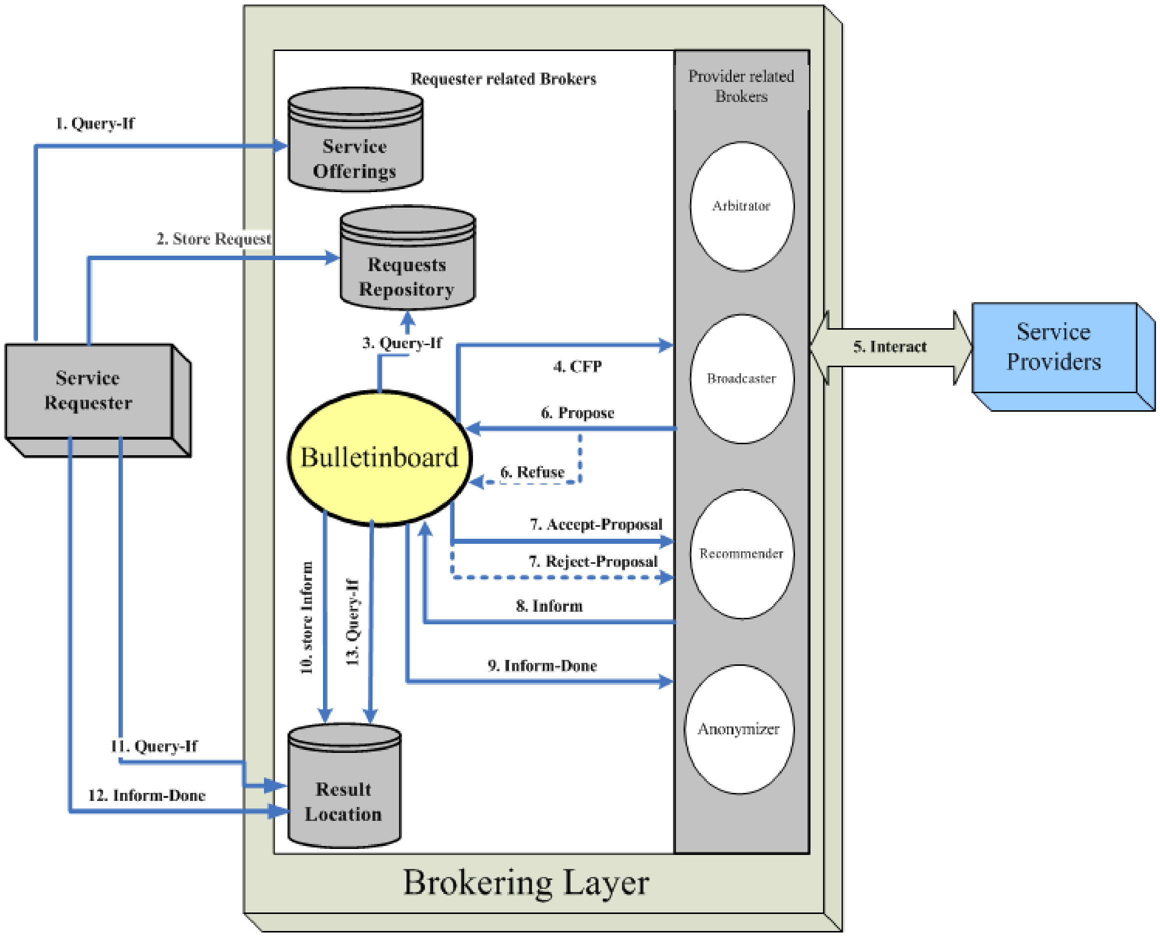

Figure 12: Interaction Pattern for the Bulletinboard

Note that, for this degree of privacy, the requester is responsible to check for the availability of the service's result and hence retrieve it. This implies that the requester should be aware of linking the result to its own request. The protocol is detailed as follows:

3. The Bulletinboard checks for ("**Request**") which indicates the availability of service requests.

4. Send ("CFP") – Sending a call for proposal message to ProvBrokers.

6. Receive ("Propose") – A Bulletinboard receives service proposal(s).

6. Receive ("Refuse") – A ProvBroker declines to participate in fulfilling a service request.

7. Send ("Accept-Proposal") – A message is sent to the wining provider indicating the acceptance of the proposal.

7. Send ("Reject-Proposal") – A rejection message is sent to those ProvBrokers who do not win.

8. Receive ("Inform") – The Bulletinboard receives the service's result.

9. Send ("Inform-Done") – the Bulletinboard informs the ProvBroker of the receipt of the service's result.

10. The Bulletinboard to store **("Inform")** indicating the availability of a service's result.

13. The Bulletinboard checks for **("Inform-Done")** that has been stored by the Requester (indicating the receipt of the result).

The requester interaction with the relevant Bulletinboard is restricted to the following:

1. Requester to check for ("**Propose**")**,** for service offerings that might be of interest.

2. The requester to store ("**Request**"**)** into a request repository.

11. The Requester checks for ("**Inform**") which indicates the availability of the result and hence retrieves it.

13. Upon retrieving the service result, the requester stores **("Inform-Done")** into the result repository.

The Bulletinboard Automaton

The input and output actions are similar to the actions generated during the interaction with the ProvBrokers. Figure 13 shows the state-machine representation for the Bulletinboard.

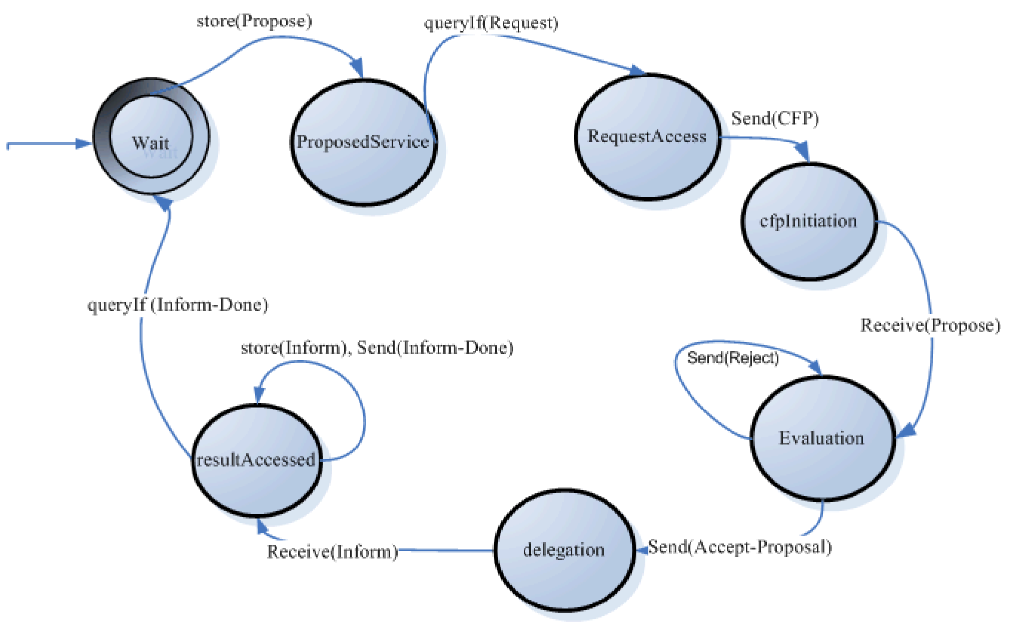

Figure 13: State Transition Diagram representing the Bulletinboard Behavior

```
store(Inform(serResult))
    precondition :
        $ResultAccessed := True$
        $resStatus := True$
   Effect :
        $ResultAccessed := True$

queryIf(Request(request))
    precondition :
        $ProposedService := True$
   Effect :
        $ProposedService := False$
        $RequestAccess := True$
        $requestStorage := True$

queryIf(Inform − Done())
    precondition :
        $RequestAccessed := True$
   Effect :
        $RequestAccessed := False$
        $requestStorage := False$
        $Wait := True$
```

The Requester Automaton

The requester hides the privacy attributes by setting the value of *reqID* in $\langle reqID, serName, reqPerf \rangle$ of the stored *request* to null. The Bulletinboard will not be able to deduce any further information from the stored request and thus the required privacy degree will not violated. The requester's actions and the associated transitions are described as follows:

```
store(Inform − Done(resStatus))
    precondition :
        resultReceived := True
    Effect :
        waiting := False

queryIf(Inform(serResult))
    precondition :
        waiting := True
        resultReceived := True
    Effect :
        resultReceived := False

store(Request(request))
    precondition :
        waiting := False
    Effect :
        resultReceived := False
        waiting := True

queryIf(Propose(servieOffer))
    precondition :

    Effect :
        resultReceived := False
```

As shown in Figure 14 the requester's interactions employ some degree of nondeterministic (or unpredictable) behavior.

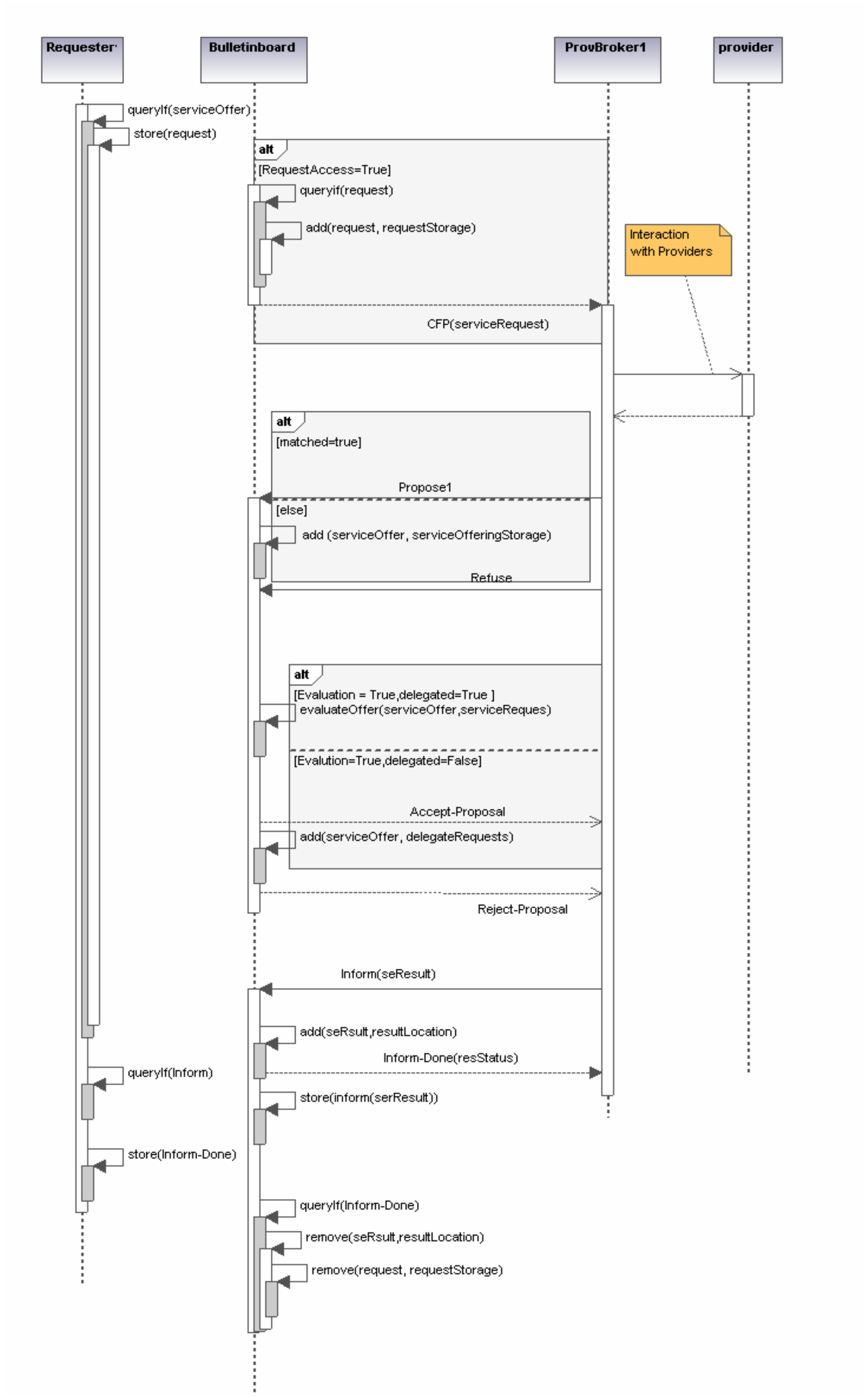
Requester
Bulletinboard
ProvBroker1
provider
queryIf(serviceOffer)
store(request)
alt
[RequestAccess=True]
queryif(request)
add(request, requestStorage)
Interaction with Providers
CFP(serviceRequest)
alt
[matched=true]
Propose1
[else]
add (serviceOffer, serviceOfferingStorage)
Refuse
alt
[Evaluation = True,delegated=True]
evaluateOffer(serviceOffer,serviceReques)
[Evalution=True,delegated=False]
Accept-Proposal
add(serviceOffer, delegateRequests)
Reject-Proposal
Inform(seResult)
add(seRsult,resultLocation)
Inform-Done(resStatus)
queryIf(Inform)
store(inform(serResult))
store(Inform-Done)
queryIf(Inform-Done)
remove(seRsult,resultLocation)
remove(request, requestStorage)

Figure 14: Sequence Diagram for the interaction Protocol of the Bulletinboard

When observed from the environment, the requester's behavior can range from being totally predictable to completely unpredictable. For example, a requester searching a repository for service offerings and looking for a service can appear to be exhibiting random behavior (might include identifying, choosing services, evaluating parameters and decide on the applicability or completely discard any offerings). However, once a service offering of interest is detected, its behavior becomes reasonably predictable. In contrast, the behavior of a Brokering layer might be highly unpredictable, the Bulletinboard needs to interact, choose, negotiate, and select potential service providers who are capable of fulfilling the service request, or it might return empty-handed.

The protocol can be only accomplished by interactive and autonomous entities that must be sensitive to their own set of internal responsibilities and be capable of using rich forms of messages. These messages can support method invocation—as well as informing other entities (brokering entities within the layer) of particular events, asking something, or receiving a response to an earlier query. Clearly objects lack the ability to initiate interaction, respond to a message in any way they choose, or decide not to participate. Additionally, the typical usage and direct support of object-oriented approaches leans toward a more predictable approach.

For instance, when a message is sent to an object, the method is predictably invoked. Yes, a requester modeled as an object may determine whether or not to choose and process messages related to posted service offerings and how to respond if it does (for example, storing a service request in a special repository). However, in common practice, if an object says no, it is considered an error situation.

It is noteworthy, that the underlying message exchange is usually a predefined flow of control from one object to another that has to be known a priori. Asynchronous messaging and event notification is not explicitly tied to the object's behavior. Within this context, systems that require such functionality have to layer these features on top of the object model and the Object-Oriented environment.

3.4.2. The Provider-Brokering Interaction Patterns

The interaction patterns allow providers to securely automate their privacy and advertise capabilities; define conditions and constraints that govern the provision of these capabilities.

Providers' capabilities are often described in terms of two main aspects, Functional and non-functional properties. The Functional properties capture the intended behavior of the service and define the input and output parameters. The input parameters specify the required information that is needed prior to any service provision, while the output parameters specify the result of the service execution (for example, a service provider with information gathering capabilities generate outputs in electronic PDF file). The non-functional properties exhibit the constraints over the functionality of a service and specify additional information about the service capabilities, such as availability, service quality, cost, payment, security, trust and ownership.

However, describing the providers' capabilities is beyond the scope of the work presented here. It is assumed that there are appropriate services and tools (for example, capability description languages) by which providers are able to describe the inherent capabilities.

The following interaction patterns depict the different brokering scenarios categorized by the privacy concerns of service providers. In all the interaction patterns, it is assumed that the ProvBrokers are able to interpret services' capabilities, match and locate providers who are capable of fulfilling a particular service request.

Note that in representing the different automat for the provider-brokering interaction action signatures of the ProvBrokers include the subsets of the input actions that are referred by the "*receive*" action to represent that the environment (being ReqBroker or a provider) is the source of the action. Whereas the output actions are referred by the "*send*" action to represent that the ProvBroker is the source of the action and can be consumed by any element of the environments. The sates are captures as variable labels with instantiation values

3.4.2.1. The Arbitrator

In many E-government[4] applications, the primary concern is to simplify the interaction with citizens and institutions. Many countries have established an on-line presence. In most cases, governments need to make decisions related to national security-threatening issues that might involve citizens, institutions and organizations.

However, making such decisions might require the collaboration of other parties (for example, intelligence-related services) who need to be protected anonymously from perspectives associated to their identities and capabilities. The Arbitrator provides coordination activities to those providers who can contribute collaboratively to provide services while shielding their identities and capabilities.

To exploit the gain of this collaboration, providers do not have to worry about their privacy from being known by other counterparties. Direct communication with the Arbitrator requires the revealing of the privacy attributes. The protocol must shield and suppress any other entity form coming to know these attributes. In order to satisfy this requirement, it is assumed that the Arbitrator supporting this privacy degree is a trusted entity.

Moreover, the Arbitrator (on behalf of the provider), engages in subsequent interactions with various ReqBrokers without revealing the privacy attributes of the provider. In other words, the identity of the Arbitrator is the only revealed attribute to other entities (ReqBrokers) when sending and receiving messages. Figure 15 depicts the interaction pattern for such a privacy case.

[4] E-government refers to the electronic delivery of government services to citizens.

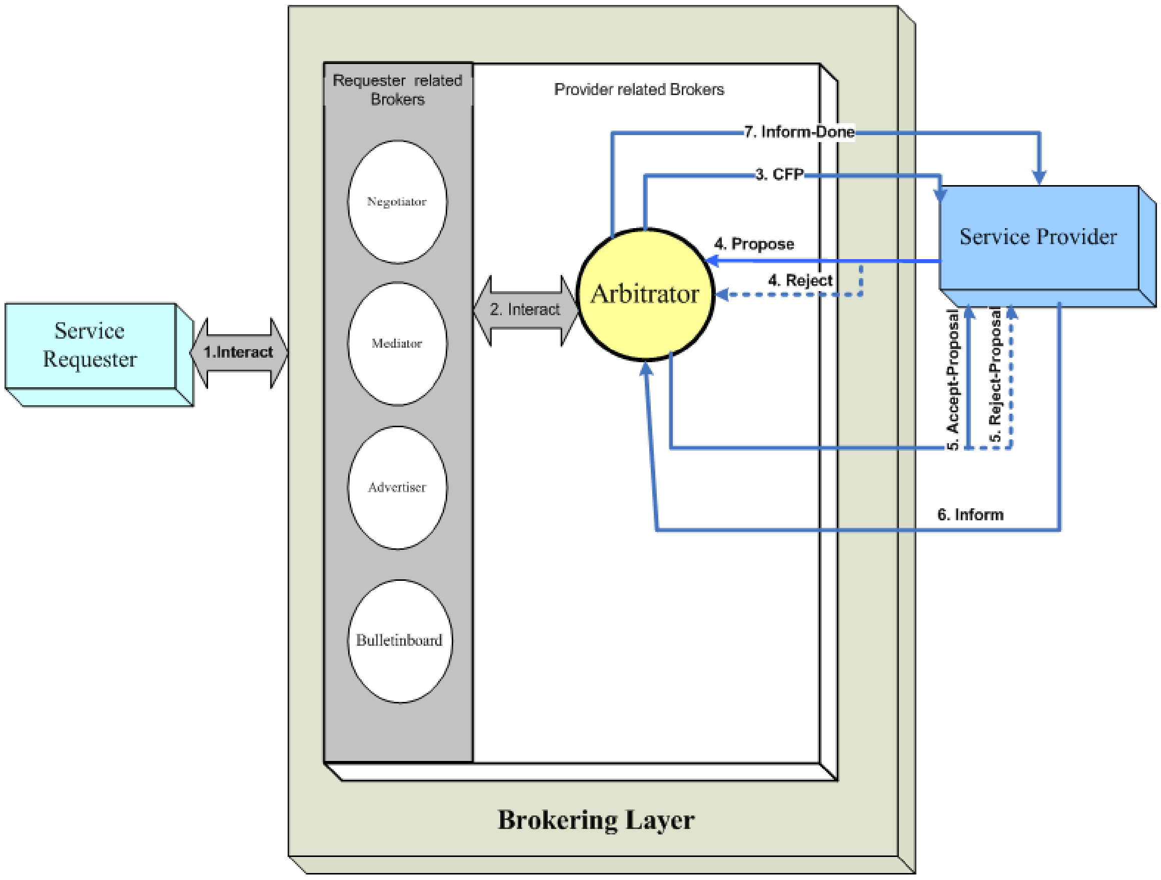

Figure 15: The Interaction Pattern for the Arbitrator

For every received service request (i.e. CFP messages received form various ReqBrokers), the Arbitrator matches the most appropriate providers to fulfill a particular request and accordingly sends the received CFP message to the matched ones.

Providers might contribute to fulfill received service requests by submitting proposals to the ProvBroker. On behalf of all potential providers, the Arbitrator sends the received proposals to the relevant pertinent ReqBroker which in turn determines and selects the appropriate service proposal.

Once the ReqBroker notifies the Arbitrator about the outcomes of the selection process, the Arbitrator will be able to issue an acceptance message to the corresponding winning provider and a dismiss message for each unselected provider. The proposed interaction pattern that supports this privacy degree will be as follows:

3. Send (“**CFP”**), the Arbitrator sends a call for proposals to all providers with known capabilities (this implies that the Arbitrator will be aware of providers who might satisfy a particular service request).

4. Receive (“**Propose**”), the Arbitrator receives service proposals from potential providers.

4. Receive (“**Reject**”), the Arbitrator receives a decline message from the provider.

5. Send (“**Accept-Proposal**”) – Upon receiving an acceptance message from the ReqBroker, the Arbitrator in turn notifies the provider (winner) accordingly.

5. Send (“**Reject-Proposal**”), a rejection message is sent to non-wining provider.

6. Receive (“**Inform**”), the Arbitrator receives the service’s result.

7. Send (“**Inform-Done**), the Arbitrator notifies the relevant provider of the receipt of the result.

The interaction pattern assumes that the protocol is initiated upon receipt of CFP message form the ReqBrokers within the brokering layer. The IOA representation is shown in Figure 16 and the IOA includes the following states:

- $RequestAccess$: represents the state in which the ProvBroker has received CFP messages from the ReqBrokers. It has two possible values $RequestAccess := True$ and $RequestAccess := False$.
- $Delegation$: represents a delegation of specific service request to a particular provider. It represents two possible values $Delegation := True$ and $Delegation := False$.
- $ResultAccesse$: represent results acknowledgments of service requests that need to be sent to the ReqBroker with two possible values $ResultAccesse := True$ and $ResultAccesse := False$.
- $cfpInitiation$: represents the state where the ProvBroker has issued CFP message to various providers with two possible values $cfpInitiation := True$ and $cfpInitiation := False$.
- $Evaluation$: denotes the state where the ProvBroker has received service offers from ProvBrokers with two possible values $Evaluation := True$ and $Evaluation := False$.

Similarly, the set of transitions are represented as action/precondition/postcondition model

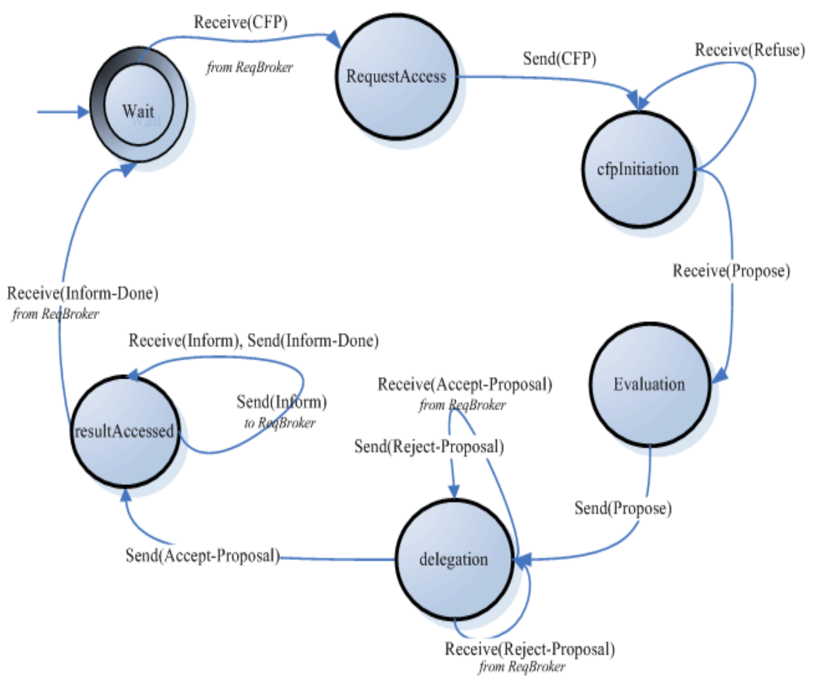

Figure 16: State Transition Diagram representing the Arbitrator Behavior

The Arbitrator Automaton

In addition to the input and output actions exchanged during the interaction with the ReqBrokers, the Arbitrator has another variable *matched* , which is of a Boolean type and with values in $\{True, False\}$. The variable is set to True when there is (are) provider(s) which fulfill a specific service's request. The Arbitrator behavior includes the following:

```
recieve(Propose(serviceProposal))
    precondition :
        cfpInitiation := True
    Effect :
        servcie := True

send(CFP(provID, serviceRequest))
    precondition :
        RequestAccess := True
    Effect :
        cfpInitiation := True

recieve(Inform(provID, resStatus))
    precondition :
        Delegation := True
        resStatus := False
    Effect :
        ReSult := True
        resStatus := True
        Delegation := False

send(Accept - Proposal(serviceProposal))
    precondition :
        Evaluation := True
        delegated := False
    Effect :
        Delegation := True
        Evaluation := False
```

```
$recieve(Refuse(serviceRequest))$
    $precondition :$
        $cfpInitiation := True$
    $Effect :$
        $cfpInitaition := True$

$send(Reject - Proposal(serviceProposal))$
        $Evaluation := True$
        $delegated := False$
    $Effect :$
        $Evaluation := True$
        $delegated := False$

$store(serResult, resultLocation, resStatus)$

    $Effect :$
        $resStatus := True$

$send(Inform - Done(provID, serResult))$
    $precondition :$
        $Result := True$
        $resStatus := Truse$
    $Effect :$
        $ReSult := True$
        $resStatus := True$

$store(serOffer, serviceOfferStorage)$

    $Effect :$
        $servcie := True$

$delete(Request, RequestStorage)$

    $Effect :$
        $requestStorage := False$

$delete(serResult, resultLocation, resStatus)$

    $Effect :$
        $resStatus := False$

$delete(serOffer, serviceOfferStorage)$

    $Effect :$
        $servcie := False$
```

The Provider Automaton

In additional to its own behaviour, the provider has the following additional variables:

- $waiting$, is a Boolean variable with values in $\{True, False\}$. True indicates that waiting for an acknowledgment of the result's receipt
- $resultRecieved$, is a Boolean variable with values in $\{True, False\}$ .True indicates the receipt of the result.

The actions generated by the provider and the associated transitions which are consumed by the Arbitrator are as follows:

```
$send(Infom(provBrokID, resStatus))$
    $precondition :$
        $resStatus := True$

    $Effect :$
        $waiting := True$

$send(Propose(serviceProposal))$
    $precondition :$
        $waiting := False$
    $Effect :$
        $waiting := True$
```

As shown in Figure 17, for every received service request, the Arbitrator matches the provider that is appropriate to fulfill a specific request. The Arbitrator initiates protocol by sending call-for-proposals (CFP) to those relevant providers (focusing mechanism) with known capabilities informing them of the service request's parameters and specifications. Upon receiving a CFP message, each potential provider evaluates the request parameters through invoking the *evaluate* method and thus decide on whether to participate in submitting service proposals or not.

In a dynamic environment in which providers are in continual increase and may unpredictably enter and leave, the Arbitrator's interaction is neither restricted to specific service providers nor committed to a fixed number of them. However, the Arbitrator may request only one operation, and that operation may only be requested via a message formatted in a very specific way.

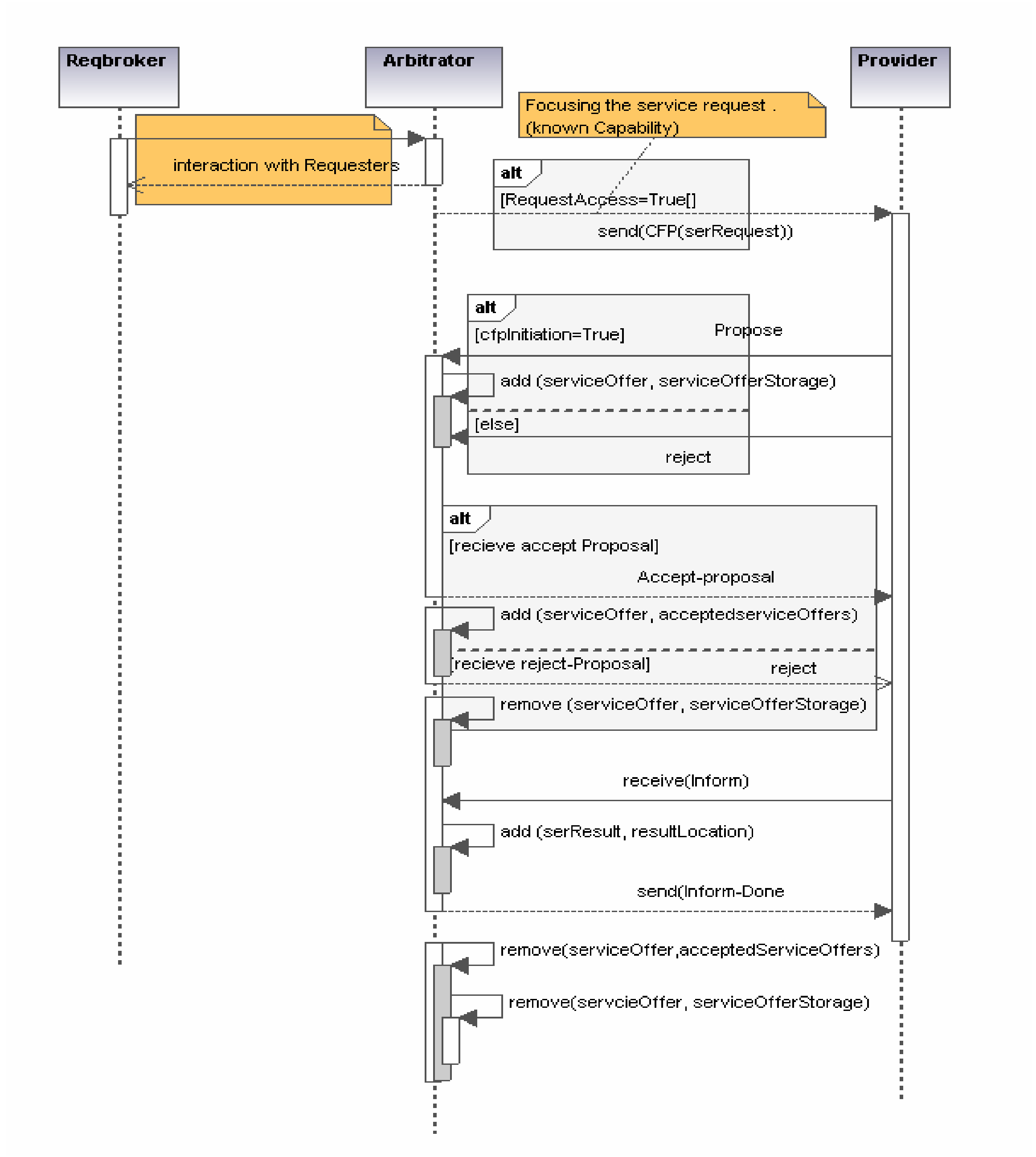
Reqbroker
Arbitrator
Provider
Focusing the service request . (known Capability)
interaction with Requesters
alt
[RequestAccess=True[]
send(CFP(serRequest))
alt
[cfpInitiation=True]
Propose
add (serviceOffer, serviceOfferStorage)
[else]
reject
alt
[recieve accept Proposal]
Accept-proposal
add (serviceOffer, acceptedserviceOffers)
[recieve reject-Proposal]
reject
remove (serviceOffer, serviceOfferStorage)
receive(Inform)
add (serResult, resultLocation)
send(Inform-Done
remove(serviceOffer,acceptedServiceOffers)
remove(servcieOffer, serviceOfferStorage)

Figure 17: Sequence Diagram for the Interaction Protocol of the Arbitrator

In other words, the Arbitrator has the job of matching each message to exactly one method invocation for exactly one object. Consequently, when the Arbitrator needs to send multiple requests to a single provider, those requests cannot be collected and delivered to the service as a single request with no major increase in the complexity of the ProvBroker or the provider--for example, the reduction in the number of roundtrip request-response activities. Since we may wish to send a message to any (and every) object, we need the expressive power to cover all desired situations, including method invocation. Therefore, a communication language is necessary for expressing communications among these objects.

The sequences of message exchanges can be more than just method invocation. Objects can be involved in long-term conversations and associations. It should be mentioned that the Arbitrator and the provider might engage in multiple transactions concurrently through the use of multiple threads or similar mechanisms. Each conversation has to be assigned a separate identity. Conventional object-oriented environments have difficulty supporting such a requirement.

More importantly, particularly for service providers with just a single method, the underlying services would not be part of the published interface. Advertising and publishing of a service's capabilities cannot be explicitly accomplished.

3.4.2.2. The Broadcaster

A number of small businesses want to use their recent point-of-sales data to cooperatively forecast future demand and thus make more informed decisions about inventory, capacity, employment, etc. Providing such capabilities and hiding the corresponding identities, would benefit all participants as well as the public at large. Providers avail themselves of more precise and reliable data collected from many sources, to assess their own local performance in comparison to global trends, and to avoid many of the inefficiencies that currently arise because of having less information available for their decision-making.

However, in a competitive environment, these small businesses might be influenced and monopolized by big dominant players if they reveal their identity and expose their relevant data. Therefore, those who are contributing and sharing reports require an access to a common pool to indirectly communicate their findings. A community of providers needs to share each other's resources (point-of-sale data) to create cooperative environment and securely prevent undesirable outcomes from revealing their identities.

The protocol permits various to hide their identities and reveal their service offerings to the relevant ProvBroker. The Broadcaster grants providers an access to various repositories (such as request repository, service repository and a result repository) either for a limited number of times or only for a limited-time period (for example during the active involvement of the provider in the corresponding interaction protocol). Service requests are posted to a dedicated repository which can be accessed by providers as shown in Figure 18.

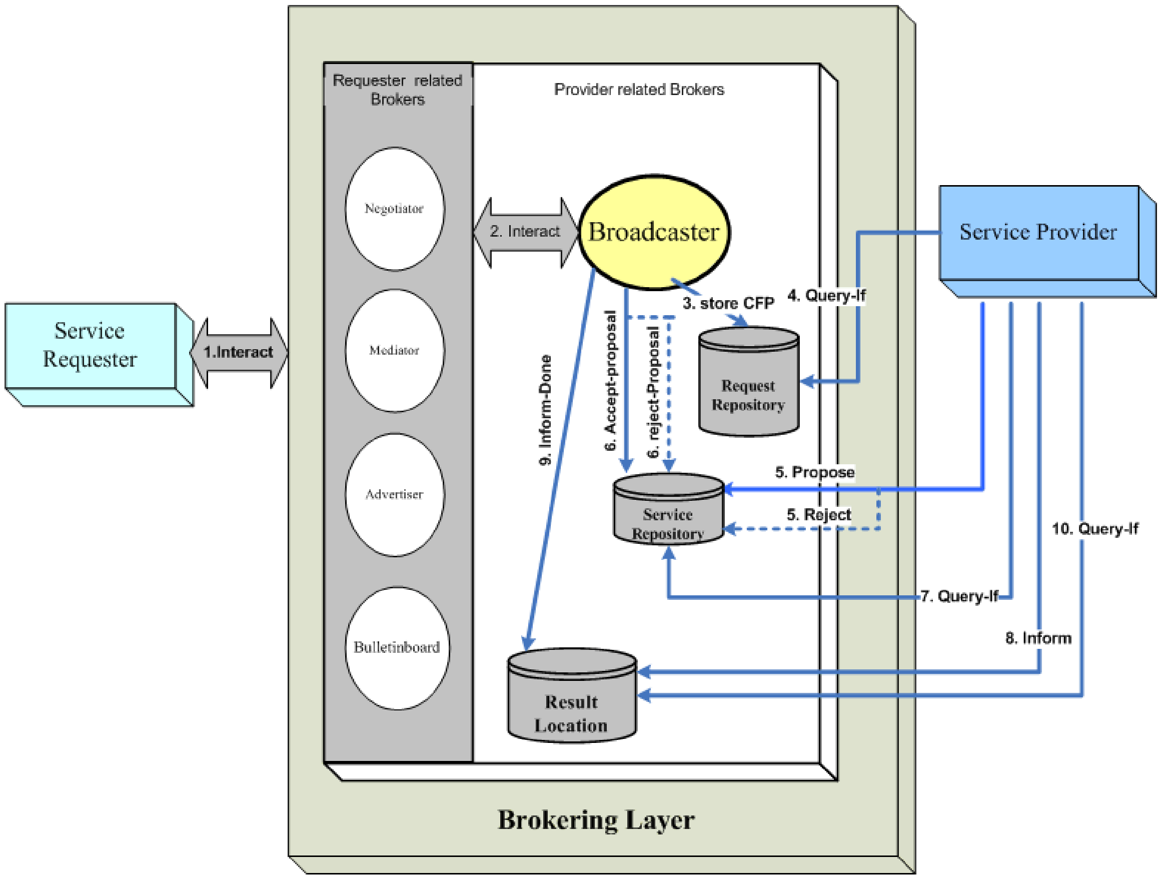

Figure 18: The Interaction Pattern for the Broadcaster

A provider may respond to call-for-proposal request by an offer posted onto a repository. Upon delegating a service request to a provider, the provider post service results to be retrieved by the Broadcaster and delivered to the proper destination. This sequence of events is shown below:

3. The Broadcaster to store ("**CFP**") in request repository.
4. Provider to check for posted services' requests, ("**CFP**").
5. Provider to store ("**Propose**"), indicating a proposed service for a particular request.

5. Provider to store ("**Refuse**"), indicating the refusal for serving a particular request.

6. The Broadcaster to store **("Accept-Proposal")** – Indicating an acceptance message;

6. The Broadcaster to store **("Reject-Proposal") –** Indicating a rejection message for a proposed service.

7. Provider to check for services proposal acceptance ("**Propose**").

8. Provider to store **(Inform)** indicating the availability of a service's result.

9. The Broadcaster to store **(Inform-Done)** upon retrieving the service's result.

10. Provider to check for the receipt of the service's result ("**Inform-Done**").

The Broadcaster Automaton

The input and output actions for the Broadcaster are similar to the actions generated during the interaction with the ReqBrokers. The internal actions and the transitions are shown in Figure 19 and are described as follows:

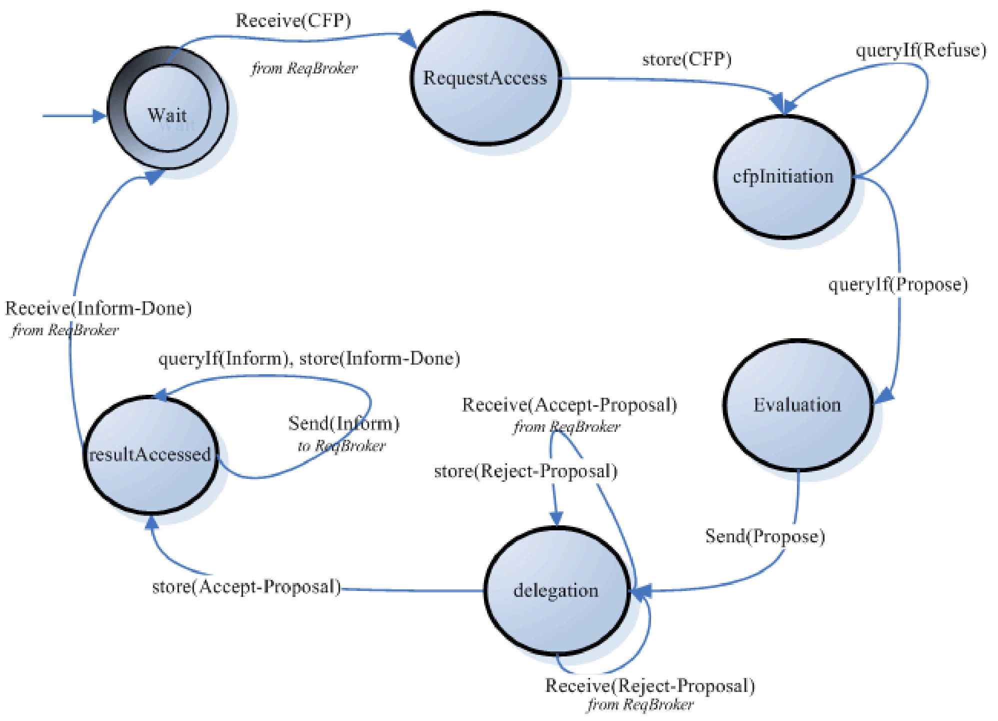

Figure 19: State Transition Diagram representing the Broadcaster Behavior

```
store(CFP(serviceRequest))
    precondition :
        RequestAccess := True
    Effect :
        cfpInitiation := True

store(Accept - Proposal(serviceProposal))
    precondition :
        Wait := True
    Effect :
        RequestAccess := True
        RequestStorage := True

store(Reject - Proposal(serviceProposal))
    precondition :
        Wait := True
    Effect :
        RequestAccess := True
        RequestStorage := True

queryIf(Propose(serviceProposal))
    precondition :
        RequestAccess := True
    Effect :
        cfpInitiation := True

queryIf(Inform(serResult))
    precondition :
        RequestAccess := True
    Effect :
        cfpInitiation := True

store(Inform − Done(serResult))
    precondition :
        RequestAccess := True
    Effect :
        cfpInitiation := True
```

The Provider Automaton

The provider hides its identity by setting the value of $provID$ in $\langle provID, serName, serPar \rangle$ of the stored $serviceProposal$ to null. The Broadcaster will not be able to deduce any further information from the stored service proposal and therefore the privacy attributes will be protected. The provider's actions and the associated transitions are described as follows:

```
store(Infom(provBrokID, resStatus))
    precondition :
        resStatus := True

    Effect :
        waiting := True

stor(Propose(serviceProposal))
    precondition :
        waiting := False
    Effect :
        waiting := True
```

As shown in Figure 20, the protocol permits providers to browse a special repository for service requests through invoking the *query-if* method on the serviceRequestStorage.

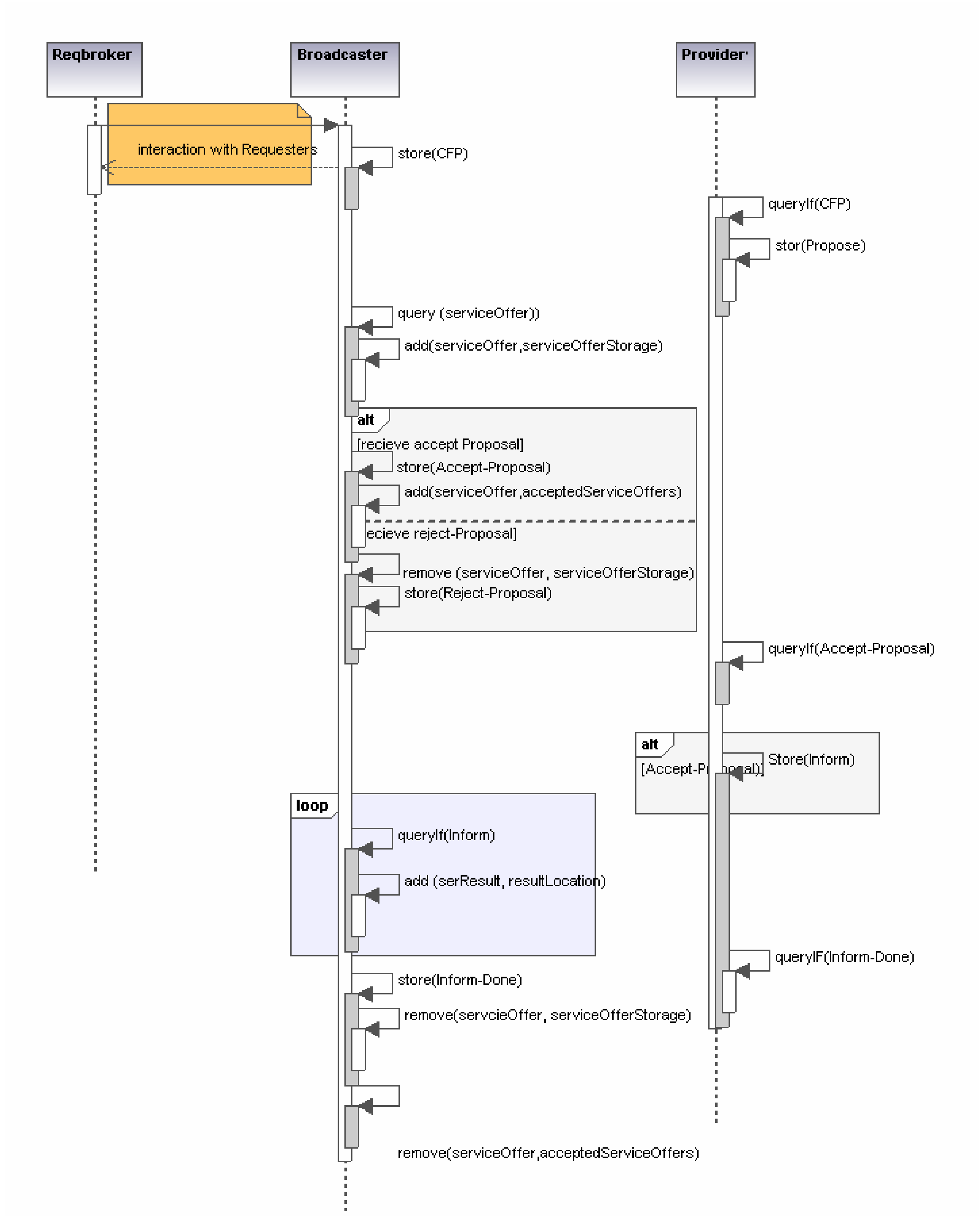
Reqbroker
Broadcaster
Provider
interaction with Requesters
store(CFP)
queryIf(CFP)
stor(Propose)
query (serviceOffer))
add(serviceOffer,serviceOfferStorage)
alt
[recieve accept Proposal]
store(Accept-Proposal)
add(serviceOffer,acceptedServiceOffers)
ecieve reject-Proposal]
remove (serviceOffer, serviceOfferStorage)
store(Reject-Proposal)
queryIf(Accept-Proposal)
alt
[Accept-Proposal)]
Store(Inform)
loop
queryIf(Inform)
add (serResult, resultLocation)
queryIF(Inform-Done)
store(Inform-Done)
remove(servcieOffer, serviceOfferStorage)
remove(serviceOffer,acceptedServiceOffers)

Figure 20: Sequence Diagram for the Interaction Protocol of the Broadcaster

The provider checks needs to include mechanisms that facilitate polling the environment for service offerings, store service proposals and deliver results. Considering the dynamic nature of the environment, it is very common for the provider with this privacy degree to engage in multiple parallel interactions with other entities. Providers need to poll the environment for events and other messages to determine what action they should take. Additionally, providers need not to react on specific method invocations only, but rather on observable events within the environment as well. The behavior of d the provider should be then based on individual goals and states, as well as the states of ongoing conversations with each other.

3.4.2.3. The Recommender

Another setting where hiding provider's capability is a useful situation. Consider a new product that has been introduced to the market such that no single (even very large) retailer can accurately predict consumer demand for it. This happens when different retailers target different groups of customers, for which shopping patterns and adaptability to new products vary. Then it is beneficial to all such stores to engage into joint forecasting, while still preserving the privacy of the encapsulated capability.

After receiving a service request, the Recommender sends it to every provider with unknown capabilities. Figure 21 shows the associated interaction pattern.

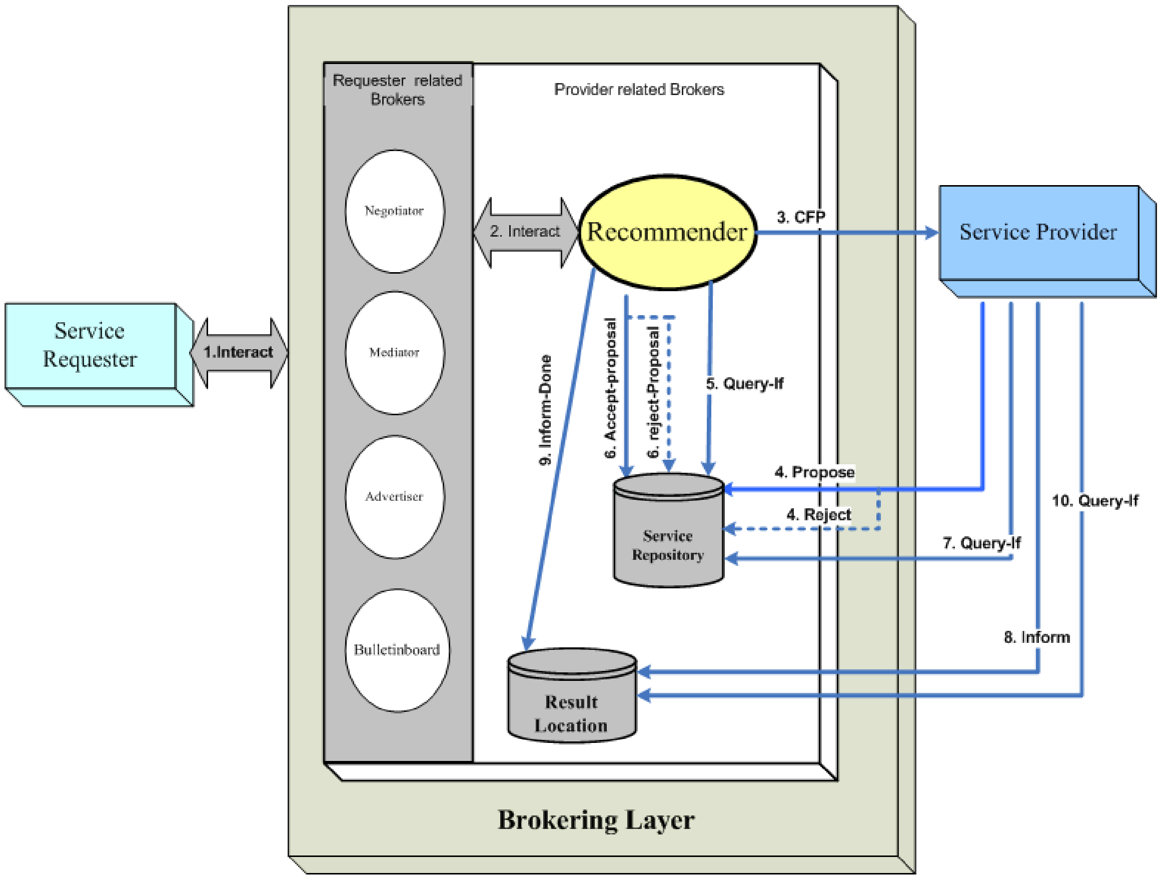

Figure 21: Interaction Pattern for the Recommender

Once a provider selects a particular service request, it sends a service proposal to the Recommender who controls the remaining transaction according to the appropriate negotiation mechanisms that are similar to what has been described in former patterns.

3. Send ("**CFP**"), the Recommender sends a call for proposal message to providers.

4. Provider to store ("**Propose**"), indicating a proposed service for a particular request.

4. Provider to store ("**Refuse**"), indicating the refusal for serving a particular request.

5. The Recommender to check for services proposals ("**Propose**").

6. The Recommender to store **("Accept-Proposal")** – Indicating an acceptance message;

6. The Recommender to store **("Reject-Proposal") –** Indicating a rejection message for a proposed service.

7. Provider to check for proposal acceptance ("**Accept-Proposal**").

8. Provider to store **(Inform)** indicating the availability of a service's result.
9. The Recommender to store **(Inform-Done)** upon retrieving the service's result.
10. Provider to check for the receipt of the service's result ("**Inform-Done**").

The Recommender Automaton

As shown in Figure 22, the Recommender has an additional output action related to sending CFP message to the provider. The Recommender's actions and the transitions are described as follows:

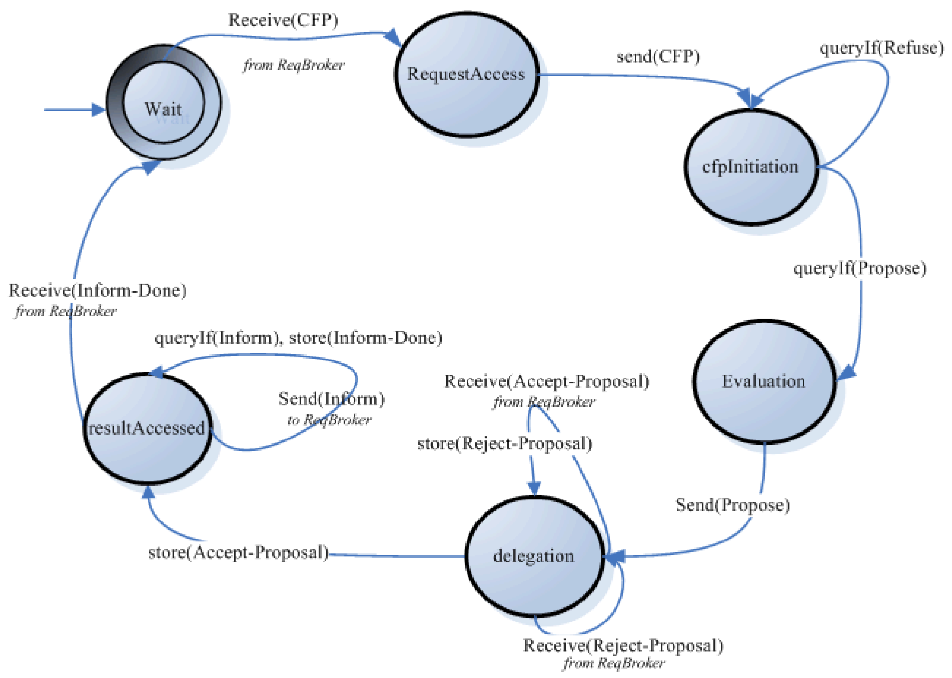

Figure 22: State Transition Diagram representing the Recommender Behavior

send(CFP(serviceRequest))
    precondition :
        RequestAccess := True
   Effect :
       cfpInitiation := True

queryIf(Propose(serviceProposal))
    precondition :
        RequestAccess := True
   Effect :
       cfpInitiation := True

```
$store(Inform - Done())$
    $precondition:$
        $RequestAccess := True$
  $Effect:$
        $cfpInitiation := True$

$store(Accept - Proposal(serviceProposal))$
    $precondition:$
        $Wait := True$
  $Effect:$
        $RequestAccess := True$
        $RequestStorage := True$

$store(Reject - Proposal(serviceProposal))$
    $precondition:$
        $Wait := True$
  $Effect:$
        $RequestAccess := True$
        $RequestStorage := True$

$queryIf(Inform(serResult))$
    $precondition:$
        $RequestAccess := True$
  $Effect:$
        $cfpInitiation := True$
```

The Provider Automaton

Although, the provider is revealing its identity, it is required to set the value of $provID$ in $\langle provID, serName, serPar \rangle$ of the stored $serviceProposal$ to null when proposing services to a particular request. In this case, the stored proposal will be of an anonymous originator, and therefore, the Recommender will not be able to deduce any further information that might link the capability to the identity of the participating provider. The provider's actions and the associated transitions are described as follows:

```
$store(Infom(provBrokID, resStatus))$
    $precondition:$
        $resStatus := True$

  $Effect:$
        $waiting := True$

$stor(Propose(serviceProposal))$
    $precondition:$
        $waiting := False$
  $Effect:$
        $waiting := True$
```

The sequence diagram depicted in Figure 23 illustrates the situation where a particular CFP message received from ReqBrokers is sent out to every registered provider with unknown capabilities. Depending on the state, conditions and rules of the involved objects, alternative courses of action are to be followed in different circumstances. Output messages are guarded by predefined conditions for which the activation methods of these messages vary.

For example, for every received service request, the provider has to determine whether the requested service is within its capabilities and/or of interest and accordingly decide on either to participate, reject or simply ignore such requests. Once a provider selects a particular service request, it responds with a service proposal. The Recommender controls the remaining message exchange according to the sequence defined in the diagram.

Providers hiding their capabilities will be flooded by a variety of service requests, for which a behavior that is associated with the evaluation of every received request needs to be included. Upon deciding on a certain function to be satisfied, the provider responds and engages in an interaction with the ReqBroker. The protocol comes to an end whenever the result of the service is delivered to the Recommender who has to acknowledge the receipt. However, in cases where providers inquire for more information relevant to the service request, cancel a service offer or negotiate terms, the protocol has to automatically and dynamically allow such conversations rather than statically invoke predefined methods. Unfortunately, those requirements cannot be accomplished when utilizing the object-oriented approach as the modeling paradigm.

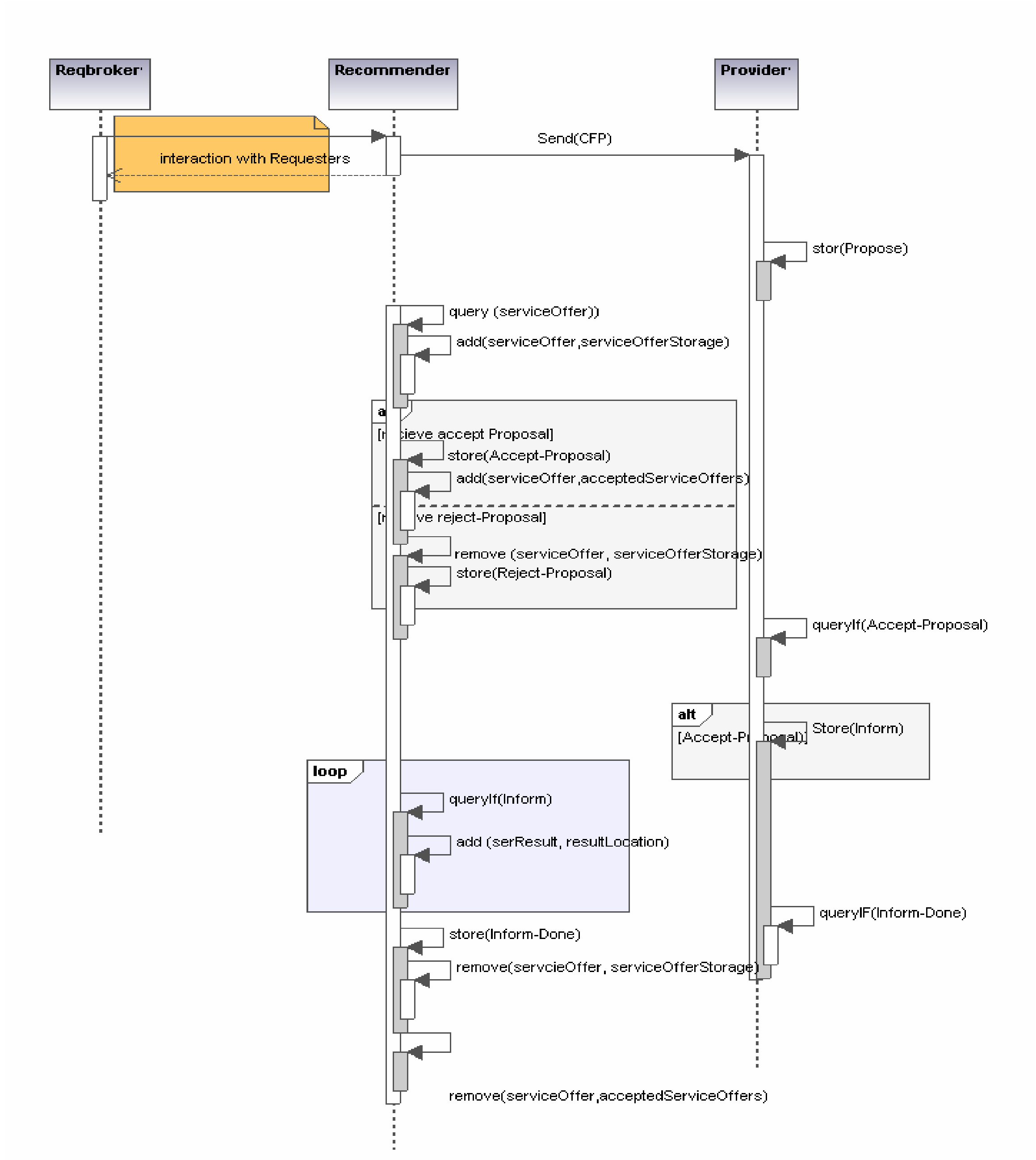

Figure 23: Sequence Diagram for the Interaction Protocol of the Recommender

3.4.2.4. The Anonymizer

Investigators and private detectives can provide a valuable aid in background checks, investigations concerning law suits and liability, crimes, fraudulent insurance claims investigations, and a variety of other situations. In many cases, investigation procedures might extend to span various distributed geographical locations and might involve the collaboration of several entities. However, the willingness of private investigators and detectives to assist in critical and personal-related issues is highly impacted by the level of guarantees and assurance exhibited towards the protection of their identities as well as the nature of the conducted work.

In such situations, providers would prefer to have secure and safe means that enable them to engage in sharing their capabilities while protecting their privacy attributes. Each provider with this privacy degree will be able to view information relevant to desired requests. A provider contributes to the fulfillment of these requests by proposing services to the designated Anonymizer whose functionality includes the ability to view and send any stored proposals to ReqBrokers. Moreover, it is assumed that the Anonymizer has the ability to match and determine capable providers with the most insight towards fulfilling the service request. Different storage repositories are available to the provider to access as shown Figure 24.

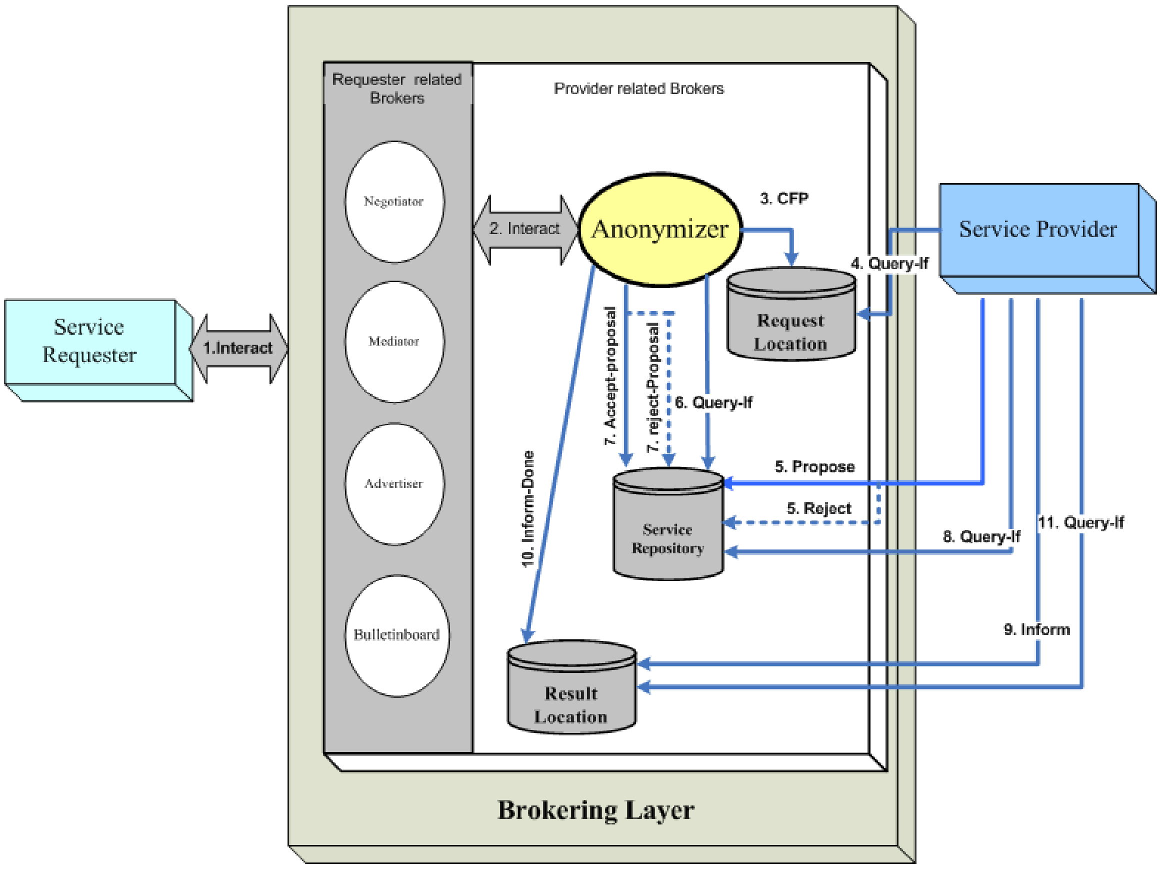

Figure 24: The Interaction Pattern for the Anonymizer

The interaction pattern will be as follows:

3. The Anonymizer to store ("**CFP**") message.

4. The provider to check for services requests, ("**CFP**").

5. Provider to store ("**Propose**") message indicating a proposed service for a particular request.

5. Provider to store ("**Refuse**"), indicating the refusal for serving a particular request.

6. The Anonymizer to for services proposals, ("**Propose**").

7. The Anonymizer to store **("Accept-Proposal")** – Indicating an acceptance message;

7. The Anonymizer to store **("Reject-Proposal") –** Indicating a rejection message for a proposed service.

8. Provider to check for **("Accept-Proposal")** which indicates an acceptance of service's proposal.

9. Provider to store **(Inform)** indicating the availability of a service's result.

10. The Anonymizer to store **(Inform-Done)** upon retrieving the service's result.

11. Provider to check for the receipt of the service's result, **(Inform-Done)**.

The Anonymizer Automaton

The input and output actions for the Anonymizer are similar to the actions generated during the interaction with the ReqBrokers as shown in Figure 25. The internal actions and the transitions described as follows:

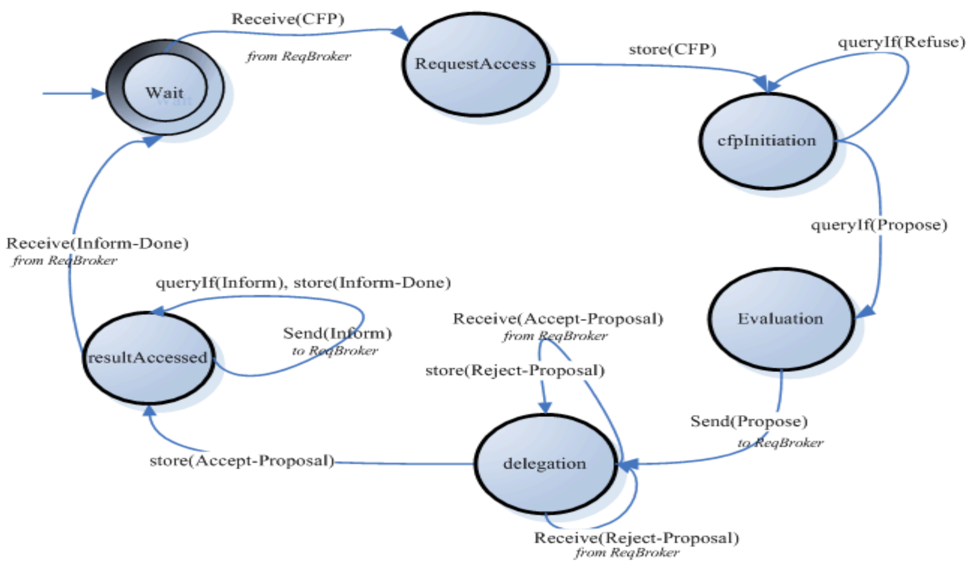

Figure 25: State Transition Diagram representing the Anonymizer Behavior

store(CFP(serviceRequest))
precondition :
RequestAccess := True
Eff :
cfpInitiation := True

store(Accept - Proposal(serviceProposal))
precondition :
Wait := True
Eff :
RequestAccess := True
RequestStorage := True

store(Reject - Proposal(serviceProposal))
precondition :
Wait := True
Eff :
RequestAccess := True
RequestStorage := True

queryIf(Propose(serviceProposal))
precondition :
RequestAccess := True
Eff :
cfpInitiation := True

queryIf(Inform(serResult))
precondition :
RequestAccess := True
Eff :
cfpInitiation := True

store(Inform – Done(serResult))
precondition :
RequestAccess := True
Eff :
cfpInitiation := True

The Provider Automaton

The provider hides the identity by setting the value of *provID* in $\langle provID, serName, serPar \rangle$ of the stored *serviceProposal* to null. The Anonymizer will not be able to deduce any further information from the stored service proposal and therefore both privacy attributes (identity and capability) will be protected. The provider's actions and the associated transitions are described as follows:

```
store(Infom(provBrokID, resStatus))
    precondition :
        resStatus := True

    Effect :
        waiting := True
```

```
store(Propose(serviceProposal))
    precondition :
        waiting := False
    Effect :
        waiting := True
```

The protocol permits providers to search a special repository for service requests by invoking the *query-if* method on the serviceRequestStorage. Upon deciding on a particular request, the ReqBrokoker4 invokes the "*store*" method on the serviceOfferStorage object to store a service proposal as shown in Figure 26.

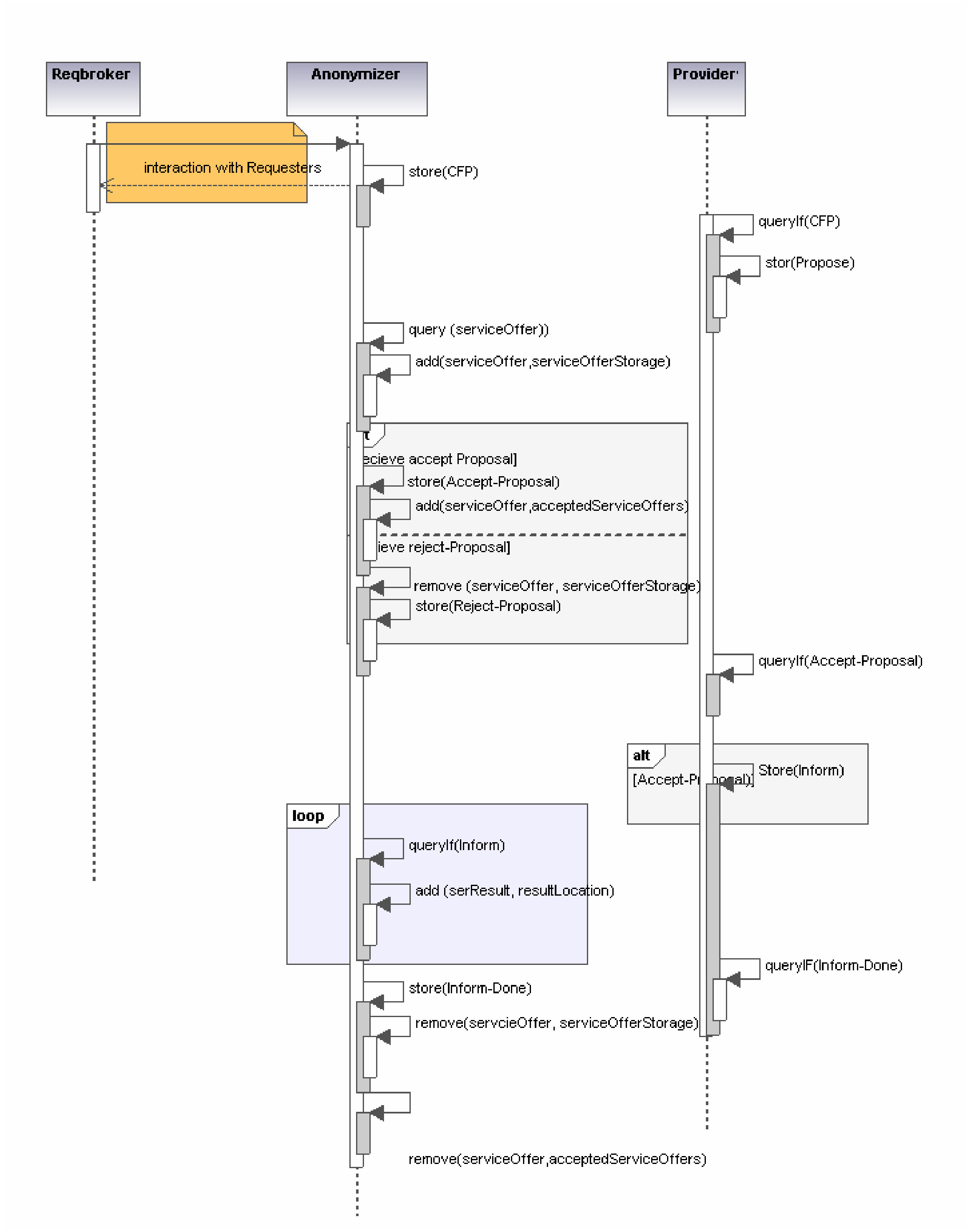

Figure 26: Sequence Diagram for the Interaction Protocol of the Anonymizer

Similar to the case of requestors hiding privacy attributes, the interactions employ some degree of nondeterministic (or unpredictable) behavior. The provider's behavior when browsing a repository for service offerings and looking for service requests appear to be randomly executed (might include identifying, choosing CFP messages, evaluating parameters and deciding on the applicability or completely discarding any service requests). The behavior of Anonymizer will be highly unpredictable in the sense that it needs to interact, choose, negotiate, and select potential service providers who are capable of fulfilling the service request, or might return with nothing at all.

Modeling the provider and the relevant Anonymizer as objects with the nature of being non-interactive and semi-autonomous would not be an appropriate choice to satisfy the required functionality. Clearly the lack of ability to initiate interaction, respond to a message in any way they choose, or decide not to participate imposes considerable limitations to capturing unpredictable and nondeterministic behavior.

3.5. Analysis of the Protocols

In the preceding sections, we have outlined a framework for identifying and characterizing privacy-based coordination solutions in cooperative distributed systems. In this section, we propose a generic architecture which puts forth a vision of how cooperation and coordination can be supported, while addressing the possible privacy concerns of the various CDS entities. The architecture identifies the major architectural aspects and entities types, what kinds of functions they perform, what information they maintain, and what kinds of interdependencies they manage in order to deal with various privacy requirements.

This approach of providing a layered architecture for cooperation support is complementary to approaches which attempt to build cooperation and privacy solutions as value-added services. In addition to dealing with coordination, we believe that this architecture is also useful in providing a framework for delivering privacy-base brokering services within an organization, in support of business processes, and in aligning them with organizational objectives. The brokering layer manages different interaction protocols that support various level of privacy; each is providing an inherent control for the dissemination and the distribution of critical information.

The interaction protocol specification provides guidelines for building privacy-based brokering applications, which has to include the following elements:

- Types and roles of participants.
- Interaction states.
- Events which trigger states changes.
- Valid actions, constraints and message types.

The different parts of the previous protocols exhibit the dynamic behavior of the entities (requesters, brokers and providers) involved in desired privacy degrees. Similarities between the different behaviors in the protocols can be observed, in which the same pattern of message exchanges is repeated in many parts of the protocol.

UML provides a means of expressing such an aggregation both structurally and behaviorally: *components* provide physical aggregations that compose classes for implementation purposes and *packages* aggregate modeling elements into a high conceptual level. From the previous sequence diagrams, the protocols depict three main interaction protocols:

(1) Requester-Broker Interaction

(2) Broker-to-Broker Interaction

(3) Broker-Provider Interaction

Each Interaction represents a sequence of messages which collectively achieve the respective goals of the participants. The sequence diagrams detail which actions are appropriately performed at each point of the interaction, what are the applicable conditions and constraints that need to be met prior to any action execution, and how the selected actions can change and affect the state of the world by producing a potential interaction.

For every service request, the brokering entity communicates, collaborates and negotiates on behalf of the requesters to fulfill that particular request. Despite the fact that the brokering protocols can be complex and non-deterministic, requesters and service providers need not to be concerned with such complexities. From the brokering perspective, these protocols can be represented at high abstraction levels that capture only the essential and relevant characteristics of the protocol and ignore other interaction details.

The previous sequence diagrams describe an allowed sequence of messages and message content among entities. They depict a set of agreed upon messages, rules for actions based upon the reception of various messages, and the assumptions made based on these messages. These constraints, rules and patterns can be abstracted and formalized at a high level of abstraction (knowledge level) that provides a concrete basis for coordinated autonomous behavior.

However, in order to implement the brokering protocols (under the assumption of open environments) there are certain points to be considered:

(1) Entities involved in any brokering scenario must be capable of initiating action independent of any other entity. Such autonomy is best characterized in degrees, rather than simply being present or not. To some extent, entities need to cooperate without direct external invocation or intervention.

(2) Entities can react not only to specific method invocations but also to observable events within the environment (for example, in some of the previous protocols, entities need to poll the environment for requests, events or messages to determine what actions they should take).

(3) The IOA depicting the brokering protocols employ some degree of unpredictable (or nondeterministic) behavior. The entity's behavior can range from being totally predictable to completely unpredictable. For example, a requester entity hiding its requests from the environment might roam around looking for services that might be of interest appears to be exhibiting random behavior (searching service repositories, posting service's requests or service providers' capabilities). However, once a service of an interest is detected, its behavior becomes reasonably predictable. In contrast, the behavior of a brokering entity might be highly unpredictable. It is difficult to predict which service provider the brokering entity will interact with, negotiate and possibly select. In fact, a brokering entity can participate in a brokering scenario and might return empty-handed when it fails to find a service that matches the service's request criteria.

(4) In open dynamic environments, a more complex degree of interaction would include entities that can react to observable events. The IOA representing the

behaviors of the brokering entities do not illustrate method invocations on other entities and present the possibility of engaging in multiple, parallel interactions with another entity. Entities might be involved in multiple long-term conversations and associations concurrently. Messages in particular interaction patterns can be assigned a separate identity (for example, requesting service, call for proposals, or evaluating proposals).

(5) The protocols represent the possibility for entities to dynamically change their configurations to play multiple roles at the same time or at different times in different domains.

3.5.1. The Privacy-Based Brokering Protocols

Because of the fact that the brokering protocols can be described as recognizable patterns of a specific interaction, they can be treated as reusable aggregates of computation processes and modeled into conceptual wholes. These patterns can be combined and expressed at different levels of abstraction in which the behavior and the functionalities of the entities should be characterized by a succinct and precise description through an interface (Thus, capturing the essence of the behavior of the entity). Therefore, the repeated patterns of the brokering entities can be packaged into various sets of high level protocols. These patterns are arranged into the following protocols:

1. **Service Soliciting Protocol**: this protocol allows domain agents playing the role of requesters to solicit help from the brokering layer. The protocol consists of two sub-protocols that support the following modes:

    a. **Direct Soliciting Mode**: in which the brokering agent receives service requests directly from the requester agent. This mode allows the requester to directly solicit help by sending its service request through the message performative REQUEST. The pattern is as follows**:**

    - **Receive ("Request")** – The ReqBroker receives a request for service from the requester.
    - **Send ("Inform")** – The ReqBroker delivers back the service's result

- **Receive ("Inform-Done")** – A confirmation message is received from the requester.

b. <u>Indirect Soliciting Mode</u>: This mode supports the interaction with requesters hiding one of their privacy attributes. The ReqBroker will be able to retrieve a stored service request, store service's result and query about the receipt of a service's result. The protocol has the following message pattern:

- The ReqBroker checks for **("Request")** message for any available service requests that were stored by requesters and need to be served.
- The ReqBroker to store **("Inform")** indicating the availability of a service's result.
- The ReqBroker checks for **("Inform-Done")** that has been stored by the Requester (indicating the receipt of the result).

2. **<u>Contracting Protocol</u>**: this protocol abstracts all messages exchanged between the brokering agents (ReqBrokers and ProvBrokers) and contains all the behavior relevant to call for proposals, bidding, evaluating proposals and awarding/ rejecting service proposal as follows:

- **Send ("CFP")** – Sending a call for proposal message to ProvBrokers.
- **Receive ("Propose")** – A ReqBroker receives service proposal(s).
- **Receive ("Refuse")** – A ProvBroker declines to participate in fulfilling a service request.
- **Send ("Accept-Proposal")** – A message is sent to the wining ProvBroker indicating the acceptance of the proposal.
- **Send ("Reject-Proposal")** – A rejection message is sent to those ProvBrokers who do not win.
- **Receive ("Inform")** – The ReqBroker receives the service's result.

- **Send ("Inform-Done") –** the ReqBroker informs the ProvBroker of the receipt of the service's result.

3. **<u>Service Delivery Protocol</u>**: this protocol abstracts all the messages and the behaviors relevant to provide specific services. The package includes two main sub-protocols, namely:

   a. **<u>Direct Delivery Mode</u>**: This mode allows a provider revealing its privacy attributes to respond directly to a CFP message by proposing a specific service offering to the corresponding ProvBroker. The protocol supports the following pattern:

      - **Send ("CFP")** – Sending a call for proposal message to the provider.
      - **Receive ("Propose")** – A ProvBroker receives service proposal.
      - **Receive ("Refuse")** – A provider declines to participate in fulfilling a service request.
      - **Send ("Accept-Proposal")** – A message is sent to the provider indicating the acceptance of the proposal.
      - **Send ("Reject-Proposal")** – A rejection message is sent to those providers who do not win.
      - **Receive ("Inform")** – The ProvBroker receives the service's result.
      - **Send ("Inform-Done") –** the ProvBroker informs the provider of the receipt of the service's result.

   b. **<u>Indirect Delivery Mode</u>**: in which the service is stored into a repository and to be retrieved by the corresponding entity. This mode entitles the ProvBroker to store responses and to query about replies associated with a specific CFP message. The protocol includes the following:

      - The ProvBroker to store **("CFP")** into the request repository**.**
      - The ProvBroker to store **("Accept-Proposal")** – Indicating an acceptance message;

- The ProvBroker to store **("Reject-Proposal")** – Indicating a rejection message for a proposed service.
- The ProvBroker to check for service's result **("Inform").**
- The ProvBroker to store **(Inform-Done)** upon retrieving the service's result.

Any coordinated interaction between various entities relies on the use of a common communication language; the communication capability allows the entities to exchange messages with the other elements of the environment, including users, agents and objects. In order to perform their tasks these entities need to depend heavily on expressive communication with others not only to perform requests, but also to propagate their capabilities, advertise their own services, and explicitly delegate tasks or requests for assistance. The previous messages depict the possible actions that can take place in any given scenario. As discussed in Section 3.4.1, the messages exchanged in every protocol need to be complete in the sense that they fulfill the possible actions generated by the involved entities. Form the proceeding proposed protocols, the messages and their contents provide means for the various participants to coordinate their behavior and collaborate with each other to fulfill a specific task.

The coordinated interaction between the brokering entities and the domain entities relies on the agreed use of semantic and the intention of the transmitted messages. The previous classification of the supported messages defines the required messages to support the proposed privacy-based protocols. However, one of the proposed directions for future work is to use formal methodologies to validate the integrity of the proposed messages such as, a stepwise methodology developed in LOTOS [23][10]. Following this methodology, the overall complexities are broken into serial sub-steps. Each step evaluates and takes a small amount of decisions in isolation.

3.6. Discussion

The main goal presented in this chapter was to develop a framework that can address the coordination and cooperation challenges encountered in designing cooperative distributed systems with special attention to capability-based coordination as brokering service.

The framework provides a building block that can be used in conjunction with other specifications and application-specific protocols to accommodate a wide variety of protocols related to the operation of CDS applications.

In many of the reviewed work, such as in [63], [65] [67] [72] [64], the approaches have viewed coordination as a problem in different application domains. In contrast, our view clearly distinguishes the coordination as a solution for the capability interdependency problem.

The broker and the matchmaker based approaches have proposed two interaction patterns, one for the broker entity and for the matchmaker. Only one broker entity governs and directs the communication between the requester in any proposed interaction. The matchmaker suggests set of possible capable providers to accomplish required functionality (functional assignment problem which is encountered in two-layered client/server architecture of information systems, in which all functions had to be assigned either to the server or to the client).

In our proposed model, an appropriate interaction pattern is proposed or each privacy degree. Two sets of agents; one set is geared towards entities that play the role of requesters; the second to serve entities that play the role of providers. The intra-brokering interaction comprises sixteen possible combinations that can take place in supporting specific service requests. The model does not require an explicit initial privacy attributes that need to be known in priori in order to support complete privacy selection (hiding privacy attributes) that might be needed by service requesters and providers. In other words, requesters and providers will have the possibility to hide their privacy attributes from the whole environment including the relevant brokering entity and still be able to solicit help, collaborate and provide services to fulfill a particular request.

The proposed approach treats privacy as a design issue for brokering services. The work presented in [20] assumes that the degree of privacy is protected only at the initial state of the system, and considers whenever the entities come into direct contact; it is possible for one entity to learn the identity or the capabilities of the other. In another approaches [36][44][88][51], it is presumed that capabilities and preferences come to be known by all participants in the society, which leads to a chaotic environment where agents might violate any privacy requirements. By contrast, our proposed model define appropriate interaction protocols that allow requesters and providers to participate and solicit help without having to reveal their identities or requests to any entity within the community, including brokering entities assigned to provide such help.

Some approaches have proposed privacy patterns for supporting users' personal protection [37][79] [81] in terms of revealing less information about themselves, and in acquiring more information from the party with whom they are communicating before committing to any service access or delivery. However, the patterns do not provide structured mechanisms for the coordination and focus only on preserving the user anonymity based on cryptographic and anonymity techniques. The patterns focus on a single service environment and provide solutions for requesters to hide their identities and requests from providers but require the revealing of the identity-related information to a third trusted party. Furthermore, the patterns do not address any privacy concern that might be needed by service providers.

The brokering specifications are modeled formally as an IO automaton which describes the essential behavior of the different interaction protocols that are needed to support any privacy requirements in CDS. Using the IOA model provides a suitable structure for formalizing the proposed interaction protocols and which is important for building privacy-based coordination solutions in CDS environments.

3.7. Summary

The chapter described a privacy–based interaction protocols that allow different domain entities in an open environment to transparently and securely request and/or provide services. The proposed patterns permit requesters and providers to automate their privacy and accordingly select the appropriate privacy degrees that suit their desire. Each protocol is described using a mathematical state-machine model (Input/Output Automata-IOA). Each IOA is represented by both formal semantics and graphical notations using UML Sequence diagrams to exhibit the behavior of the entities (requesters, brokers and providers). In open dynamic environments, entities need to ubiquitously interact with each other, be able to self-manage at run-time as well as increase their degree of autonomy and responsibility. The chapter defined the patterns that can be represented at a higher level of abstraction and hence compose the privacy-based brokering protocols.

Chapter 4

DESIGN AND IMPLEMENTATION

This chapter provides a detailed design of an agent-oriented privacy-based brokering for CDS, based on Coordinated Intelligent Rational Agent (CIR). The chapter also presents as a proof-of-concept prototype for information-gathering capabilities in healthcare environments.

4.1. Modelling Cooperative Distributed Systems

It is clear that the development of coordination solutions in distributed open environments requires a new design paradigm, improved integration architectures, and services. The architecture must describe the organization and the interconnection among the software entities. In this architecture, the environment can be envisioned as a cooperative distributed system (CDS) comprised of a collection of economically motivated software agents that interact competitively or cooperatively, find and process information, and disseminate it to humans and other agents. It also enables common services that facilitate the coordination and the cooperation activities amongst various domain entities and support ad hoc and automated configurations.

Our proposed SOSDA framework provides the abstraction to support domain entities and applications independent of any specific technology. In this framework, a CDS is conceptualized as a dynamic community of agent and non-agent entities that contribute with different services. Based on the above view, an agent might play different roles and be able to coordinate cooperatively or competitively with other agents, including humans. Therefore, within the SOSDA architecture, the CDS entities are mapped as follows:

- *Service Requester:* is a domain specific entity that can interact with the environment and request services.

- *Service Provider:* a domain entity that provide application-specific services.
- *Brokering entity***:** is an agent that provides common SOSDA coordination services and facilities for the generic cooperative distributed systems environment.

4.2. Agent-Based Brokering Services for SOSDA

The dynamic nature of the entities participating in different brokering scenarios requires that they be able to change their configuration according to their roles. The challenge here is how to adopt a technology that provides means and mechanisms by which these entities would be able to interact with each other and determine an appropriate privacy degree. Clearly and as previously analyzed, such interaction is characterized by the non-determinism aspect and the dynamic nature of the environment where these entities exist and operate. These requirements could not be met using traditional ways of manually configuring software.

We strongly believe that agent-orientation is an appropriate design paradigm for providing coordination services and mechanisms in such settings. Indeed, such a paradigm is essential to modeling open, distributed, and heterogeneous environments in which an agent should be able to operate as part of a community of cooperative distributed systems environments including human users.

A key aspect of agent-orientation is the ability to design artifacts that are able to perceive, reason, interact and act in a coordinated fashion. We define an agent as an individual collection of primitive components that provide a focused and cohesive set of capabilities. We focus on the notion of Agenthood as a metaphorical conceptualization tool at a high level of abstraction (knowledge level) that captures, supports and implements features that are useful for distributed computation in open environments. These features include cooperation, coordination, interaction, as well as intelligence, adaptability as well as economic and logical rationality.

4.3. Example: Brokering for SOSDA Healthcare CDS

Many initiatives and programs have been established to promote the development of less costly and more effective healthcare networks and systems at national and international scale. The objectives of these healthcare networks is to improve diagnosis through online access to medical specialists, on-line reservation of analysis and hospital services by practitioners extended on wide global scale, transplant matching, etc. A complete electronic medical patient case file, which might be shared between specialists and can be interchanged between hospitals and with GPs, will be crucial in diagnosing diseases correctly, avoiding duplicative risky and expensive tests, and developing effective treatment plans.

However, medical patient case files may contain some sensitive information about critical and vital topics such as abortions, emotional and psychiatric care, sexual behaviors, sexually transmitted diseases, HIV status, and genetic predisposition to diseases. Privacy and the confidentiality of medical records have to be especially safeguarded. Without broad trust in medical privacy, patients may avoid crucial health care provision.

Healthcare professionals and care-providers prefer to have the ability of controlling the collection, retention and distribution of information about themselves. On the other hand, healthcare service providers need to effectively manage and prevent any abuse of the information or service they provide in addition to the ability of protecting their identities. An important feature of the various healthcare sectors is that they share similar problems and are faced with challenges that can be characterized as follows:

- In open distributed healthcare environments, it is no longer practical to expect healthcare clinicians, staff, care providers and patients to determine and keep track of the information and services relevant to his/her requests and demands. For example a patient shall be ubiquitously able to access his/her medical record from anywhere at any time or may request medical services offered by available healthcare centers in a particular city without being aware of the distributed sources and irrespective of their locations. In addition, an application should be able to manage distributed data in a unified fashion. This involves several tasks, such as maintaining consistency and data integrity among distributed data sources, and auditing access.
- The distributed nature of the knowledge among multiple healthcare locations may require collaboration for in formation gathering. For example, each unit in a hospital keeps its own information about patients' records.
- The solution of specific medical problem includes complex activities and requires collaborative effort of different individuals who posses distinct roles and skills. For

example, the provision of care to hospitalized patients involves various procedures and requires the coordinated interaction amongst various staff and medical members. It is essential that all the involved medical staff and professionals must coordinate their activities in a manner that will guarantee the best appropriate treatment that can be offered to the patient.

Healthcare professionals and care-providers prefer to have the ability of controlling the collection, retention and distribution of information about themselves. A recent survey shows that 67% of the American national respondents are concerned about the privacy of their personal medical records, 52% fear that their health insurance information might be used by employers to limit job opportunities while only 30% are willing to share their personal health information with health professionals not directly involved in their case. As few as 27% respondents are willing to share their medical records with drug companies [90].

To explore such issues, distributed healthcare systems need to have an access to a service that can enable collaboration between different healthcare service requesters and providers. The proposed brokering model controls coordination activities among various healthcare service requesters and providers. Healthcare personnel get access to different services managed by various providers without having to be aware of the location, identities, access mechanisms, or the contents of these services. The model provides seamlessly integrated healthcare environment and presents additional privacy opportunities to patients, visitors, medical staff and vendors.

4.4. The Coordinated Intelligent Rational Agent (CIR-Agent) Model

The representative agents of domain and brokering entities within the context of SOSDA-based CDS are built on the foundation of CIR-agent architecture with focuses on utilizing the model to capture the participants' individual behavior towards achieving a desirable goal while maintaining a required privacy degree.

The CIR-Agent is an individual collection of primitive components that provide a focused and cohesive set of capabilities. The basic components include problem-solving, interaction, and communication components, as shown in Figure 27(b). A particular arrangement (or interconnection) of components is required to constitute an agent. This arrangement reflects the pattern of the agent's mental state as related to its reasoning about achieving a goal. However, no specific assumptions need to be made on the detailed design of the agent components. Therefore, the internal structure of the components can be designed and implemented using object oriented or another technology, provided that the developer conceptualizes the specified architecture of the agent as described in Figure 27.

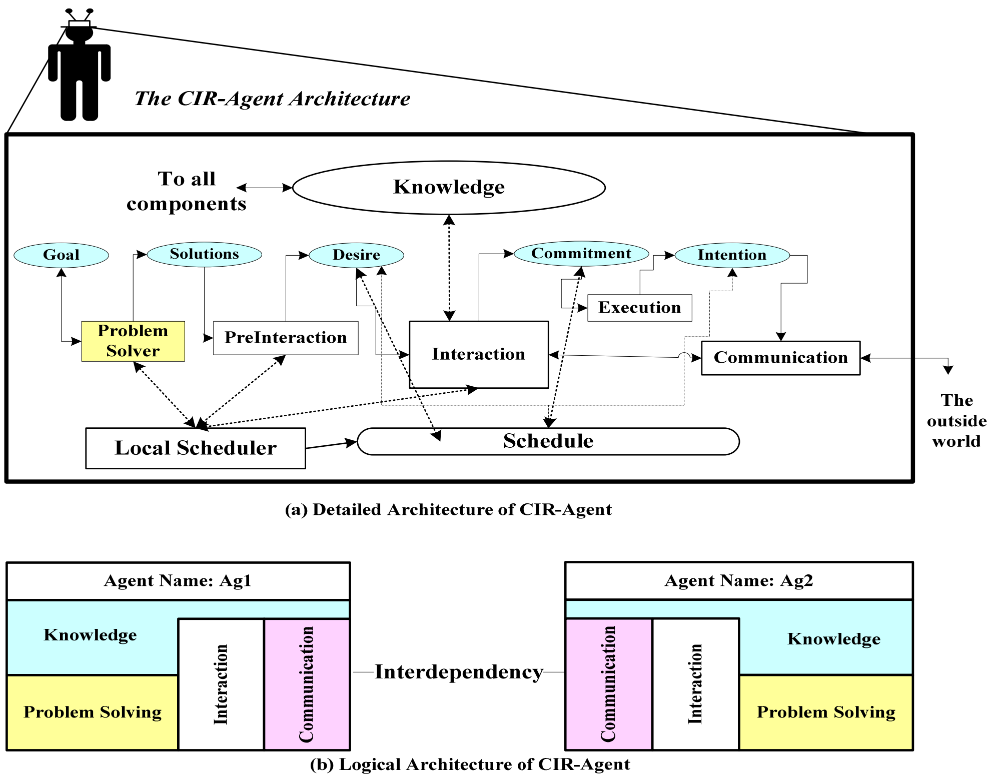

Figure 27: The CIR Agent's Architecture

Basically, each agent consists of knowledge and capability components. Each of which is tailored according to the agent’s specific role.

The agent's knowledge contains the information about the environment and the expected world. The knowledge includes the agent self-model, other agents' model, goals that need to be satisfied, possible solutions generated to satisfy each goal, and the local history of the world that consists of all possible local views for an agent at any given time. The agent’s knowledge also includes the agent's desires, commitments and intentions toward achieving each goal.

The capability package includes the reasoning component, the domain actions component which contains the possible set of domain actions that when executed, the state of the world will be changed and the communication component where the agent sends and receives messages to and from other agents and the outside world.

The problem solver component represents the particular role of the agent and provides the agent with the capability of reasoning about its knowledge to generate appropriate solutions directed to satisfy its goal.

During the interaction processes, the agents engage with each other while resolving problems that are related to different types of interdependencies. The coordination mechanisms are meant to reduce and resolve the problems associated with interdependencies. Interdependencies are goal-relevant interrelationships between actions performed by various agents.

As argued in [32], the agent's interaction module identifies the type of interdependencies that may exist in a particular domain. Consequently, agents select an appropriate interaction device[5] that is suitable to resolve a particular interdependency. These devices are categorized as follows:

- Contract-based, includes the *assignment* device;
- Negotiation-based, includes *resource scheduling*, *conflict resolution*, *synchronization*, and *redundancy avoidance* devices.

Within the context of brokering, the interdependency problem is classified as capability interdependency and the interaction device is the "*assignment*". The basic characteristics of the assignment device are problem specifications, evaluation parameters, and the sub-processes. The problem specifications might include, for example, the request, the desired-satisfying time, and the expiration time.

A collection of basic components comprises the structure of the agent model and represents its capabilities. The agents' architectures are based on the CIR-Agent model as

[5] Interaction device is an agent's component by which it interacts with the other elements of the environment through a communication device. A device is a piece or a component with software characteristics that is designed to service a special purpose or perform a special function.

shown in Figure 28. A brokering session mainly recognizes two types of agents, namely, domain agent (Requester or Provider) and brokering agent (ReqBroker or ProvBroker). The architecture of each agent type is described in details below.

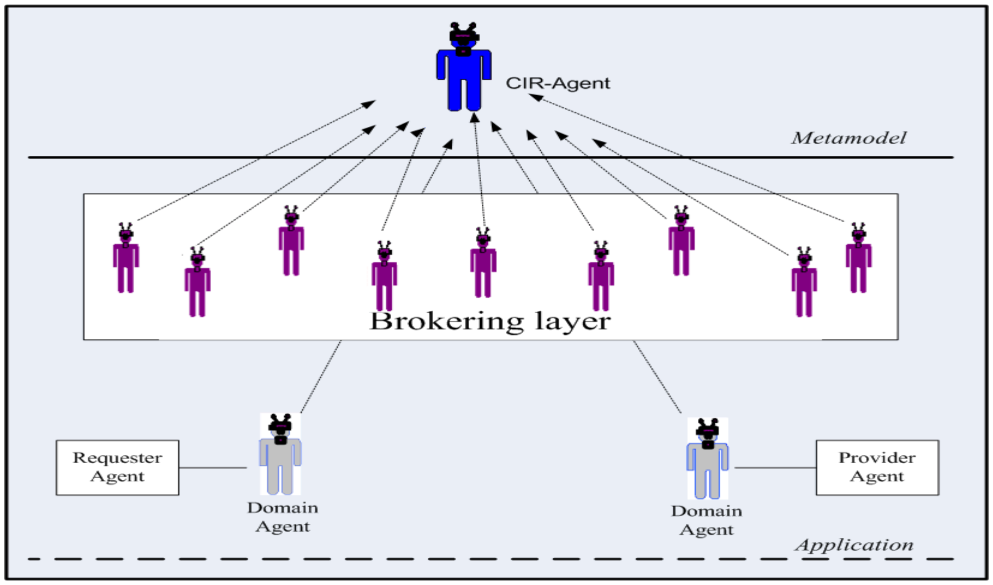

Figure 28: The Overall System Model

4.5. The Domain Agent: Service Providers and Requesters

Service providers and requesters are modeled as domain agents as shown in Figure 29. The requester agent can participate with various privacy degrees and request services from the brokering layer. A requester delegates the service's request(s) to the relevant brokering agent according to the interaction protocol the selected privacy degree. The domain agent possesses *knowledge* and *capability*. The knowledge includes the model of the brokering agents in terms of the supported privacy degree, self model and the local history. The capability is categorized into three components: *reasoning* that includes *problem-solving* and *coordination, communication* and a *set of domain actions*.

A domain agent playing the role of a service provider can select the appropriate privacy degree and thus participate on providing the capability that meets the needs of another domain entity. The problem solver the domain agent hiding any of the privacy attributes encompasses the accessing of different storage repositories. For example, the problem solver of a requester includes functionalities related to formulating service requests, check for available service offerings and access various storage repositories to store requests or to retrieve service results. On the other hand, the problem solver of a provider hiding its identity and capability attributes consists of modules related to accessing storage repositories to check for stored service requests that might be fulfilled and hence participating in storing service proposals and service's results.

The coordination component of a requester comprises the interaction device which entails soliciting service from the relevant ReqBroker agent. The interaction device of the provider agent manages the coordination activities which involve proposing services to specific CFP messages and engage in bidding processes.

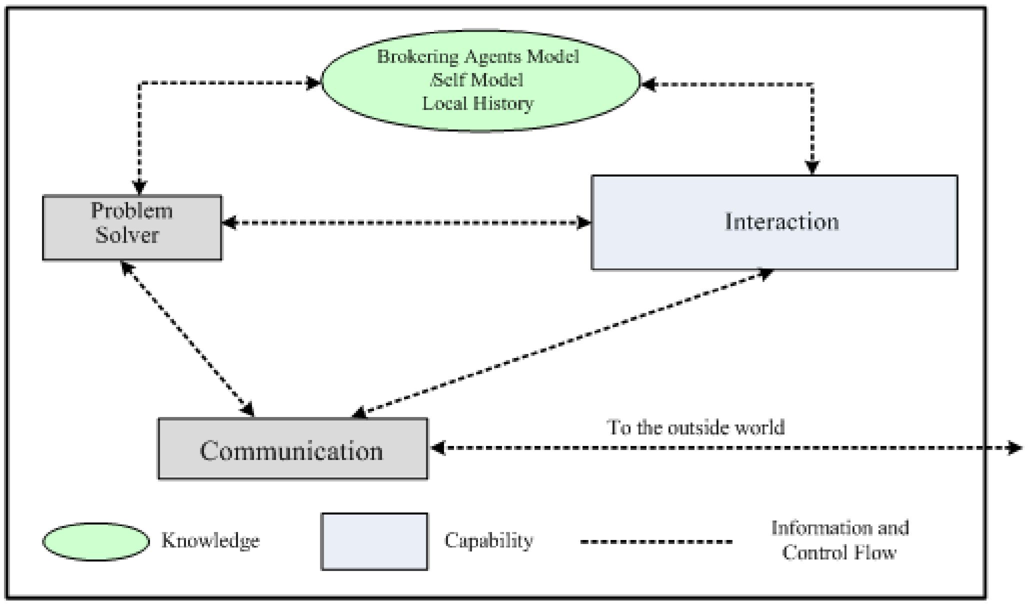

Figure 29: The Domain Agent Architecture

4.6. The Brokering Agents: ReqBrokers and ProvBrokers

A brokering agent is composed of two components namely, the *knowledge* and *capability*. The knowledge component contains the information in the agent's memory about the environment and the expected world. As shown in Figure 30, this includes the agent self-model, models of the domain agents in terms of their roles (requester/provider) and/or capabilities and the local history of the world. The knowledge includes all possible local views for an agent at any given time (such as the knowledge of physical repositories available services requests, services offerings and service results).

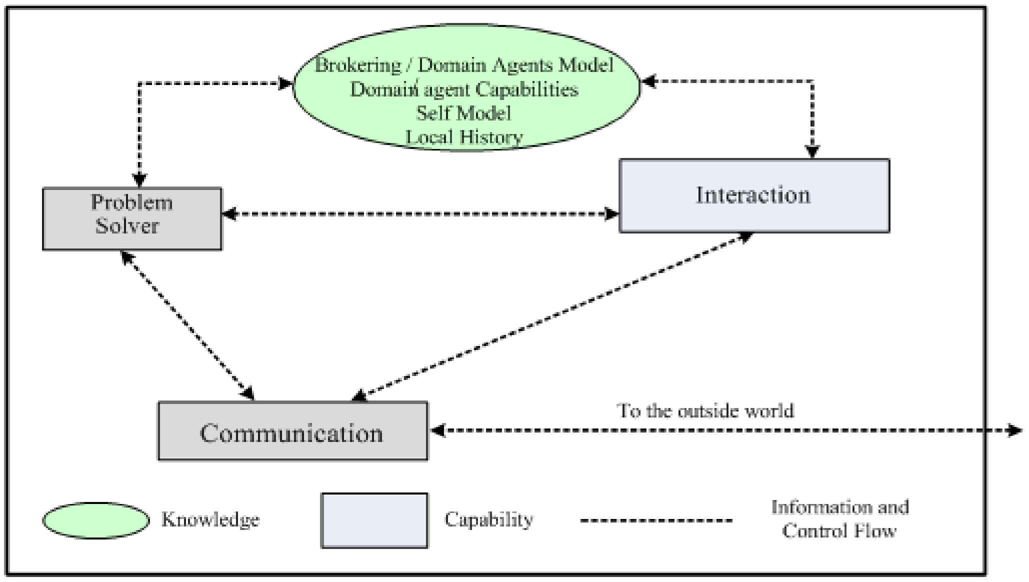

Figure 30: The Brokering Agent Architecture

4.6.1. The ReqBroker Agent

The problem solver component varies form one brokering agent to another. The ReqBroker's problem solver component includes: accessing various storage repositories, locate and identify services' requests, deliver and store services' results. The interaction component comprises the following activities: (1) Preparing the "CFP" message that formulates the "announcement" to be sent out to the ProvBrokers, (2) Collecting service proposals and (3) evaluating these proposals against certain criteria (for example parameters identified in the service's request).

4.6.1.1. The ReqBroker Interaction Device: Assignment

The main function of the assignment device is to resolve problems associated with capability and decomposition interdependencies. The basic characteristics of the assignment device are problem specifications and evaluation parameters. With reducing complexity in achieving a goal as the agent's main objective, a solution can be selected based for example on the goal quality.

The implementation technique of the assignment device is based on the soliciting approach as in the case of the contract-net approach which depends on (1) the modeling approach for other agent's capabilities, and (2) the solicitation approach for the local schedule and workload of other agents. To achieve a high degree of parallelism in the assignment device, the implementation consists of the following processes:

1. **Call for Proposals:** The initiating ReqBroker agent (or the manager) informs all the other potential ProvBrokers agents (or contractors) of the problem specification by an announcement. The problem specification might include the goal, and the desired satisfying time for example. The ProvBrokers initialize generates a CFP message to the relevant service providers. A focusing strategy might be used by the ProvBrokers to identify the set of potential contractors (service providers) based on their capabilities (in scenarios related to providers revealing their capabilities). At a certain time, ProvBrokers representing their interested relevant providers send "Propose" message to the ReqBroker agent indicating the start of the bidding process. A focusing strategy might be used by the ProvBrokers to identify the set of potential contractors (service providers) based on their capabilities (in scenarios related to providers revealing their capabilities).
2. **Evaluate:** At a certain time, ProvBrokers representing their interested relevant providers send "Propose" messages to the ReqBroker agent indicating the start of the bidding process. When the service satisfying deadline reaches, the ReqBroker ceases to accept any new messages related to either request's inquiries or new service offers. Based on the evaluation parameters, the ReqBroker evaluates submitted proposals and accordingly selects the best bid.
3. **Award/Reject**: the process allows the ReqBroker to issue an award/reject message to the potential ProvBrokers. For the selected (winning) ProvBrokers, a contract form is created, an award (Accept-Proposal) message is sent to the corresponding ProvBrokers and a reject (Reject-Proposal) messages is sent to the non-wining.

The reasoning components of the ReqBrokers vary according to the privacy degree they support. As previously shown, the interaction protocols for the various brokering scenarios have illustrated repeated patterns and a behavior that can be represented at a higher level of abstraction. Utilizing the privacy-based protocols defined in Section 3.5.1 the reasoning components of the various ReqBrokers will be represented as follows:

4.6.1.2. The Negotiator Design

Requesters are required to reveal their privacy attributes to the related Negotiator. As shown in Figure 31, the interaction component includes the following protocols: (1) the contracting protocol that exhibits all the interaction with the ProvBrokers and (2) the direct mode of the service soliciting protocol which abstracts all the interaction activities with the service provider.

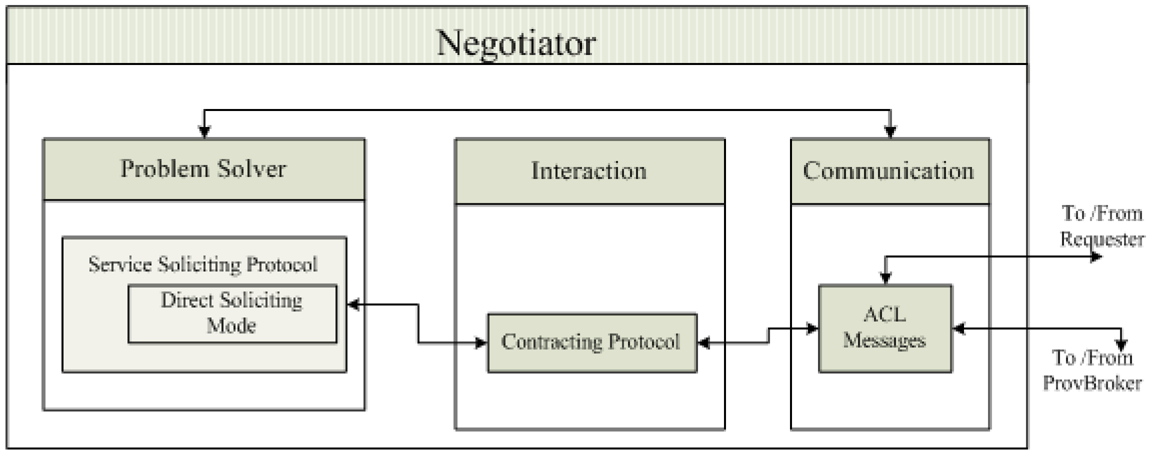

Figure 31: Architecture of the Negotiator's Reasoning Component

4.6.1.3. The Mediator Design

Requestors are permitted to have an access to special repositories by which they would be able to post their required service requests without having to reveal their identity to any other entity in the environment. The Mediator's problem solver functionality is solely to access these repositories to: query about available service requests, store service's result into the result location repository and to check for result's receipt acknowledgments. The interaction component utilizes the contracting protocol as shown in Figure 32.

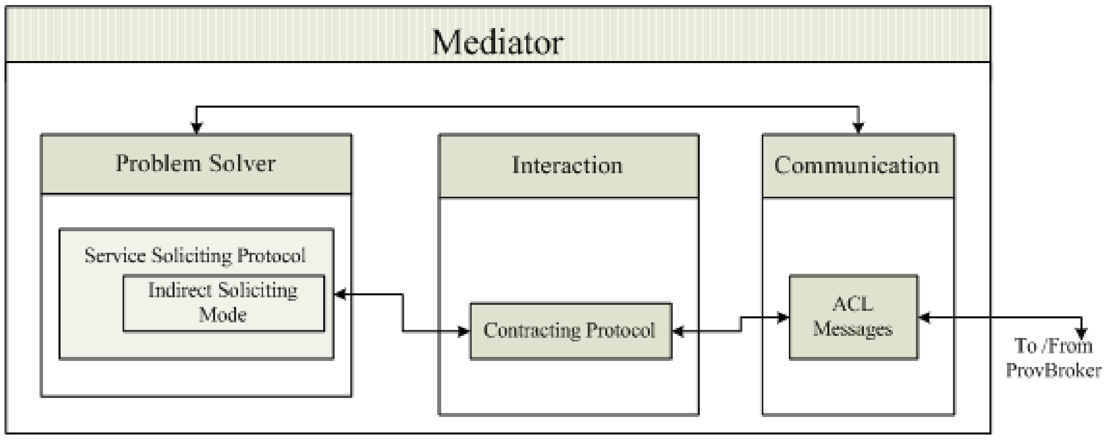

Figure 32: Architecture of the Mediator's Reasoning Component

4.6.1.4. The Advertiser Design

The protocol preserves the privacy attribute of the requester (hidden request.). Upon deciding on particular offerings, the Advertiser's interaction component incorporates the contracting protocol. As shown in Figure 33, the problem solver component includes the service soliciting protocol (indirect mode) while the interaction component includes the contracting protocol.

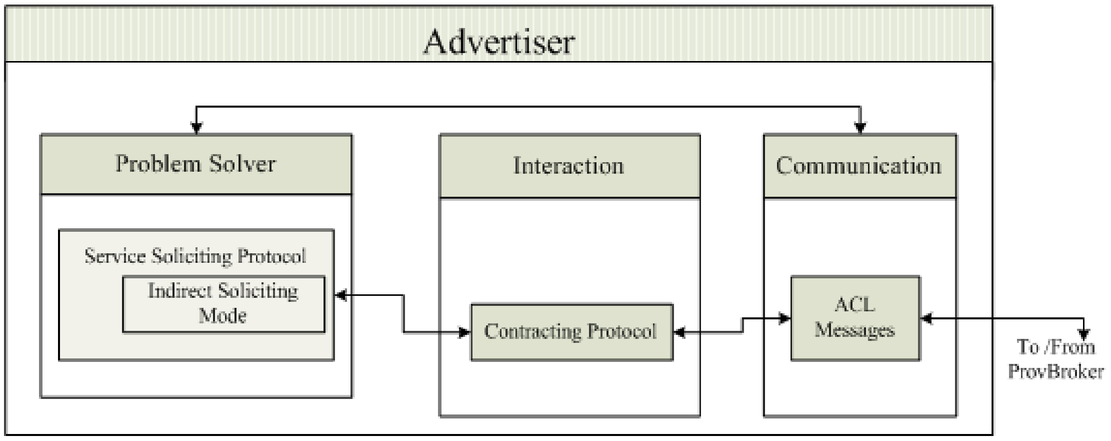

Figure 33: Architecture of the Advertiser's Reasoning Component

4.6.1.5. The Bulletinboard Design

The Bulletinboard's problem solver functionalities are limited to accessing the various repositories (either to check for service requests, store service results or to check for receipts acknowledgment). These functionalities are performed by utilizing the indirect mode of the service soliciting protocol as shown in Figure 34.

Similarly, the interactions with the ProvBrokers are governed by the contracting protocol which composes the coordination component.

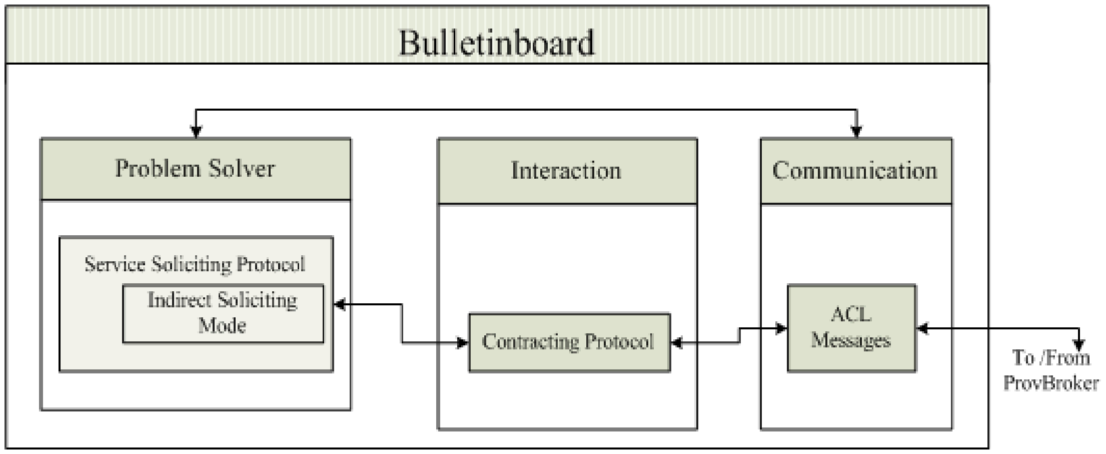

Figure 34: Architecture of the Bulletinboard's Reasoning Component

4.6.2. The ProvBroker Agent

The ReqBroker agent sends (or store) a "CFP" message to service providers. The ProvBroker carries out the interaction with the ReqBrokers and accordingly reports the outcome of the interaction to the participating service provider. The ProvBroker's architecture varies according to the supported privacy degrees.

4.6.2.1. The ProvBroker Interaction Device: Assignment

Similarly, the implementation technique of the assignment device of the ProvBroker is based on the soliciting approach as in the case of the contract-net approach. The implementation consists of the following processes:

1. **Call for Proposals:** The ProvBrokers initialize a CFP message to the relevant service providers. A focusing strategy might be used by the ProvBrokers to identify the set of potential contractors (service providers) based on their capabilities (in scenarios related to providers revealing their capabilities).
2. **Propose:** At a certain time, ProvBrokers representing their interested relevant providers send "Propose" messages to the ReqBroker agent indicating the start of the

bidding process. A focusing strategy might be used by the ProvBrokers to identify the set of potential contractors (service providers) based on their capabilities (in scenarios related to providers revealing their capabilities).

3. **Winning/Rejection**: By receiving the award message, the ProvBroker creates the winning process and accordingly informs the winning provider (either by sending an acceptance message to the providers or by storing the Accept-Proposal message into a repository). Note that process initiates a commitment state which indicates the engagement of the ProvBroker into a contract. Alternatively, upon the receipt of a rejection message (Reject-Proposal), the process allows the ProvBrokers to notify the non-winning service provider and consequently destroys all the information relevant to the rejected service proposals. The reasoning components for the ProvBrokers as described in the following sections.

4.6.2.2. The Arbitrator Design

The Arbitrator acts on behalf of the service provider to participate in proposing services for a particular request. All interactions with ReqBrokers entail the exposure of only the identity of the engaged Arbitrator. As shown in Figure 35, the interaction component of the Arbitrator includes the contracting protocol and the service delivery protocol (direct mode).

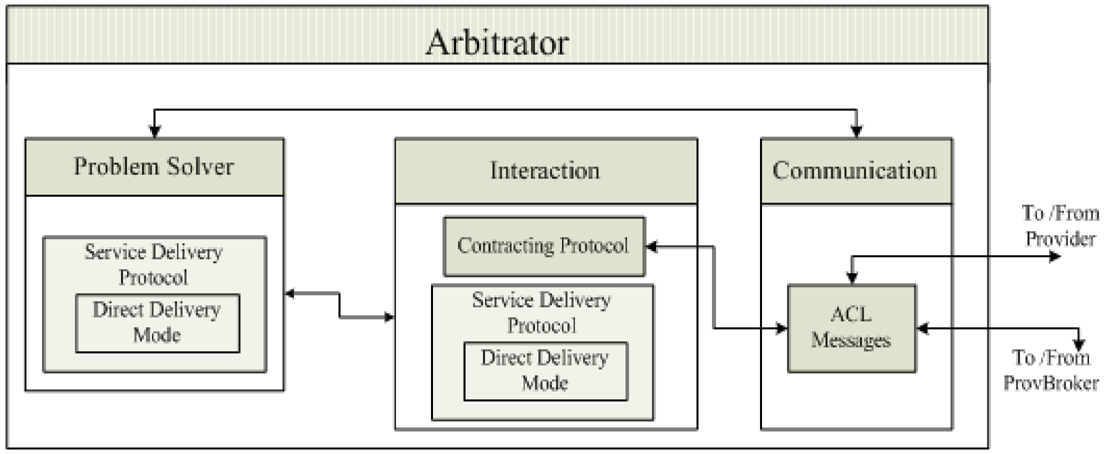

Figure 35: Architecture of the Arbitrator's reasoning Component

4.6.2.3. The Broadcaster Design

Providers are allowed to access repositories to check for requests that might be of an interest without having to reveal their identity to any other entity in the environment. All the interaction with provider with this privacy degree is accomplished by the problem solver component which includes the indirect mode of the service delivery protocol. Additionally, the problem solver includes functionalities related to link a specific CFP message to a potential service proposal and to map the service's result to that particular request. As shown in Figure 36, the Broadcaster's interaction component uses the contracting protocol.

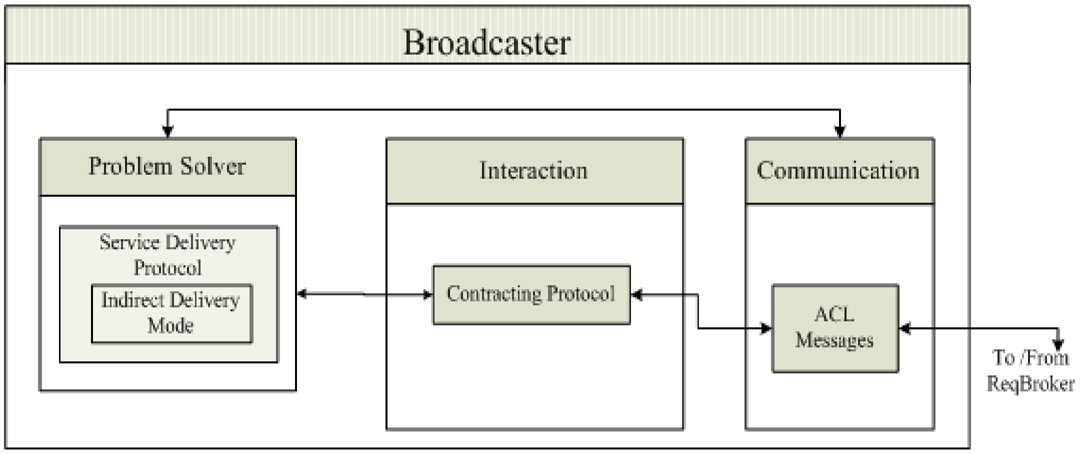

Figure 36: Architecture of the Broadcaster's Reasoning Component

4.6.2.4. The Recommender Design

The Recommender's interaction enables sending every received CFP service request (from ReqBrokers) to the provider with unknown capabilities. As shown in Figure 37, the interaction component includes the contracting protocol while the problem solver comprises the indirect mode of the service delivery protocol

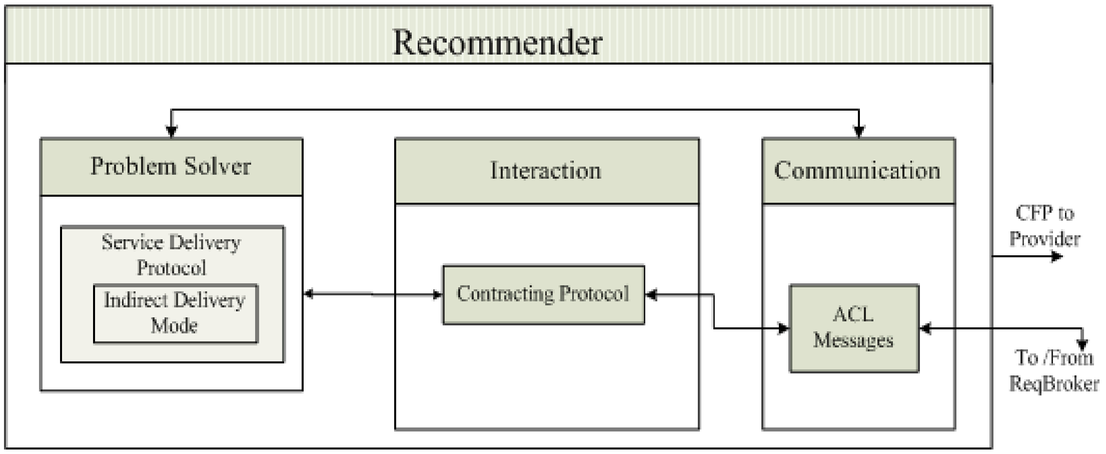

Figure 37: Architecture of the Recommender's Reasoning Component

4.6.2.5. The Anonymizer Design

The Anonymizer informs (indirectly) those providers wishing to hide their privacy attributes of any service request that might be fulfilled by their capabilities. All service requests and service offerings and services results are stored into special storage repositories (service request and result locations that are accessed consecutively by both the Anonymizer and the provider). In order to achieve such functionalities, the problem solver component utilizes the indirect mode of the service delivery protocol as shown in Figure 37.

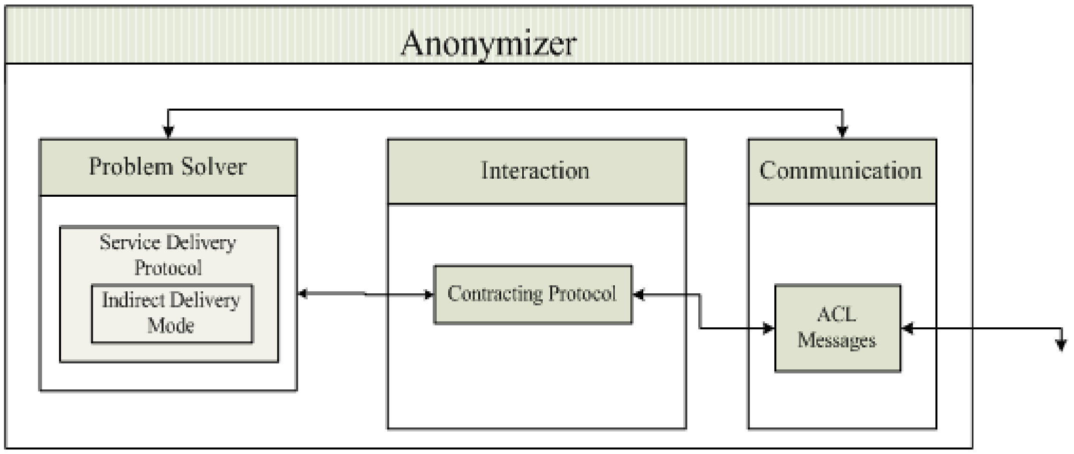

Figure 38: Architecture of the Anonymizer Reasoning Component

4.7. Supporting Services

In open environments, entities can join/disjoin unpredictably, and thus the system should have means to handle joining/removal of entities both (brokering and others) at the run-time without losing the system's integrity. The proposed system assumes the availability of some supporting service such as:

- Capability and Service Description Service: In general, the brokering entities should be able (in real time) to parse, validate, understand and respectively process capability and service descriptions it receives. To enable the dynamic discovery of services, a mechanism is required to describe the capability aspects of services, such as the functional description of a service, the conditions and the constraints of the service and the nature of the results. A matching mechanism allows brokering entities to understand, automatically process requests and accordingly determine the capabilities of service providers that are most appropriate for a given request.
- Management of data and knowledge: In many proposed protocols, domain and brokering entities need to have access to various physical repositories for efficient storage of service requests, service offerings and service results. The corresponding databases, repositories and know ledge base of a the brokering layer need to be inspected by other agents and users for different purposes such as, the specification of appropriate requests or searching for applicable services according to given valid access restrictions and security policies. However, the systems shall provide interface services to the domain entities to let them access such repositories. These interface services have to be restricted according to given security policies and requirements. One of the possible solution is to use public key encryption techniques
- Provision of registration and naming services: The registration and naming service allows building up a knowledge base of the environment that can be utilized to facilitate locating and identifying the relevant existing service's sources and their contents for serving a specific request. It is crucial to be able to identify the subset of relevant information at a source and to combine partially relevant

information across different sources; this requires the process of identification and retrieval of a subset of required service at any source. It is clear that in such environment, different sources would provide relevant information to a different extent. It is assumed that each entity (brokering or domain) has to register with this service. In the design of these entities (agent-based), the knowledge component shall be able to have mechanisms of dynamic updates. It is assumed that the registration service shall have means of notifications and will be able to regularly monitor the availability of brokering and domain entities as assurance of presence. Requesters and providers will have handles (interfaces) to interact with the brokering entity. There might be situations where a brokering entity fails after registering itself for a particular role or a provider disjoin after issuing service advertisement. The brokering service maintains and stores associations between the services provided by providers and the descriptions of such services (metadata). These associations enable brokers to increase the likelihood of an accurate service discovery with greater precision. This implies the ability to allow adding or removal services at runtime and hence updating service repositories accordingly.

4.8. Implementation Example: Agent-Oriented Privacy Brokering for SOSDA Healthcare CDS

In this section we show an example of our proposed model applied to healthcare environments to support information-gathering capabilities utilizing the specifications defined in [71].

Healthcare services can be modeled and implemented as CDS. The healthcare is viewed as a collection of autonomous units that can act independently and cooperate in providing services and synergize medical data according to mutual interests. The infrastructure of the participants of healthcare can handle only the internal administrative and clinical processes. The model provides querying ability and coordination activities that enhance the overall connectivity of distributed, autonomous, and possibly heterogeneous information sources (databases) of different healthcare providers and hospitals.

The following describes a scenario for a requester hiding its identity and three service providers; one is revealing privacy attributes, the second is hiding its identity while third is hiding its own privacy attributes (identities and capabilities). Consider three online information providers, *E-VirtualMedInfo Inc*., *E-VirtualDiagnosis Inc.*, and *FutureDocAssistant Inc*[6]. , each of them provide medical information, healthcare guidelines and clinical diagnosis in various formats (online delivery, hard copies or access to online medical repositories). E-VirtualMedInfo Inc, is revealing its privacy attributes and supported by the Arbitrator agent, E-VirtualDiagnosis comprise diagnosis capabilities jointly derived by retired medical doctors and had selected hiding its identity for which the Broadcaster agent will be the dedicated ProvBroker, whereas FutureDocAssistant, a company that provide various online samples of medical exams and virtual evaluation assessments decided to hide both the identity and capabilities and will be supported by the Anonymizer agent.

Alice, a fourth year medical student, is conducting a research on the most fatal diseases in Canada, the mortality death rates of each disease and the possible diagnosis and prevention procedures that would help a trainee-student in examining and diagnosing patients with such diseases. Deciding to hide her identity, Alice anonymously can request this information by posting the service request in special repository dedicated to such privacy degree. Note that, the Mediator will be the assigned brokering agent which acts on behalf of Alice to fulfill her requests. As shown in Figure 39, the protocol will be as follow:

[6] Names are fictitious

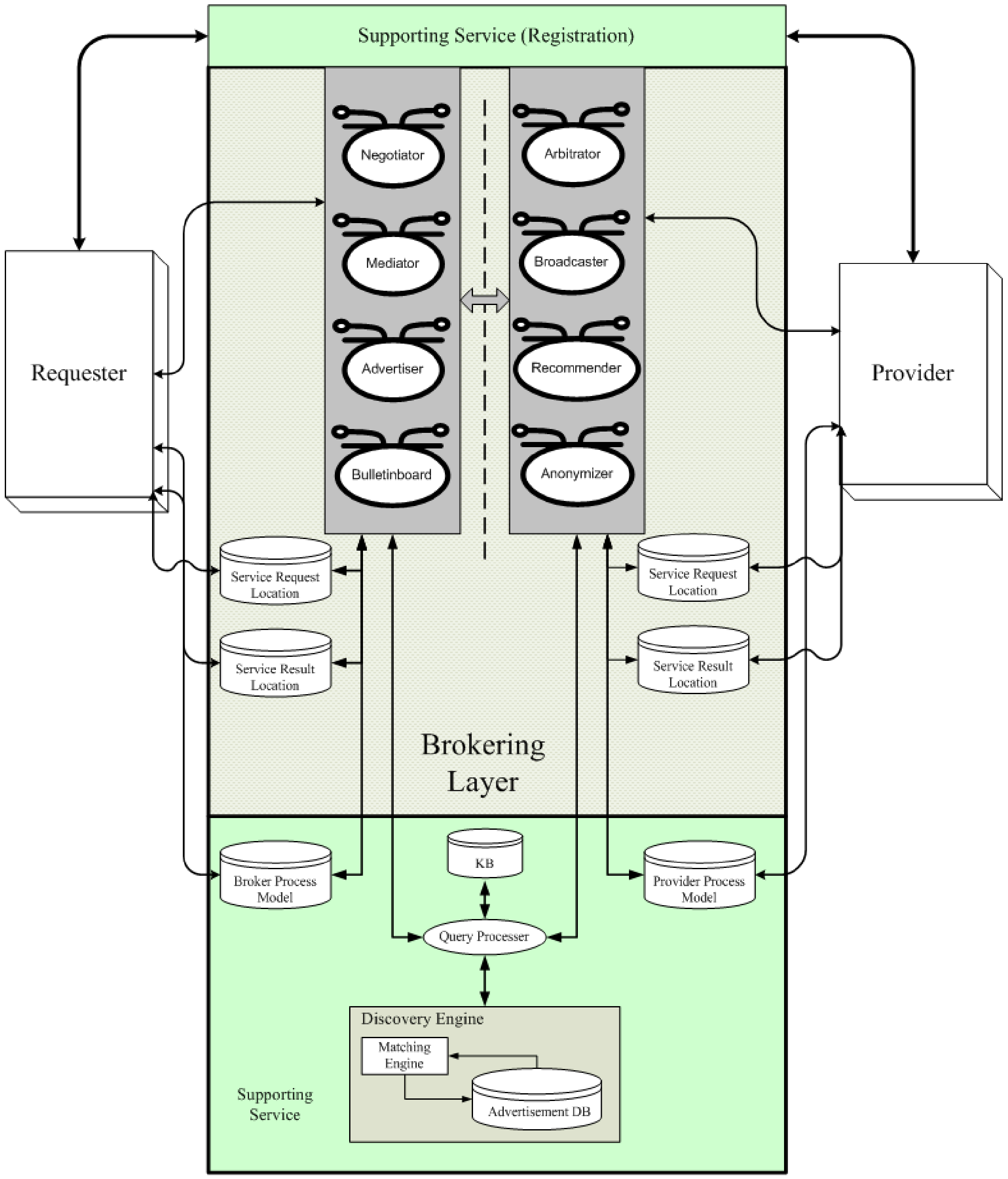

Figure 39: The Brokering Layer Architecture

1. Alice anonymously requests information by posting the request in a special repository dedicated to such privacy degree.
2. Alice's assigned Mediator retrieves the posted request, abstracts the required capabilities and constructs a service request.

3. The Mediator interacts with various ProvBrokers (including the Arbitrator, the Broadcaster and the Anonymizer) and consequently (acts as a manager) issues a call-for-proposals (CFP) to those ProvBrokers (act as potential contractors) informing them of the Alice's request specifications (note that Alice's identity is anonymous to each participant including its own supporting Mediator).
4. Note that for the E-VirtualDiagnosis and FutureDocAssistant Companies, the request is dispatched into a dedicated storage repository by their relevant ProvBroker (the Broadcaster and the Anonymizer). Every company (through its representing agent) determines the evaluation parameters (such as information quality, expiration time, and cost) and accordingly submits a bid along with the offer parameters to the relevant brokering agent. The Virtue-Info-Medic agent sends bids directly to the Arbitrator, while the E-VirtualDiagnosis and FutureDocAssistant corresponding agents store their bids into a special repository. It is to be noted that the Arbitrator might locate the applicable provider that fulfills the service request and hence focusing the request.
5. The Mediator receives those bids the Arbitrator, the Broadcaster and the Anonymizer, carries on the evaluation process and accordingly determines the most bid (or bids) that fulfill Alice's request. Assume that the winning provider is the FutureDocAssistant, therefore the Mediator sends an acceptance message to the Anonymizer, and in the mean time sends a rejection message to both the Arbitrator, the Broadcaster agents.
6. The Anonymizer informs the FutureDocAssistant agent about the acceptance of its offer by storing an acceptance message into the service repository.
7. Once the request is fulfilled, the FutureDocAssistant agent stores the result (required medical information) into service's result repository to be retrieved by the Anonymizer.
8. Upon retrieving the service's result, the Anonymizer stores a receipt acknowledgment informing the FutureDocAssistant agent about the receipt of the result and consequently delivers it back to the Mediator.
9. The Mediator stores the result into the result repository for which Alice will be able to retrieve it without having to reveal its own identity.

4.8.1. Implementation

A prototype of the proposed system has been implemented to support and provide information-gathering capabilities to different participants in healthcare environments where the accessibility of private information is a desirable feature to various categories of the healthcare personnel, patients, and clinicians.

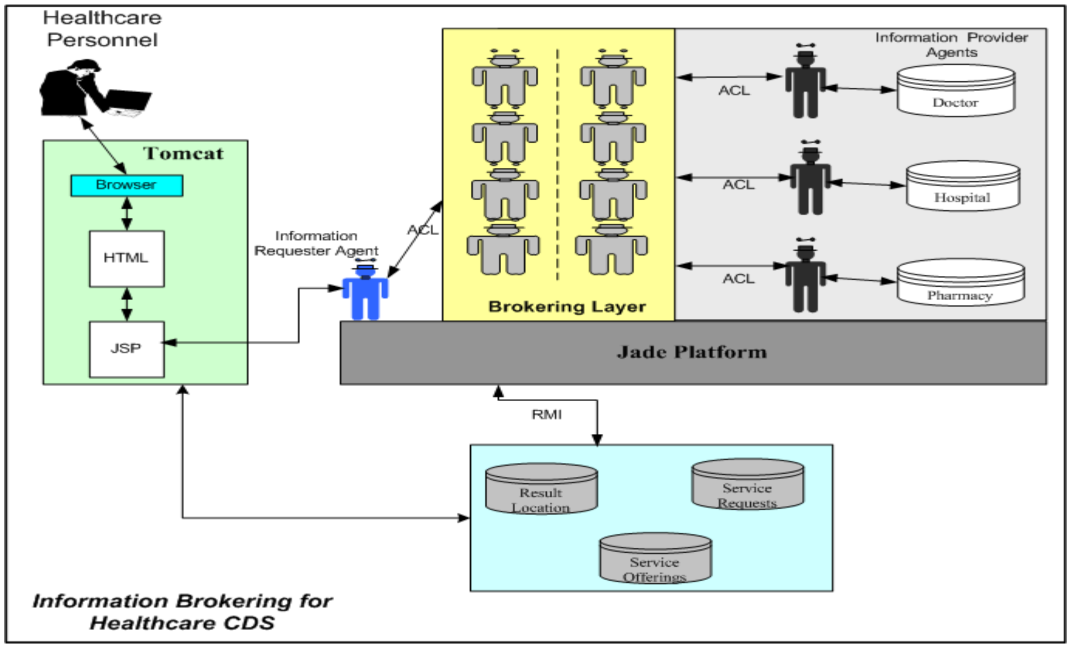

Figure 40: Information Brokering for Healthcare CDS

As shown in Figure 40, three databases represent various medical data for three distributed locations, each being managed by a dedicated agent that can play both roles of an information requester as well as a provider. A web interface is available for healthcare participants to select their desired privacy degree along with any capability they might posses (medical data, patient's diagnosis and treatment reports, Pharmaceutical data reports, etc.). Based on the privacy degree required by both the requester an information provider, dedicated brokering agents handle the interaction according to the relevant interaction protocols associated to the selected privacy degrees.

The implementation utilizes Java Web Services Development Pack (JWSDP) [42]and the JADE platform [41], which is a software framework to develop agent applications in compliance with the FIPA specifications for multi-agent systems. JADE supports a distributed environment of agent containers. They provide a run-time environment that is optimized to allow several agents to execute concurrently.

As described above, the architecture of agents is based on the CIR-Agent model as described in the following subsections. The role of the ReqBroker agents is to formulate the service request, sends it out to all ProvBroker agents and collects the service proposals (bids). At the end of the bidding-time, it evaluates the bids, determines the winner bidder-agent and notifies both the ProvBrokers of the outcome. As described before, the ReqBroker is a collection of knowledge and capabilities components. The knowledge component includes the agent's self-model, model of other agents, and the local history. The main capabilities of the CIR-ReqBroker agent include communication, reasoning and domain actions components.

The ReqBroker's problem-solver component contains a set of Jade behavior classes (*simpleBehaviors* and *cyclicBehaviors*) that represent the ReqBroker's specific tasks, agent's platform tasks such as registration with the directory facilitator (DF) service and to handle the incoming messages from both the requester and the ProvBrokers. The cyclic behavior class equips the ReqBroker agent with ability to check for service requests that have been stored by requesters hiding their privacy attributes.

The communication component is implemented as a set of classes that inherit the *jade.Core.Agent* and *jade.lang.acl.ACLMessage* existing classes of the jade platform. These classes provide means to construct, send and receive messages via several FIPA performatives such as REQUEST, INFORM, INFORM-DONE, QUERY-IF, etc. The communication component is equipped with an incoming message inbox, and message polling that can be both blocking and non-blocking, with an optional timeout.

The messages exchanged in the interaction protocols are implemented as per the structure defined as FIPA ACL Messages specifications [24]. Each message contains a set of one or more message elements. The elements vary according to the brokering scenario; the only element that is mandatory in all ACL messages is the *performative,* although most of the ACL messages will also contain a *sender*, *receiver* and *content* elements. For example in case of an entity of a hidden identity, the sender or a receiver element is not defined. The communication component utilizes the Jade class *ACLMessage* which implements an ACL message compliant to the FIPA ACL Message Structure Specification. Agents are able to get contents and set the content of a particular message by overriding the methods *setContent* and *getContent.*

As shown in Figure 41, the coordination component contains the Interaction-Assignment class that extends a JADE behavior class namely the *FipaContractNetBehaviour*. The *Initiator* class allows the ReqBroker/ProvBroker to play the role of the initiator of the protocol (Note that from the provider viewpoint, a ProvBroker is considered the initiator of the protocol) and implements the three methods that are called by *FipaContractNetInitiatorBehaviour*:

- *createCfpContent*, this method is called upon receiving or retrieving a service request. The method formulates the announcement of the required service and return the ("CFP") message content (contains conditions or constraints) to be sent to all the receivers (ProvBrokers).
- *handleProposeMessages*, to evaluate all the received proposals ("Propose") from the ProvBrokers and to return a vector of ACLMessage objects to be sent to the ProvBrokers in response to the proposals. By overriding this method, the ReqBroker will be able to evaluate received proposals and accordingly accept or reject service proposals. This method sends an acceptance message {"Accept-Proposal") to the wining bidder and a rejection message {"Reject-Proposal") to the non-winning bidders.
- *handleAllResponses*, to handle all received service results ("Inform") that are sent by the ProvBroker or by the service providers. A return message ("Inform-Done") is sent back to the sender acknowledging the receipt of the service's result.
- *handleRefuse,* this method allows the initiator to handle messages related to a decline message ("Refuse") for a specific service request.

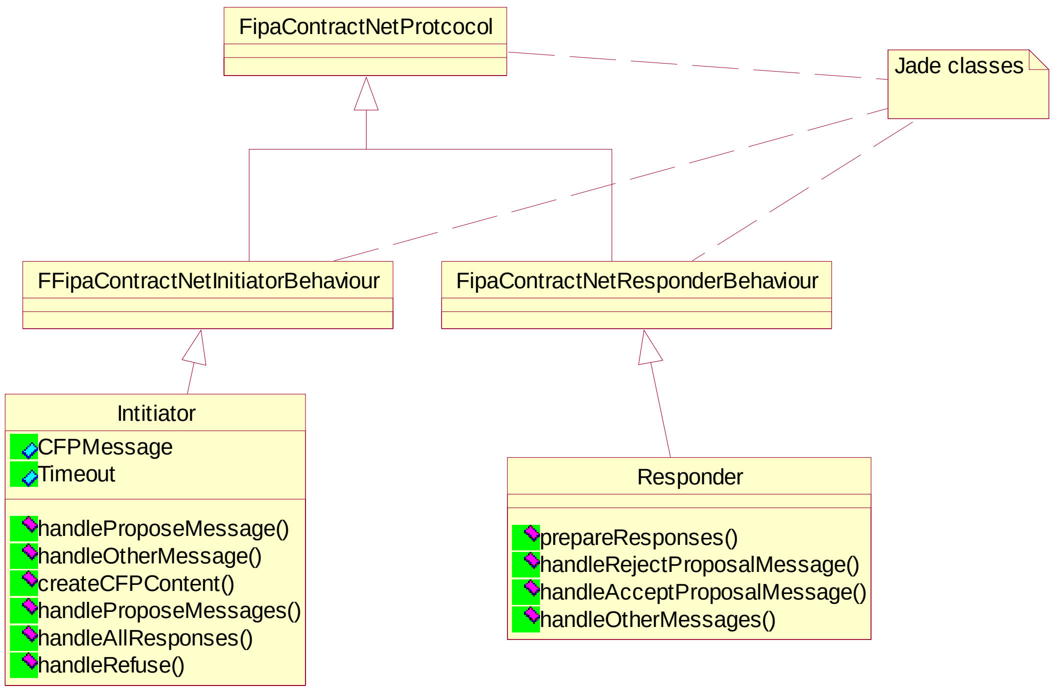

Figure 41: The Interaction Component (Assignment Device)

The coordination component of the ProvBroker (acting as a responder) contains the *Responder* class that extends the FIPA-compliant class *FipaContarctNetResponderBehavior*. This abstract behavior implements the interaction protocol from the point of view of a responder to a call for proposal (CFP) message. The class Implements the following methods:

- *prepareResponses*, the method returns an ACL message to be sent to the initiator in response to the CFP message which might have the content of Propose or Refuse. If null is returned, then the CFP is ignored and the behavior is reset and starts again waiting for CFP messages form ReqBrokers.

- *handleAcceptProposalMessage*, to evaluate the received ("Accept-Proposal") to be returned to the responder. Upon the invocation of this method, the responder is able to return the service's result ("Inform").

- *handleRejectProposalMessage*, allows the responder to handle messages related to a rejection of a particular service proposal ("Reject-Proposal"). After this method, the protocol is reset and it restarts again.

Additionally, in the contracting protocol, error messages (received messages that have other performatives) are handled through *handleOtherMessages* method.

4.9. Summary

Brokering services provide transparent access to a collection of distributed entities in a given domain. Transparency means that these entities need not to be concerned with any details regarding requesting or providing services in an open dynamic environment. Following the limitations inherent in using existing technologies such as object-oriented to model the interaction, the use of agent technology is the foundation of the proposed architecture. The chapter described a detailed design of an agent privacy-based brokering architecture that allows different domain entities to solicit help and select an appropriate privacy degree suitable to their concern. Each privacy degree is modeled and designed as an agent with specific architecture and a relevant interaction pattern. As proof of concept, the chapter illustrated a prototype of the proposed architecture to support information gathering in distributed cooperative healthcare systems.

Chapter 5

SUMMARY AND CONCLUSION

The aim of the research presented in this dissertation is to define a generic brokering architecture that enables cooperation under a desired level of privacy protection in CDS. The work presents in depth analysis of the capability-based brokering with the ability to support different degrees of privacy and accordingly proposes various interaction protocols and the suitable mechanisms of the coordinated control. In this thesis we introduced a suitable structure for formalizing and representing the privacy-based interaction protocols. Furthermore, these protocols are analyzed and consequently we provide a detailed design and implementation guidelines for a privacy-based brokering model in CDS environments. This chapter reviews the main contribution of this work, outlines some of its limitations, and suggests future research directions.

5.1. Summary of Contributions

The main objective of the research presented in this dissertation has been directed to provide a fundamental understanding of the capability-based coordination in CDS with a special focus on privacy. In this thesis, we have proposed an agent-based brokering framework that provides seamlessly coordination solutions and presents additional privacy opportunities to various participants within cooperative distributed systems. The proposed multi-layer architecture minimizes the complexity encountered in direct-interaction architectures (where interactions between agents often utilize more complex processes for encompassing a series of message exchanges and forming a single point of failure) and makes it less vulnerable to failure. The following summarizes the main contributions:

5.1.1. Brokering Model and Architecture

The brokering is viewed as a capability-based coordination solution in cooperative distributed systems. Architecturally, the proposed model is viewed as a layer of services where different roles can be played by the various entities (requestors, brokers and providers). The brokering role into several sub-roles based on the attributes designated to describe the desired privacy degree of both the service provider and the service requestor. Each role is modeled as an agent with a specific architecture and an interaction protocol that is appropriate to support a required privacy degree.

Within the layer two sets of brokering entities are available to service requesters and providers. The first set handles interactions with requestors according to the desired privacy degree that is appropriate to their preferences, while the other set supports privacy degrees required by service providers. A brokering pattern is realized by the different roles played by the domain entities and their corresponding brokering agent. A complete brokering scenario is accomplished by performing different levels of interaction namely: (1) Requester-to-Broker Interaction, (2) Broker-to-Broker Interaction and (3) Broker-to-Provider Interaction. Different combinations within the layer can take place to support the Inter-Brokering interactions. The proposed layered-architecture provides an appropriate separation of responsibilities, allowing developers and programmers to focus on modeling solutions and solving their particular application's problems in a manner and semantics most suitable to the local perspective.

Another important innovative aspect of the model is that it treats the privacy as a design issue that has to be taken into consideration in developing brokering services for cooperative distributed systems.

By utilizing the Agent-Oriented paradigm, the privacy-based brokering is modelled at a high level of abstraction, in which the distributed environment is viewed collectively as a coherent universe of interacting and collaborative agents and consequently provides high degree of decentralization of capabilities, which is the key to system scalability and extensibility.

5.1.2. Interaction Protocols

In this work, we have defined the generic architecture of the privacy-based interaction protocols in CDS environments, i.e. the basic components and the associated set of communication between involved entities. The various proposed interaction protocols allow requesters and providers to select the privacy degree that is appropriate to their concerns and desires. The interaction protocols can be viewed as reusable software components to design privacy-based interactions in open distributed environments. The developed interaction protocols define generic privacy architecture of the agents' interaction, express many fundamental and essential characteristics of an agent's interaction components, and provide a suitable structure for formalizing agents' interaction, which is an important characteristic in building correct privacy-based brokering specifications.

5.1.3. Privacy

Within the context of brokering, we model privacy in terms of the ability of CDS entities to reveal or hide its information related to the identities, requests and and/or capabilities. Each privacy degree is supported by a dedicated brokering entity (agent) with a specific architecture and interaction protocol. Requesters and providers are able to conceal their privacy concerns from the whole environment including the brokering layer itself.

The brokering layer incorporates a means of *virtual pseudonymity* (protecting identities) and *anonymity* techniques (hiding the goals and capabilities) since the brokering agents within the layer act as proxies to both service requestors and providers. Clearly, in every protocol, the interactions within the layer are constrained to sending service requests and receiving service offerings and results without having to reveal who is actually requesting or providing the service.

5.1.4. Formulation and Description

The work presented a suitable structure for formalizing the agent interactions, which is an important characteristic for building correct privacy based interaction protocols. This architecture, which expresses many fundamental and essential characteristics of agent interaction, can be reused to develop different protocols.

The interaction protocols are described in terms of a combination of the different interactions within the brokering layer and the possible interactions with the domain entities. Each protocol is captured and modeled using the Input/Output Automata (IOA) which depicts the entities' behavior in any privacy-based brokering scenario. To provide a deep understanding and formal treatments of these protocols, we have also applied the object oriented paradigm to model and represent the various protocols using UML interaction diagrams (sequence diagrams). With in depth analysis, we have shown that the privacy-based protocols depicted by the sequence diagrams represent a set of patterns that can be abstracted and formalized at a high level of abstraction.

5.1.5. The Use of the Architecture in Application Domains

The feasibility of the proposed agent-based model has been demonstrated by applying it to a vital application domain. For example, in the healthcare domain, the increasing demand and dependency on information in healthcare organizations has brought the issues of privacy to every aspect of the healthcare environment. It is expected that medical data such as genome information, medical records, and other critical personal information must be respected and treated with caution. However, users still prefer to have the ability to control the distribution of personal information in such a way that guarantees the accessibility of the right information from the appropriate source whenever required. The high degree of collaborative work needed in healthcare environments implies that developers and researchers should think of other venues that can manage and automate this complex collaboration efficiently.

Nevertheless, privacy concerns over the inappropriate use of the information make it hard to successfully exploit the advantages of sharing such information. This restricts the willingness of healthcare individuals and personnel to disseminate or publicize information that might lead to adverse outcomes. Within this context, a healthcare environment is modeled as a cooperative distributed system, in which entities are able to exercise some degree of authority in sharing information about their identities, preferences and capabilities. The privacy model is very desirable in different healthcare sectors where it can efficiently govern different types of health data such as genetic, HIV, mental health and pharmacy records from being distributed or abused.

5.2. Limitations

The work presented in this dissertation attempts to investigate, analyze and address issues related to enable cooperation under a desired level of privacy protection in open distributed cooperative systems. Consequently, a few assumptions and restrictions have been introduced during the course of this analysis as useful simplifications. This section discusses the limitations of the research effort.

Some of the proposed protocols assume that the involved brokering entity behaves in a trustworthy manner in terms of guaranteeing the desired privacy degree and has no incentive or intention in violating any revealed privacy attribute during a given brokering scenario. However, there is an obvious need for the different participants to use an appropriate trust model to analyze and asses the risks of revealing their privacy attributes to the relevant brokering entity.

Regardless of how many service requests are already received by a brokering entity, a new request can be accepted. However, the model assumes that a brokering entity responds and fulfills a single request at a time. In some protocols, the brokering entities are a communication bottleneck (since all request and replies need to go through the brokering entity).

In every interaction, agents are assumed to not violate their commitments. The main reason behind this assumption is to avoid "*commitments revision*" that has no direct relation or bearing on the interdependency problem. However; this issue will inevitably arise in environments that are unreliable i.e. where the violation of commitments goes beyond the agents' capabilities.

5.3. Directions for Future Research

A crucial element in addressing privacy concerns is the level of trust between domain entities and the brokering layer. In security, trust relates much to the degree of confidence that an entity has in the ability of other entity to conform to any selected privacy requirements. Entities should be able to generate a *quantified* trust measure about the brokering layer. Therefore, mapping privacy to trust would provide a mechanism for different participants to determine the relevant privacy degrees. In other words, requesters and service providers would be able to generate trust relationships with the brokering layer prior to any interaction.

To deal with the heterogeneity characteristic of the CDS, the brokering shall have the ability to process requests and description of capabilities by utilizing a formal, adequate and expressiveness representation. In general, the brokering entities have to dynamically understand, respectively interpret service requests and accordingly determine which of the services capabilities are most appropriate to fulfill a given request. The representation shall be rich enough to formulate and describe the privacy concerns, the permissions, rules, and allowed data flows in legislative manner.

Locating relevant services presumes that its capabilities can be named at any instant; this implies the utilization of registration and naming services. However, one direction is to consider the scalability of the proposed brokering model in which brokering entities are to be able to cross register services' capabilities from one society to another.

The choice of matching mechanism depends on the structure and semantics of the descriptions to be matched as well as the desired privacy level. One of the important directions for future work is to expand the proposed model to include the capability of semantic brokering, by introducing new functionality in the layer to resolve common types of structural and semantic heterogeneity. We believe that it will be necessary to support multiple, independently created and managed ontologies that capture the terminologies of different and sometimes overlapping domains. This requires some ontological services when dealing with processing request/services that provide functionalities such as: managing domain ontologies that capture standardized terminologies, defining the ontological relationships between terms across different ontologies and the mapping of one request expressed in a particular ontology into another request using terms from another related ontology.

Besides the proof that the system implementation corresponds to the privacy model, we would like to focus on the formal analysis of the soundness and correctness of the proposed interaction protocols and show that the model enforces the stated privacy degrees. One step in the formal system verification is to prove that the specification conforms to functions, invariants and constraints of the model. One of the proposed directions is to use is the TLC tool [92], which is a model checker for specifications written in Temporal Logic of Actions (TLA) [52].

BIBLIOGRAPHY

[1] Actional Control Broker , Available online : http://www.actional.com/

[2] Aldea A. , López B, Moreno A., Riaño V. and Valls A., "A Multi-Agent Systems for Organ Transplant Coordination", Artificial Intelligence in Medicine, Lecture Notes in Computer Science, Springer Verlag, 413-416, 2001.

[3] An Introduction to Cryptography, in PGP 6.5.1 User's Guide, New York Associates Inc. p.11-36, online : http://www.fi.pgpi.org/doc/pgpintro/.

[4] Anonymizer Tool, Available online: www.anonymizer.com

[5] Arisha K., Eiter T., Kraus S., Ozcan F., Ross R. and Subrahmanian V., "IMPACT: The Interactive Maryland Platform for Agents Collaborating Together", *IEEE Intelligent Systems* magazine, Vol. 14, Nr. 2, pps 64 -72, 2001.

[6] Bao F. and Deng R., "Privacy protection for transactions of digital goods," Pro. of international conference on information and communications security, LNCS 2229, pp. 202-213, 2001.

[7] Beneventano D., and Bergamasch, S., "The MOMIS Methodology for Integrating Heterogeneous Data Sources", IFIP World Computer Congress. Toulouse France, 22-27 August 2004.

[8] Beresford A. and Stajano R., "Location Privacy in Pervasive Computing", *Pervasive Computing*, IEEE, Volume 2, Issue 1, pps. 46-55, 2003.

[9] Blaze M., Feigenbaum J. and Strauss M., "Compliance Checking in the PolicyMaker Trust Management System," Financial Cryptography: 2nd Int'l. Conf, British West Indies.: Springer-Verlag. LNCS 1465: 254--274, 1998.

[10] Bolognesi T., van de Lagemaat J., and Vissers, C. editors. "The LotoSphere Project", Kluwer Academic Publishers, London, UK, 1995.

[11] Brooke J. and Fellows D., "An Architecture for Distributed Grid Brokering", 11th International Euro-Par Conference, Lisbon, Portugal, 2005.

[12] Camarinha-Matos L. and Afsarmanesh H., "Virtual Communities and Elderly Support", Advances in Automation, Multimedia and Video Systems, and Modern Computer Science, WSES, pp. 279-284, 2001.

[13] Chawathe S. et al, “The TSIMMIS project: integration of heterogeneous information sources”, *In Proceedings of the 10th Meeting of the Information Processing Society of Japan*. pp. 7-18, 1994.

[14] Cheng H., Zhang D. and Tan J., “Protection of Privacy in Pervasive Computing Environments”, *International Conference on Information Technology: Coding and Computing*, 4-6 April 2005, pp. 242-247.

[15] Cheyer A., and Martin D., “The Open Agent Architecture”. *Journal of Autonomous Agents and Multi-Agent Systems*, vol. 4 , no. 1, pp. 143-148, March 2001.

[16] Chor B., Goldreich O., Kushilevitz E. and Sudan M., “Private Information Retrieval”, *In 36th Annual Symposium on Foundations of Computer Science*, pages 41–50, 1995.

[17] Chung E. et al., “Development and Evaluation of Emerging Design Patterns for Ubiquitous Computing,” Patterns C1-C15, DIS2004, 2004..

[18] Clarke R., “Identification, Anonymity and Pseudonymity in Consumer Transactions : A Vital System Design and Public Policy Issue”: Available online as of July, 2007 http://www.anu.edu.au/people/Roger.Clarke/DV/AnonPsPol.html

[19] CrossWorlds (WebSphere Business Integration Toolset), Available online : http://www-306.ibm.com/software/integration/wbitools/

[20] Decker K., Sycara K., and Williamson M.,” Middle-agents for the internet” In IJCAI97 International Joint Conference on Artificial Intelligence, Nagoya, Japan, pps. 578-584. 1997.

[21] Dumitrescu C., “Problems for Resource Brokering in Large and Dynamic Grid Environments”, The 12th International Euro-Par Conference, Germany, pps. 448-458 2006.

[22] Feigenbaum J., Freedman J., Sander T. and Shaostack. A., “Privacy Engineering for digital Rights Management Systems”. ACM Workshop on Security and Privacy in Digital Rights Management, pps.76-105, 2001.

[23] Ferreira L. and Lopes de Souza W., “Step-wise refinement design example using LOTOS”, In FORTE, pages 255–262, 1990.

[24] FIPA Agent Communication Language (FIPA- ACL), Available online : http://www.fipa.org/repository/aclspecs.html

[25] FIPA Agent Software Integration Specification. [Online]. Available: http://ww.fipa.org/specs/fipa00079/XC00079B.htm.

[26] Foss J., “Brokering Automated Enterprises”, Internet Society, Available online: http://www.isoc.org/inet99/proceedings/1d/1d_3.htm

[27] Garcia-Molina Y. et al, "The TSlMMlS approach to mediation", Data models and Languages. *Journal of Intelligent Information Systems*, 1996.

[28] Garland S. and. Lynch N., "Using I/O automata for developing distributed systems," *Foundations of Component-Based Systems*, Cambridge University Press, 2000.

[29] Genersereth M. et al. "Infomaster: Information Integration System", in proceedings of 1997 ACM SIGMOD Conference, 1997.

[30] Gerigoris, A. et al., "A Deductive Semantic Brokering System", R. Khosla et al. (Eds.): KES 2005, LNAI 3682, pp. 746.752, 2005.

[31] Ghenniwa H. and Huhns M., "Intelligent Enterprise Integration: eMarketplace Model", in Creating Knowledge Based Organizations, J. Gupta and S. Sharma (Eds.), Idea Group Publishing, Hershey, Pennsylvania, USA, pp. 46-79, 2004.

[32] Ghenniwa H. and Kamel M., ``Interaction Devices for Coordinating Cooperative Distributed Systems", Journal of Intelligent Automation and Soft Computing, 2000.

[33] Global InfoTech, Inc. "A Report on the Applicability of Mediation in ALP" A Technical Report, 1998.

[34] Goldberg I., Wagner D. and Brewer E., "Privacy-enhancing technologies for the Internet" In Proceedings of IEEE COMPCON 97, pages 103-109, 1998.

[35] Goldschlag D., Reed M, Syverson P., "Onion Routing", Communications of the ACM, Volume 42, Number 2, 1999.

[36] Graham J. and Decker K., "Towards a Distributed, Environment-Centered Agent Framework –The DECAF Agent Framework", Lecture Notes in Computer Science, pps. 290 – 304, 2000.

[37] Hafiz M., "A Collection of Privacy Patterns", Plop 2006 Conference, Portland, Oregon, October 21-23, 2006.

[38] Health Insurance Portability and Accountability Act (HIPAA). Available :http://www.intellimark-it.com/privacysecurity/hipaa.asp

[39] Howard R. and Kerschberg L., "Semantic Brokering via Intelligent Middleware Agents within a Knowledge-Based Framework", *The IEEE/WIC/ACM International Conference on Intelligent Agent Technology*", China, 513-516, 2004.

[40] Initiative for Privacy Standardization in Europe (IPSE). Available: http://www.hi-europe.info/files/2002/9963.htm

[41] Java Agent Development Framework: Jade, Home Page: http://www.jade.cselt.it/

[42] Java Web Services Developer (JWSDP); [Online]: URL:/http://java.sun.com/webservices/jwsdp/index.jspS.

[43] Jennings N., Turner P., Garcha K., Foss J., "Brokerage in an Information Economy" , Available online : http://www.isoc.org/inet2000/cdproceedings/7a/7a_1.htm#r5

[44] K. Decker et al. "MACRON: An Architecture for Multi-agent Cooperation Information Gathering", Proceedings of the CIKM 95 Workshop on Intelligent Information Agents, pps. 319–346, 1995.

[45] Kashyap V. and Sheth A.., "Semantics Based Information Brokering", Proceedings of the 3rd International Conference on Information and Knowledge Systems: 363-370, 1994.

[46] Kenny S. and Borking J., "The Value of Privacy Engineering". [Online]. Available : http://elj.warwick.ac.uk/jilt/02-1/kenny.html

[47] Kertész A. and Kacsuk, P., "Grid Meta-Broker Architecture: Towards an Interoperable Grid Resource Brokering Service", W. Lehner et al. (Eds.): Euro-Par 2006 Workshops, LNCS 4375, pp. 112–115, 2007.

[48] Korba L. and Song R., "Investigating of Network-Based Approaches for Privacy". NRC Report: ERB-1091, NRC No.: 44900, Nov. 2001.

[49] Kuokka D. and Harrada L., "On using KQML for matchmaking". *In Proceedings of the First International Conference on Multi-Agent Systems*, pps. 239–245, 1995.

[50] Kurose J. and Ross k., "*Computer Networking: A top down approach featuring the Internet*", 2nd e, Addison Wesley, 2005.

[51] L. Cabral, Domingue, J., Galizia S., Gugliotta A., Tanasescu V., Pedrinaci C. and Norton B.,"IRS-III: A Broker for Semantic Web Services Based Applications", pps.: 201-214, 2006.

[52] Lamport L., "A Temporal Logic of Actions" available online: "http://research.microsoft.com/users/lamport/pubs/old-tla-src.pdf.

[53] Langheinrich M., "A Privacy Awareness System for Ubiquitous Computing Environments", *Ubicomp*, Lecture Notes in Computer Science, Volume 2498, pps. 237-245, Springer, 2002.

[54] Lee W. and Chang C., "User Identification and Key Distribution Maintaining Anonymity for Distributed Computer Network," Computer System Science Engineering, vol. 15, no. 4, pp. 113-116, 2000.

[55] Li L. and Horrocks I., "A Software Framework for Matchmaking Based on Semantic Web Technology". In Proceedings of the 12th International Conference on WWW. Budapest Hungary 2003.

[56] Lindell Y., and Pinkas B., "Privacy preserving data mining", In Advances in Cryptology - CRYPTO'00, pages 36–54, 2000.

[57] Lynch N. and Tuttle M., "An Introduction to Input / Output Automata", CWI Quarterly, 2(3):219-246, 1998.

[58] Mangipudi K. and Katti R., "A Secure Identification and Key Agreement Protocol with User Anonymity (SIKA)," Computers & Security, vol. 25, pp. 420-425, 2006.

[59] Mena E., llarramendi A., Kashyap V. and Sheth A., "OBSERVER: An approach for query processing in global information systems based on interoperation across pre-existing Ontologies", Distributed and Parallel Databases, 8(2):223–271, 2000.

[60] Minano B., Lera I., Sancho1 P., Juiz1 C. and Puigjaner R., "Context-Broker Service Architecture for AmI Systems through Mobile-Agents and Ontologies as Middleware", ISPA 2006, pps. 907-916 Italy, Dec. 2006.

[61] Moreno A. and Isern D., "Accessing distributed health-care services through smart agents", the 4th IEEE Int. Workshop on Enterprise Networking and Computing in the Health Care Industry - HealthCom 2002 -France, 2002.

[62] Moreno A., Valls A. and Bocio J., "Management of Hospital Teams for Organ Transplants Using Multi-Agent Systems", Artificial Intelligence in Medicine, Lecture Notes in Computer Science, Springer Verlag, 413-416, 2001.

[63] Motta E., Domingue J., Cabral L. and Gaspari M., "IRS-II: A Framework and Infrastructure for Semantic Web Services" In Proc. of the International Semantic Web Conference (ISWC'03), USA-Oct 2003.

[64] Myeong-Wuk W., Abdel Momen A. and Agha G.., "A Flexible Coordination Framework for Application-Oriented Matchmaking and Brokering Services" *Proceedings of IEEE/WIC/ACM IAT (Intelligent Agent Technology*)-2004, pp. 393-396, China, 2004.

[65] Navarro F., Jones K., Gordhan S. and Garnham N., "An Agent-Based Service Brokering Architecture for Multi service Next-Generation Networks", FUJITSU Sci. Tech. J., pp.97-108, 2001.

[66] NEON eBusiness Integration Servers, Available online : http://www.neonsys.com/

[67] Nicholas G., Harris S. and Shadbolt N., "Agent–Based Semantic Web Services" *Journal of Web Semantics: Science, Services and Agents on the World Wide Web* 1(2), Hungary, pps. 141-154, 2003.

[68] O. Baudron and J. Stern, "Non-interactive private auctions" *In Financial Crypto'01*. Springer–Verlag, pps. 364 - 378 , 2002.

[69] Object Management Group, OMG Inc.: "CORBA: The Common Object Request Broker Architecture and Specification"; Revision 2.0. Framingham, MA, July 1995.

[70] P3P - The platform for privacy preferences 1.0 (P3P1.0) specification”. W3C Recommendations. [Online]. Available: www.w3.org/TR/P3P/, April 2007.

[71] Paolucci M. et al. “A Broker for OWL-S Web Services”, *In Proceedings of AAAI 2004*, USA, March, 2004.

[72] Paolucci M., Soudry J., Srinivasan N. and Sycara K., “"Untangling the Broker Paradox in OWL-S", *In Proceedings of AAAI 2004 ,* 17(3), pp. 84-86, March, 2004.

[73] Paton N., Stevens R., Baker P. and Goble C., “Transparent Access to Multiple Bioinformatics Information”. The 11th Int. Conf. on Scientific and Statistical Databases (SSDBM), IEEE Press, 118-147, 1999.

[74] PISA – The Privacy Incorporated Software Agent. [Online]. Available: http://www.cbpweb.nl/bis/top-1-1-9.html.

[75] Privacy protection for web Services - AC020, Available online: http://www.w3.org/TR/wsa-reqs/#AC020.

[76] Purvis M. et al. “The NZDIS Project: multi-agent system for the integration of distributed environmental information”, Environmental Modeling and Software, Vol. 18 No. 6, pp. 565-72, 2003.

[77] Reiter M. and Rubin A., “Crowds: Anonymity for Web Transactions”, ACM Transactions on Information and Systems Security (TISSEC), Volume 1, Issue 1, 1999.

[78] Rivest R., Shamir A. and Adleman L., “A Method for Obtaining Digital Signatures and Public-Key Cryptosystems”, Communications of the ACM, 21 (2), pp. 120-126, February 1978.

[79] Romanosky S. et. al, “Privacy Patterns for Online Interactions”. Plop 2005, Monticello, Illinois, Sept. 7-10, 2005.

[80] Schumacher M., “Security Patterns and Security Standards - With Selected Security Patterns for Anonymity and Privacy,” European Conference on Pattern Languages of Programs, (EuroPLoP), 2002

[81] Schümmer T., “The Public Privacy - Patterns for Filtering Personal Information in Collaborative Systems”, Technical Report, FernUnivesität in Hagen, 2004.

[82] SeeBeyond EBusiness Integration Suite, Available online: http://www.sun.com/software/seebeyond/

[83] Shankarama V., Amorosiadou V. and Robinson B., “Agents in Medical Informatics”, in Proc. Of IASTED International Conference on Applied Informatics, Austria, 2000.

[84] Sheth A. and Larson J., ``Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Database", ACM Computing Surveys, vol.22, no.3, pp. 183-235, 1990.

[85] Silver B., Andonyadis C. and Morales A., "Web-based healthcare agents; the case of reminders and to-dos", *Artificial Intelligence in Medicine*, 14(3), 295-316, 1998.

[86] Smith R., "The Contract Net Protocol: High-level Communication and Control in a Distributed Problem Solver". IEEE Trans. On Computers C-29 (12): 1104- 1113, 1980.

[87] Sun Microsystems, "JINI Architecture Specification, Version 1.2," Sun Microsystems, December 2001, http://www.sun.com/jini/.

[88] Sycara K., et al. "The RESTINA MAS Infrastructure", Tech. report CMU-RI-TR-01-05, Robotics Institute, CMU, http://www.ri.cmu.edu/pubs/prub_3509.html. March 2001.

[89] Sycara K., Lu J., Klusch M. and Widoff S., "Dynamic Service Matchmaking among agents in Open Information Environments", Journal ACM SIGMOD Record, Special Issue on Semantic Interoperability in Global Information Systems, 1999.

[90] The Association of American Physicians and Surgeons , "Doctors Lie to Protect Patient Privacy" – Poll Survey , Available online : http://www.aapsonline.org/press/nrnewpoll.htm

[91] The Mercator Enterprise Broker Available online : http://h71028.www7.hp.com/enterprise/html/4270-0-0-0-121.html

[92] The TLA+ tool, available online: http://research.microsoft.com/users/lamport/tla/tools.html.

[93] Venugopal S., Buyya R. and Winton L., "A Grid Service Broker for Scheduling e-Science Applications on Global Data Grids", Journal of Concurrency and Computation: Practice and Experience, Wiley Press, USA, 2005.

[94] Wang C., and Leung H., "Mobile Agents for Secure e-Commerce Transaction with Privacy Protection of the Customers", the 2005 IEEE International Conference on e-Technology, e-Commerce and e-Service – China, 2005.

[95] Web Services, Executive and Technical Papers available at http://www.webservices.org/

[96] Wiederhold G., "Mediation in the Architecture of Future Information Systems", The IEEE Computer Society Press, 23(3), 1992.

[97] Woelk D. et al, "Infosleuth project", Available online: http://www.mcc.com/projects/infosleuth.

[98] Woerndl W., Koch M., "Privacy in Distributed User Profile Management", (WWW2003), Poster, Budapest, Hungary, May 2003.

[99] Woods S. and Barbacci M., “Architectural Evaluation of Collaborative Agent-Based Systems”, Technical Report, CMU/SEI-99-TR-025, Software Engineering Institute, Carnegie Mellon University, PA, USA, 1999.

[100] Wu T. and Hsu C., “Efficient user identification protocol with key distribution preserving anonymity for distributed computer networks,” Computers & Security, vol. 23, pp. 120-125, 2004.

[101] X.509 Certificates and Certificate Revocation Lists (CRLs), Sun Microsystems Inc. http://java.sun.com/products/jdk/1.2/docs/guide/security/cert3.html

[102] Xiaodong J. and Landay J., “Modeling privacy control in context-aware systems”, *Pervasive Computing*, IEEE, Volume 1, Issue 3, pps. 59-63, 2002.

[103] Yee G., Korba L. and Song R., “Ensuring Privacy for E-Health Service”, Proceedings of the First International Conference on Availability, Reliability and Security (ARES 2006). Vienna, Austria. April, 2006.

[104] Yu T. and Lin K., “A Broker-Based Framework for QoS-Aware Web Service Composition”, IEEE International Conference on e-Technology, e-Commerce and e-Service, Hong Kong, 2005.

VITA

Name: AbdulMutalib Mohamed Masaud-Wahaishi

Post-secondary Education and Degrees:

The University of Western Ontario
London, Ontario, Canada
2001-2003 M. Sc. Eng.

The University of Western Ontario
London, Ontario, Canada
2003-2007 Ph.D.

Honors and Awards:

- International Graduate Student Scholarships (IGSS), University of Western Ontario.
- Special University Scholarships (SUS), University of Western Ontario.
- Best Teaching Assistant Award nomination 2005
- Best Teaching Assistant Award 2006, Department of Electrical and Computer Engineering, UWO

Related Work and Experience:

Teaching Assistant
The University of Western Ontario
2001-2007

Project manager, department manager
The Great Man-Made River Project
Libya
1988-2000

Publications:

1. A. Masaud-Wahaishi, H. Ghenniwa and W. Shen, "Integration in Cooperative Distributed Systems: *Privacy-Based Brokering Architecture for Virtual Enterprises*", accepted for publication in *Virtual Enterprise Integration: Technological and Organizational Perspectives,* G. Putnik and M. Cunha (Eds.), published by IDEA Group, Inc, imprints 2005.
2. A. Masaud-Wahaishi and Hamada Ghenniwa, "Information Brokering Architecture for Healthcare SmartHomes: Privacy Based Model". The First International Workshop on Smart Homes for Tele-Health, Niagara Falls , Canada, 2007
3. A. Masaud-Wahaishi and Hamada Ghenniwa, "Agent Privacy-Based Brokering Architecture for Enterprise Cooperative Systems", the 9th International Conference on Enterprise Information Systems, Portugal, 2007.
4. A. Masaud-Wahaishi, Hamada Ghenniwa and Weiming Shen, "Agent-Based Brokering Architecture for Collaborative Distributed Virtual Environments", The 8th IFIP Working

Conference on Virtual Enterprises, Portugal, September, 2007.

5. A. Masaud-Wahaishi and Hamada Ghenniwa, “Formal Specification for Privacy-Based Brokering Services for Cooperative Distributed Systems”. The Fourth International Conference on Cooperative Internet Computing (CIC 2006) - Hong Kong, China, 2006
6. A. Masaud-Wahaishi, H. Ghenniwa, and W. Shen, “Brokering Services in Cooperative Distributed Systems: Privacy-Based Model”, 4th International Conference on Electronic Commerce and Web Technologies- EC-Web 2003.
7. A. Masaud-Wahaishi and M. Bennett “Metrics for Agents”, 7th IASTED International Conference on Software Engineering and Applications-CA, USA-2003.
8. A. Masaud-Wahaishi, H. Ghenniwa, and W. Shen, “Healthcare Information Brokering: The value of Privacy” The 16th International Conference on Advanced Information Systems Engineering CAiSE04, Latvia June 2004.
9. A. Masaud-Wahaishi, H. Ghenniwa, W. Shen “Agent-Based Information Brokering for Healthcare Environments” World Automation Congress, WAC 2004, Spain, 2004.
10. A. Masaud-Wahaishi, H. Ghenniwa, and W. Shen, “Protecting Privacy in Healthcare Environment: An Agent-Based Information Brokering Architecture” IEEE Canadian Conference on Electrical and Computer Engineering, CCECE2004, Niagara Falls, May 2004
11. Abdul Masaud-Wahaishi and Ghenniwa, H. “Integration in Cooperative Distributed Systems”. Proceedings of the Graduate Research Symposium, UWO (2001).
12. Abdul Masaud-Wahaishi and Ghenniwa, H. “Brokering Services in Cooperative Distributed Systems”. Proceedings of the Graduate Research Symposium, UWO (2002).