JOURNAL OF OBJECT TECHNOLOGY

Online at http://www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2008**

Vol. 7, No. 6, July-August 2008

Domain-Specific Architecture for Software Agents

Suganthy. A
M.Tech Computer Science, Ramanujan School of Mathematics and Compuer Science, Pondicherry University, India

T. Chithralekha
Reader, Department of Banking Technology, Pondicherry University, India

Abstract

The principal contribution of this paper is a methodology for the support of developing domain-specific software agents and the development of a Reference architecture for agents pertaining to web service discovery by following the phases described in the proposed methodology. The proposed methodology and the resulting architecture are evaluated to illustrate its appropriateness in contributing for domain-specific software architecture.

Keywords

Software Agents, Domain-Specific Agents, Methodology for Domain-Specific Agents, Service Discovery Agent.

1 INTRODUCTION AND MOTIVATION

The need for improved methods of software engineering is widely recognized as a way to improve the quality of the software products. One prominent idea is to synthesize new application systems by configuring appropriate sets of reusable software components.

In order to realize the promise of reusable software, it is necessary to engineer highly reusable software components from the start. “Domain-Specific Software Architectures (DSSAs)” have been advanced as a methodology for factoring large software systems into components that have high reuse potential within a particular application domain.

As software agents are widely used in many application domains like information gathering, network management, decision and logistic supports etc., it is necessary to standardize domain-specific software architecture for software agents. Development of agents from domain-specific software architecture helps the agent developers to reuse the architectural artifacts from the analysis phase.

A. Suganthy and T. Chithralekha: “Domain-Specific Architecture for Software Agents”, in *Journal of Object Technology*, vol. 7, no. 6, July-August 2008, pp. 77-100 http://www.jot.fm/issues/issue_2008_07/article2/

The research areas in agent technology are focusing on standardizing the agent architecture, enabling them to interoperate with other agents. The support for the development of agents from reusable components is not yet focused. Hence, a methodology for building domain-specific software architecture for software agents could also not be found in the agent literature [Joaq-1 06].

Hence, this paper is designed to overcome the limitations of developing an agent from reusable architectural artifacts by:

- Proposing a methodology for building domain-specific architecture for software agents and
- Developing domain-specific architecture for web service discovery agent based on the proposed methodology.

An evaluation step is also carried out both for the proposed methodology and the developed domain-specific software architecture for software agents in web service discovery.

The remaining sections of this paper are arranged as follows: Section 2 of this paper gives the background of domain-specific software architecture and software agents. Section 3 describes the novel methodology for building domain-specific agents. Section 4 explains the evaluation method carried out for the proposed methodology and section 5 explains in the development method of domain-specific software architecture for service discovery agent. Section 6 gives the evaluation results of the proposed domain-specific architecture and section 7 gives the conclusion of this paper.

2 DSSA AND SOFTWARE AGENTS

Domain-specific software architecture (DSSA) covers the entire software lifecycle required to develop a software system from which concrete architectures pertaining to that domain can be substantiated. The phases involved in the development of DSSA are domain engineering and application engineering [Steven 02].

Domain engineering phase: This phase is responsible for providing the reusable core assets that are exploited during application engineering when assembling or customizing the individual applications. This phase describes the requirements of the complete family of products, highlighting both the common and variable features across the family [Richard 95], [Tracz 95]. In this phase, commonality analysis is of great importance for determining the commonalities and variabilities in the domain [Josh 05], [Baojian 99].

Application engineering phase: Application engineering phase involves the process of building an application based on DSSA.

Both these phases can be further divided into requirements analysis, design, and implementation (a typical software development lifecycle).

DSSA for Software Agents concentrates on the process of providing common and variable features and on the process of deriving a final architecture for all the applications pertaining to a domain.

As the agent architectures and frameworks were designed to match the project requirements, it creates incompatible systems that are difficult to reuse from project to project. Therefore, research in the area of agent technology has focused on the standardization of architectures supporting for distributed interoperation of agents. This forms the basis for developing core architecture for agents [Joaq-1 06]. But the standardization of agent architecture cannot facilitate the reuse of architecture of agents pertaining to a single domain. This is because the domain-specific architecture will contain domain-specific components which facilitate a higher level of reusability for developing agents pertaining to a domain.

An initial attempt in developing Reference architecture for agents is made by DARPA Control of Agent Based Systems (CoABS) which gives only a template that provides the various functional viewpoints which has to be addressed while developing architecture for agents [Craig-3]. This template only provides a guideline rather than providing an insight in developing reference architecture for agents.

The feasibility analysis for developing domain-specific agents has been conducted by NASA in 2006 [Joaq-2 06]. This is only a feasibility study and does not provide any concrete details for developing domain-specific architecture for agents.

In conclusion, it could be found that, though the necessity for developing domain-specific agents is well understood by the research community, yet realizing the development of domain-specific architecture for agents has not reached a reasonable milestone. Because of this limitation a methodology for developing domain-specific architecture for agents could also not be found. Hence the objectives of this work are devising a methodology for developing domain-specific architecture for agents and illustrate the use of the same in developing domain-specific architecture for agents.

3 PROPOSED METHODOLOGY FOR DOMAIN-SPECIFIC AGENT DEVELOPMENT

As the development of software agents is also similar to the conventional software development methods, the following phases are identified for building domain-specific agents:

- Domain engineering
- Creation of Reference Architecture
- Validation of the reference architecture and
- Application engineering

These phases differ from those used in the conventional DSSA development methods. The variations are discussed in the following sections with respect to each phase.

The proposed methodology for the development of domain-specific agents is diagrammatically represented in Figure 1. The first phase in the construction of domain-specific agent is the domain engineering. This phase identifies the domain artifacts that may be reused in developing agents in a domain. These domain artifacts are then used in the next phase for creating the reference architecture (otherwise called as domain-specific architecture). The third phase involves the validation of the reference architecture. The feedback of the validation results are then sent back to the

domain engineering phase for the refinement process. The final step is the development of agents using the reference architecture.

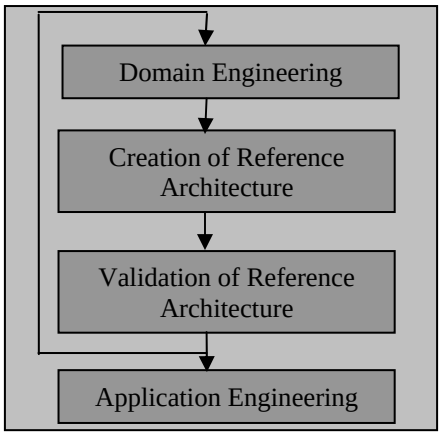

Figure 1: Methodology for Domain-specific Agent development

Domain Engineering

The purpose of domain engineering is to develop domain artifacts that may be reused in developing application for a given domain. Domain engineering consists of activities for gathering and representing information on systems that share a common set of capabilities and data. In conventional software development the domain engineering phase consists of identifying the reusable components.

In this methodology, the domain knowledge is engineered and organized in a more comprehensive manner. This phase involves in identifying the roles of the agents, as all AOSE (Agent-Oriented Software Engineering) methodologies use role abstraction for characterizing agents. These role models are similar to the one used in agent-oriented methodologies like GAIA [Woold 00], MaSE [Wood 01] etc. The agent developed using this methodology plays in two different types of roles: domain dependent Roles and application dependent Roles. Figure 2 shows the roles played by an agent.

Each Role (Domain dependent or application dependent) identified in the domain engineering phase should be defined using a Role Schema [Woold 00]. The role of an agent also identifies the agent's properties like autonomy, reactivity, proactivity and social ability that should be present in the domain-architecture. The template for defining the role schema is given in Figure 3.

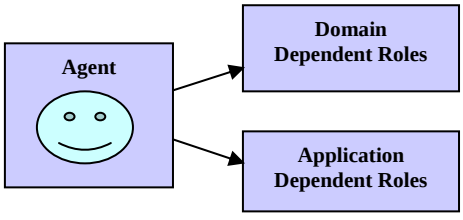

Figure 2: Roles played by an agent

Role Schema	Name of the Role
Description	Short English description of the role
Protocols and Activities	Protocols and activities in which the role plays a part
Permissions	Rights associated with the role
Responsibilities Liveness Safety	 Liveness responsibilities Safety responsibilities

Figure 3: Template for Role Schema

Activities to be carried out in the domain engineering phase of this methodology are:

- Identification of Domain Dependent Roles
- Identification of Application Dependent Roles and
- Identification of Role interactions

Identification of Domain Dependent Roles: The domain dependent roles are the roles of the agent that are common to all the applications that can be developed in a domain. For example, the communication role provides the specifications for communication between the systems (from one system to another). This specification will not be changed from one application to another and hence this role is domain dependent.

Identification of Application Dependent Roles: This step identifies the roles which consist of the customizable implementation of agents with application-specific functionality. For example, if the agent acts as a match maker role, then the match making algorithm, the parameter used in the match making process etc, will depend on the application to be developed.

Identification of Role Interactions: The final step in domain engineering phase is the identification of the interaction between the domain dependent and the application dependent roles. These interactions may help the roles to communicate and/or collaborate among themselves.

The steps used in the domain engineering phase of the proposed methodology are diagrammatically represented in Figure 4.

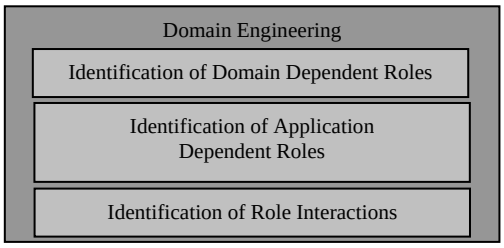

Figure 4: Domain Engineering

Creation of Reference Architecture

This phase uses the outcomes of the domain enginering phase, the domain dependent roles, the application dependent roles and the interactions among these roles. This phase involves the following activities:

- Design of Agent model and
- Design of Reference architecture

Design of Agent Model: The design of agent model involves in transforming the analysis model into sufficiently low level of abstraction for implementing the agents. The following three models are generated in this design:

- Agent model: This model documents the various agent types that will be used in the system under development and the agent instances that will realize these agent types at run time.
- Services model: This model identifies the services (functions) associated with each agent role and specifies the main properties of these services.
- Acquaintance model: This model defines the communication links that exist between agent types.

Design of Reference Architecture: The design of reference architecture involves in the creation of reference architecture for the domain under consideration.

Reference architecture should be defined in terms of the components. Hence, the roles identified in the domain engineering phase are considered to be corresponding to the components. A role identified in the domain engineering phase may correspond to a single component (one-to-one mapping) or multiple roles correspond to a single component (many-to-one) or a role can be present in more than one component (one-to-many). The mapping of the roles to components is diagrammatically depicted in Figure 5.

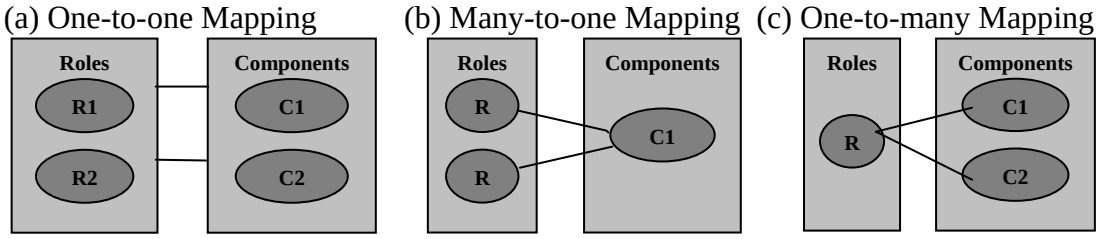

Figure 5: Mapping of Roles to Components

The crucial step in this phase is the identification of the suitable agent architecture for the placement of these roles. Existing agent architectures like Reactive architecture, Deliberative architecture, Layered architecture and Hybrid architectures are evaluated with reference to the domain under consideration for placing these roles.

Validation of the Reference Architecture

Once the reference architecture is created, it should be assessed for its quality. This phase is used for evaluating the quality of the reference architecture using any of the existing DSSA evaluation methods.

Some of the existing DSSA validation techniques are SACAM (Software Architecture Comparison Analysis Method), DoSAM (Domain-Specific Architecture Comparison Model) [Klaus 05], a variation of the SAAM model (Software Architecture Analysis Method) [Mugurel 03] and Multi-Criterion Analysis of Reference Architecture [Marius 05].

The results of the evaluation process are then sent back to the domain engineering phase for the refinement process if necessary.

Application Engineering Phase

The last phase is the development of the agent itself. This phase involves the process of building an agent based on the reference architecture. The agent to be developed should instantiate the reference architecture and it should customize the implementation details of the application dependent roles.

Application engineering is the process of developing a specific application by making use of the domain knowledge obtained during domain engineering. Application engineering proceeds by first analyzing user's requirements, then framing the corresponding application design and completing the application development by reusing the software components identified in the domain engineering phase. The application engineering phase is further divided into [Boajin 99]:

- Application requirements analysis
- Application design and
- Application development

Application requirements analysis: Application engineering starts with the requirements analysis phase. This is done by first analyzing user's requirements for the target application, and finding a matching set of roles identified in the domain engineering phase. Apart from the specific requirements of the application, the other domain-specific requirements are taken from the domain engineering and the reference architecture. Thus, this phase gets its input from the outcomes of the domain engineering and the reference architecture and developers work is simplified as the reusability is started from the analysis phase of the application development.

Application design: The application requirements are then used for the design of the application under development. The reference architecture is instantiated in this phase and the application requirements from the previous phase helps to define the application dependent roles of the reference architecture. The architecture pertaining to the specific application is developed in this phase.

Application development: The final step is the development of the agents using any agent development languages and agent development tools/frameworks.

4 EVALUATION OF THE METHODOLOGY

The evaluation of the proposed methodology helps to understand the limitations existing in it and thereby helps to develop a better solution. Evaluation process identifies the strengths, weaknesses and the ways to improve the methodology. Only a few frameworks for comparing the agent-oriented methodologies have been suggested [Sudeikat], [Chia-En 07].

The evaluation process carried out for the proposed methodology includes the criteria for both software processes and agent-oriented properties. The evaluation framework adopted here is based on the work done by Sturm and Shehory [Sturm 03]. This framework consists of the following four evaluation criteria:

- Concepts and Properties
- Notations and Modeling Techniques
- Process
- Pragmatics

Concepts and Properties

A concept is an abstraction or a notation inferred or derived from specific instances within a problem domain. A property is a special capability or a characteristic. Concepts and Properties collect all the basic building blocks of Agents. This category deals with questions on whether or not a methodology adheres to the basic notions of Agents.

Notations and Modeling Techniques

Notations are a technical system of symbols used to represent elements within a system. A modeling technique is a set of models that depict a system at different levels of abstraction and different system's aspects. Hence the methodology will found to be complete only when it supports the notations and modeling techniques.

Process

A process is a series of steps that guide practitioners to construct a software system from the beginning to the end. It serves as a detailed guideline of all activities throughout subsequent phases. The newly developed methodology should also have a well defined process and should describe the activities that it supports.

Pragmatics

Pragmatics refers to real use scenarios as developers apply methodology in building Agent-based systems. This provides reviews in real situations from instituting concepts, building models, and implementing details. This division deals with the exploration of practical deployment while using a methodology.

Evaluation Results

The evaluation process carried out for the proposed methodology based on the four evaluation criteria: Concepts and Properties, Notations and Modeling techniques, Process and Pragmatics are summarized in Table 1. These evaluation results show that the methodology is suitable, appropriate and well defined for the development of the domain-specific agents.

Criteria	Issues to be addressed	Supportability in the proposed methodology	Reason
Concepts and Properties	Autonomy	Yes	The roles identified in the domain engineering phase possess the autonomy property.
	Mental Mechanism	Domain dependent	Mental mechanism is possessed by the roles identified in a domain.
	Adaptation	Yes	The application dependent roles are flexible enough to the changing environment.
	Concurrency	No	-

	Communication	Yes	Communication is supported by the interaction between the domain dependent and the application dependent roles.
	Collaboration	Yes	The methodology supports collaboration among the domain dependent and the application dependent roles.
	Abstraction	Yes	Abstraction is provided with the help of role schemas for describing the roles of the agent.
	Agent-oriented	Yes	The roles identified in the methodology are mapped to agents and thereby supporting agent-oriented criteria.
Notations and Modeling Techniques	Expressiveness	Yes	This methodology uses Role schema for supporting expressiveness.
	Complexity	Yes	Decomposition of agent into roles helps to avoid the complexity of the agent.
	Modularity	Yes	Identification of roles helps in supporting modularity.
	Executable	Yes	Support for the development of prototypes using the roles helps the executable property.
	Refinement	Yes	The role schema and the protocol definitions of the agents are used in the refinement process.
	Traceability	Yes	The methodology supports the traceability of the development process in a linear fashion.
Process	Specification	Yes	The methodology provides a way of forming a system specification from the scratch.
	Life-cycle Coverage	Yes	Analysis, Design, Implementation and Testing phases are covered in this methodology.
	Architecture Design	Yes	The methodology provides mechanisms to facilitate design by using the roles and their interactions.
	Implementation Tools	No	-
	Deployment	No	-
Pragmatics	Tools Available	No	-
	Required Expertise	Minimal	The methodology does not require any prior knowledge for its application.
	Modeling Suitability	No	The methodology is not based on any Model.
	Domain Applicability	Applied to all domains	The methodology does not restrict any particular domains for its applicability in developing domain-specific software architecture for agents.
	Scalability	Yes	The methodology can also be used for the development of MAS.

Table 1: Evaluation Results

5 CASE STUDY – DSSA FOR SERVICE DISCOVERY AGENT

This section explains the development of DSSA for service discovery agent using the methodology described in Section 3.

Domain Engineering

The domain engineering phase of developing the web service discovery agent is carried out by static analysis method [Pragnesh 06]. By this method, the domain dependent roles, the application dependent roles and the interactions among those roles are identified by analyzing the existing web service discovery architectures. The main goal of this domain engineering phase is finding the architectural artifacts pertaining to web service discovery.

This section explains the domain engineering phase carried out for the service discovery agent. Role schema for each of the domain dependent roles identified in the domain engineering phase is also given in this section.

Identification of Domain Dependent Roles

The domain dependent roles are identified by analyzing the common properties found in all of the existing web service discovery architecture. The definition of these roles remains to be the same with respect to all the applications in the service discovery agent. The roles are defined using the Role Schema.

The following are the domain dependent roles identified in the domain engineering phase:

- Communication handler
- Message handler
- Repository
- Security

Role Schema for Communication Handler

The purpose of this role is to provide a mechanism for the interchange of messages between the service requestor and the discovery agent. This role gets the input messages from the user and submits it to the message handler for further processing. This role helps in obtaining the agent's social behavior property. The agent can able to communicate with the environment (in this case the user) using this role and hence the social ability property of the agent is present in this role. Figure 6 gives the role schema for communication handler.

Role Schema: Communication Handler
Description: This role acts as a communication media. It passes the incoming and the outgoing messages.
Protocol and Activities: ReceiveMessages SubmitMessages
Permissions: Pass Received messages
Responsibilities: Liveness: (ReceiveMessages . SubmitMessages) + Safety: Messages arrived # Null

Figure 6: Role Schema for Communication Handler

Role Schema for Message Handler

The correct interpretation of the user's request is handled by this role by handling different types of user interactions. This role possesses the social ability property of the agent. It is used for parsing the incoming and the outgoing messages based on the type of the request interacting with the service discovery agent. The users requesting the services may be an agent, other web services or a software system. These systems may interact with the discovery agent by KQML/ACL, SOAP or Natural Languages (NL) respectively. Message handler role should contain three types of handlers for handling the messages from different users. Figure 7 gives the role schema for Message handler. The functions provided by this message handler role is

- Conversion of ACL/KQML to WSDL and vice versa.
- Handling the SOAP messages
- Conversion of NL request into WSDL format and generating the NL messages from WSDL.

Role Schema: Message Handler
Description: This role handles the different types of the messages by parsing them and submits it for the match making process.
Protocol and Activities: ReceiveUserRequest SubmitResult ConvertRequest ConvertResult EncryptResult EncryptRequest
Permissions: Read Request query, Result, Security policy Change Result format // encrypt Request format // decrypt
Responsibilities: Liveness: (ReceiveUserRequest . [DecryptRequest] . ConvertRequest) + (ConvertResult . [EncryptResult] . SubmitResult) + Safety: Messages arrived # Null

Figure 7: Role Schema for Message Handler

Role Schema for Repository

This role is used to provide a quick reference to the recent responses sent by the agent in reply to the discovered services. Repository contains only a subset of the services registered in UDDI registry. Figure 8 gives the role schema for repository.

Role Schema: Repository
Description: This role acts as a repository for quick reference.
Protocol and Activities: ReceiveSelectionRequest

ReceiveSelectionResult ProvideResult UpdateRepository QueryRepository
Permissions: Read Request query, Selection Result Update Repository contents
Responsibilities: Liveness: (ReceiveSelectionRequest . QueryRepository . ProvideResult) + (ReceiveSelectionResult . [UpdateRepository] + Safety: Request for selection query # Null Result of the selection # Null

Figure 8: Role Schema for Repository

Role Schema for Security

Security role of the agent helps in providing the services in a secure manner. This role is also used for encrypting and decrypting the incoming and the outgoing messages. Figure 9 gives the role schema for security.

Role Schema: Security
Description: Provides security mechanism by encrypting and decrypting the outgoing and the incoming messages respectively.
Protocol and Activities: ProvideSecurityMechanism
Permissions: Read Request query, Result, Securitypolicy
Responsibilities: Liveness: (ProvideSecurityMechanism)* Safety: Security Request # Null

Figure 9: Role Schema for Security

Identification of Application Dependent Roles

The application dependent roles of the domain-specific software architecture for service discovery agent are identified by analyzing the features of the application in the domain under consideration. The definitions of these roles vary with respect to the application under development in service discovery domain.

The following are the application dependent roles for the service discovery agent identified in the domain engineering phase:

- Match Maker
- Context handler
- Domain ontology
- User Profile handler

Role Schema for Match Maker

This role is used for making the matching process of the request queries. Figure 10 shows the role schema for match maker. The following are the functions identified for accomplishing the matching processes:

- *Domain filteration:* For selecting the application domain.
- *Similarity calculation:* Calculates the percentage of matches among the results provided by the domain filtration. This process arranges the results of its processing in the decreasing order of the calculated percentage (Best match will have high percentage).
- *Selection:* This function is used for the final selection processes. The final results are selected with respect to the best matches based on the results provided by the similarity calculation.

Role Schema: Match Maker
Description: Gets the input query from message handler and finds the match, updates relevant entries and sends the result to the user through message handler.
Protocol and Activities: ReceiveRequest, CheckRepository, SearchUDDI, CalculateSimilarityPercent, ReferContextualInformation, SelectResult, PassResultToProfileHandler UpdateRepository , SendResult
Permissions: Read Customer query, Contextual information Update Repository Query Repository, UDDI Registry
Responsibilities: Liveness: (ReceiveRequest . CheckRepository . [SearchUDDI] . CalculateSimilarityPercent . ReferContextualInformation . SelectResult . PassResultToProfileHandler . [UpdateRepository] . SendResult) + Safety: Request query # Null

Figure 10: Role Schema for Match Maker

Role Schema for Context Handler

This role helps in providing a reasoning mechanism for handling the contextual information needed for the selection process. Context handler role uses the domain ontology and the user profile handler for its processing. The role schema for context handler is given in Figure 11.

Role Schema: Context Handler
Description: Provides contextual information for the selection process with the help of domain ontology and user's preferences.
Protocol and Activities: ReceiveQuery, ReferDomainOntology, ReferUserProfile MakeDecision, ProvideDecisionResult

Permissions:
Read Domain ontology, User Profile, Request
Responsibilities:
Liveness:
(ReceiveQuery . ReferDomainOntology . [ReferUserprofile] . MakeDecision . ProvideDecisionResult) +
Safety:
Request received # Null

Figure 11: Role Schema for Context Handler

Role Schema for Domain Ontology

This role maintains the inner details of the application domain. The domain information present in the domain ontology helps the context handler to carry out the reasoning mechanism. Figure 12 gives the role schema for Domain Ontology.

Role Schema: Domain Ontology
Description:
Provides the application domain ontology.
Protocol and Activities:
ReceiveRequestQuery
ProvideDomainOntology
Permissions:
Read Request Query
Responsibilities:
Liveness: (ReceiveRequestQuery . ProvideDomainOntology)+
Safety: Request received # Null

Figure 12: Role Schema for Domain Ontology

Role Schema for User Profile Handler

This role is used for maintaining the preferences of the user. When the preference of the user is changed with respect to the query for the selection process, the profile handler undergoes an updation process. Figure 13 gives the role schema for profile handler.

Role Schema: Profile Handler
Description:
Contains the updated information of the user and is referred by context handler.
Protocol and Activities:
UpdateProfile
GetSelectionResult
CheckEntries
Permissions:
Read SelectionResults
Update User Profile
Responsibilities:
Liveness: (GetSelectionResult . CheckEntries . [UpdateProfile])+
Safety: Selection results # Null

Figure 13: Role Schema for Profile Handler

Identification of Role Interactions

The dependencies and the relationships among the roles identified in the previous sections for building the domain-specific architecture for service discovery agents are explained in the interaction model. These interactions are described in terms of the protocols associated with each role model.

The protocol description for communication handler component is given in figure 14 and 15. Similarly, the protocols for other components (Domain-dependent and application dependent) were defined which is not given in this paper.

Protocol associated with Communication Handler

The protocols associated with Communication Handler role are *ReceiveMessages* and *SubmitMessages*.

ReceiveMessages: This protocol interacts with the user for receiving their request and passes these requests to the message handler for parsing them. The protocol description for ReceiveMessages is given in Figure 14.

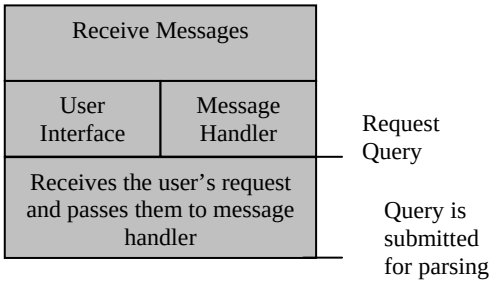

Figure 14: Protocol Description for ReceiveMessages

SubmitMessages: This protocol gets the result message from message handler and submits them to the user. Figure 15 shows the protocol description for SubmitMessages.

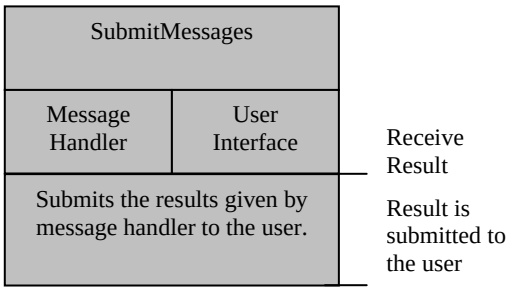

Figure 15: Protocol Description for SubmitMessages

Design Model for Service Discovery Agent

This section transforms the analysis model into sufficiently low level of abstraction in order to implement agents. This design process involves generating three models:

- Agent Model

- Services Model and
- Acquaintance Model

Agent Model: The purpose of agent model is to document the various agent types that will be used in the system under development and agent instances that will realize these agent types at run-time.

Figure 16 shows the service discovery agent model. The service discovery agent plays various roles like communication handler, message handler, security, repository, match maker, user profile, context handler and domain ontology.

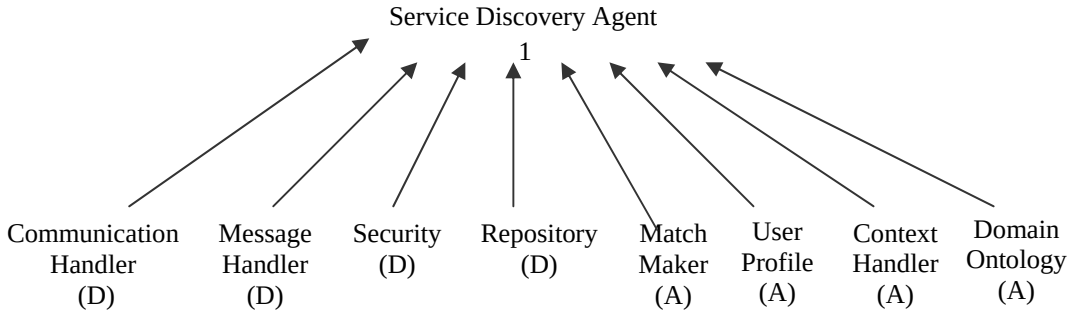

Figure 16: Service Discovery Agent Model

Services Model: The aim of the Gaia services model is to identify the *services* (functions) associated with each agent role, and to specify the main properties of these services. The properties of each service like inputs, outputs, pre-conditions and post-conditions must be identified. Every activity identified at the analysis stage will correspond to a service but every service will not correspond to an activity.

Acquaintance Model: Acquaintance models define the communication links that exist between agent types. These models do not define what messages are sent or when the messages are sent. It only indicates the communication pathways existing between the agents. Figure 17 gives the acquaintance model for service discovery agent.

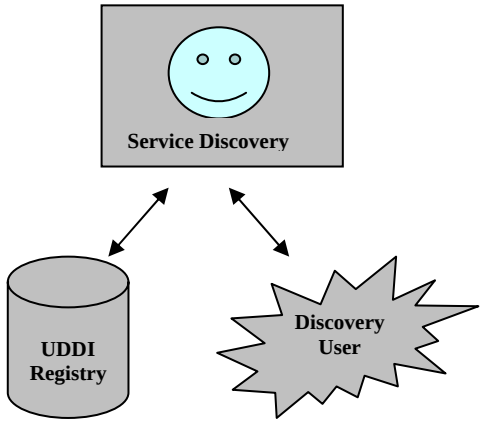

Figure 17: Acquaintance Model for Service Discovery agent

Reference Architecture

The domain engineering phase identified the domain dependent roles, the application dependent roles and their interactions. The roles and interactions are well defined in terms of Role Schemas and the protocol definitions for roles and interactions respectively. The next phase in domain-specific software architecture for software agents is the creation of the reference architecture.

Each role of service discovery agent, identified in the domain engineering phase corresponds to a single component (ie) there is one-to-mapping exists between the roles and the components of the service discovery agent.

This subsection describes the reference architecture for web service discovery agent. Figure 18 shows the arrangements of the identified components in the reference architecture.

External user interacts with the service discovery agent by submitting their requests through communication component. This component then submits the request to the message handler, where the request is parsed with respect to the type of the user. If the user interacting the communication component is an agent, then the request are parsed with the help of the ACL/KQML handler, if it is other web service then the SOAP handler takes the responsibility of parsing and if it is a normal user program, then the NL interaction handler is used for converting the request to the format used for discovering the service by the match maker component. The message handler also uses the security component for decrypting the input messages from the external user. After processing the request it then passes it to the match maker component.

The match maker component after receiving the parsed request from the message handler, checks it with the repository which maintains a subset of the services present in the UDDI registry (only frequently accessed resources are available in UDDI registry). If it matches with the entries in the repository then the similarity calculator is used for calculating the percentage of the matching, for this it requests the information from the context handler which uses the domain ontology and the user preferences. The selection component then selects the best matched results and submits them to message handler. The user preference handler is also updated with the preferences of the recently accessed request.

If the repository does not contain the request made by the user, then the match maker component uses its domain filter component for discovering the service from the UDDI registry. The whole process as described above is repeated.

The message handler after receiving the results of the selection process from the match maker converts the results into a format as required by the user.

It uses the security component for encrypting the result if necessary and passes them to the communication component which then sends back to the user.

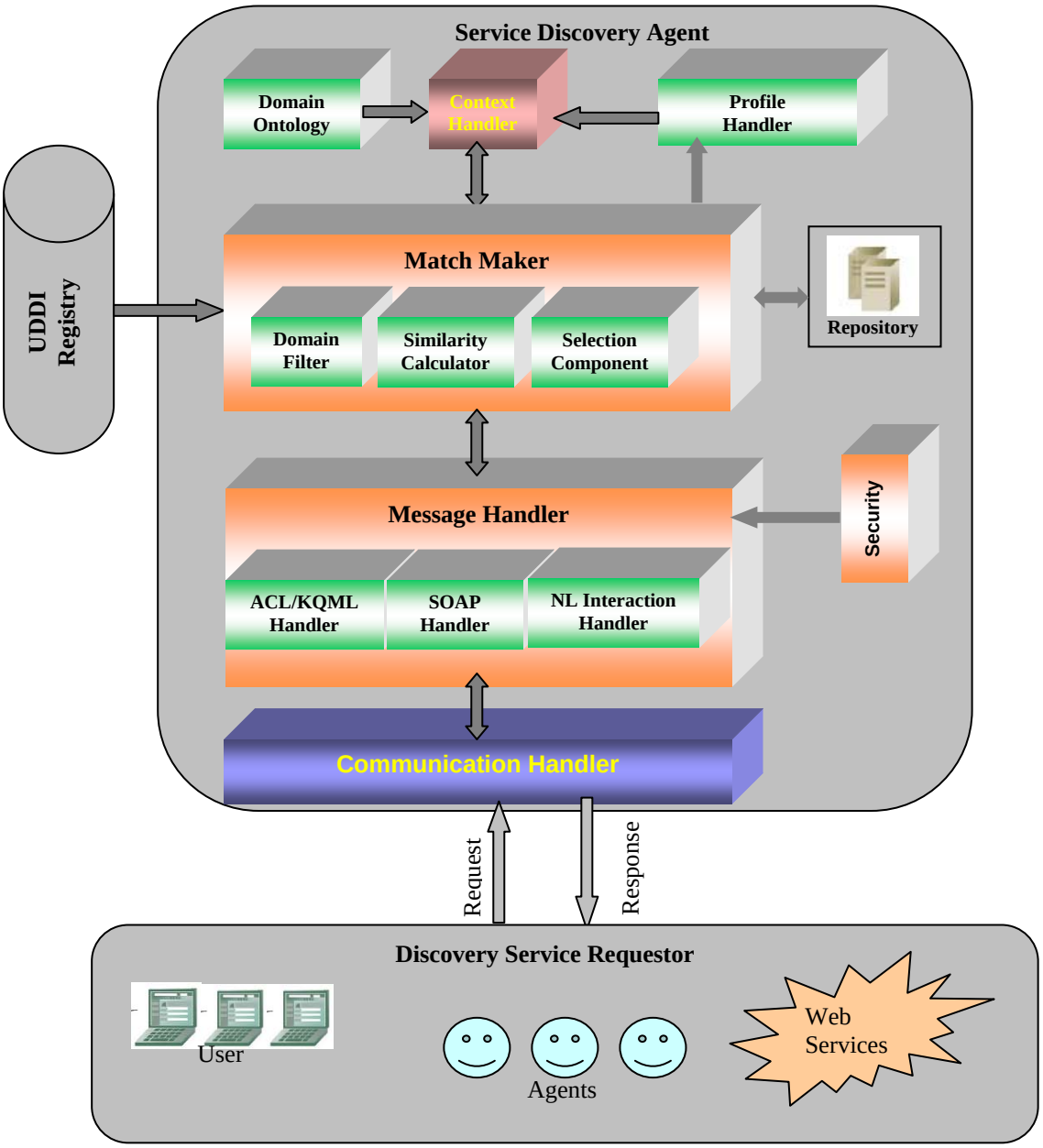

Figure 18: Reference Architecture for Web Service Discovery agent

6 EVALUATION OF THE PROPOSED ARCHITECTURE

The proposed architecture is evaluated by tailoring the Software Architecture Analysis Method (SAAM). All the scenarios corresponding to each applications of the service discovery are listed and evaluated. The steps corresponding to SAAM for evaluating the domain-specific architecture are described as follows:

Scenario Development

The Scenarios chosen for evaluation of the architecture are given below (only the indirect scenarios are shown here):

- Scenario 1: The response time of the system should be less.
- Scenario 2: Degraded operation mode.

- Scenario 3: Language neutral
- Scenario 4: Support for easy up gradation
- Scenario 5: Integrate with new development environment.
- Scenario 6: Support of agent's properties.
- Scenario 7: Support for other of agent architectures.
- Scenario 8: Ability to discover the agents that provide the required services in a Multi Agent System environment.
- Scenario 9: Ability to incorporate other functional behaviors that rely on service discovery.

Scenario Classification

The Scenarios are classified as direct or indirect with respect to the supportability of the scenario in the Reference architecture shown in Figure 18. As only indirect scenarios are taken here for evaluation of the architecture, each scenario is evaluated in the next section.

Scenario Evaluation

Each of indirect scenarios is evaluated as follows:

Scenario 1: This scenario requires that the user should get the responses to their query in a fast manner. To provide for this the repository component which helps to store the frequently/recently accessed queries are available instead of contacting to the UDDI registry with takes some delay.

Scenario 2: This scenario can be achieved by the use of the repository component. The repository component contains the backup of the frequently / recently accessed queries. When there is a failure in the UDDI server or in the network, the user can get the responses to their queries with the help of the repository if the query requested by the user is available in the repository. This architecture does not degrade the operation fully; it supports the user to continue their work even when there is failure in the network or in UDDI registry.

Scenario 3: This scenario requires that the architecture should be implemented in any language. It can be easily supported as the proposed architecture does not force any language specifications for the components.

Scenario 4: This scenario aims in the inclusion of any new components in the architecture. This scenario can be easily achieved as the components in the architecture communicate only through messages. This helps in the addition of any new components in the architecture or the addition of any new subcomponents within a single component. The subcomponents within a component are highly cohesive and the components have less coupling among them. Hence this helps in the addition or removal of a new single component or a set of components along with its sub components easily.

Scenario 5: As the architecture describes the components in domain dependent manner, any application pertaining to that domain can be implemented easily.

Scenario 6: This scenario requires the architecture to possess the agent's properties like autonomy, reactivity, proactivity, social ability and adaptability. The

components described in the architecture possess these properties. For example, the autonomy property of the agent is present in the match maker and the context handler component for taking the decision in the matching process. The *reactivity* property is present in the match maker component. It responds to the users query in a reactive manner. The *social ability* property is taken care with the help of the message handler component. The architecture is adaptable to the user's service preferences thus it supports the *adaptability* property.

Scenario 7: As the architecture gives the roles of the agents, these roles can be implemented with the help of any agent architecture like BDI, Reactive or Hybrid architecture.

Scenario 8: This scenario requires the architecture to be used in MAS environment. Since the architecture can be used to find any agents in DF which provides the required service in MAS, this request is also fulfilled.

Scenario 9: The interfaces provided between the components are more flexible and hence this architecture requires only minimal changes for the support of this scenario. The concrete agent architecture developed can also be integrated or extended into any applications like brokery agent, buying agent or any other e-commerce applications that need service discovery as part of their functionality without much changes in the existing components.

Scenario Interaction

Agent's properties are mainly supported with the help of the match maker, context handler, user profile handler, message handler and communication components.

Table: 2 gives the list of scenarios which affects the component description. Figure 19 gives the scenario interaction chart corresponding to Table 2. The vertical axis of figure 19, depicts the number of scenarios affected by each component in the horizontal axis.

Components	Scenario affecting the component
Communication Handler	Scenario 6, Scenario 9
Message Handler	Scenario 6
Security	-
Repository	Scenario 1, Scenario 2
Match Maker	Scenario 6
Context Handler	Scenario 6
User Profile	-
Domain Ontology	-

Table 2: Scenarios affecting the components

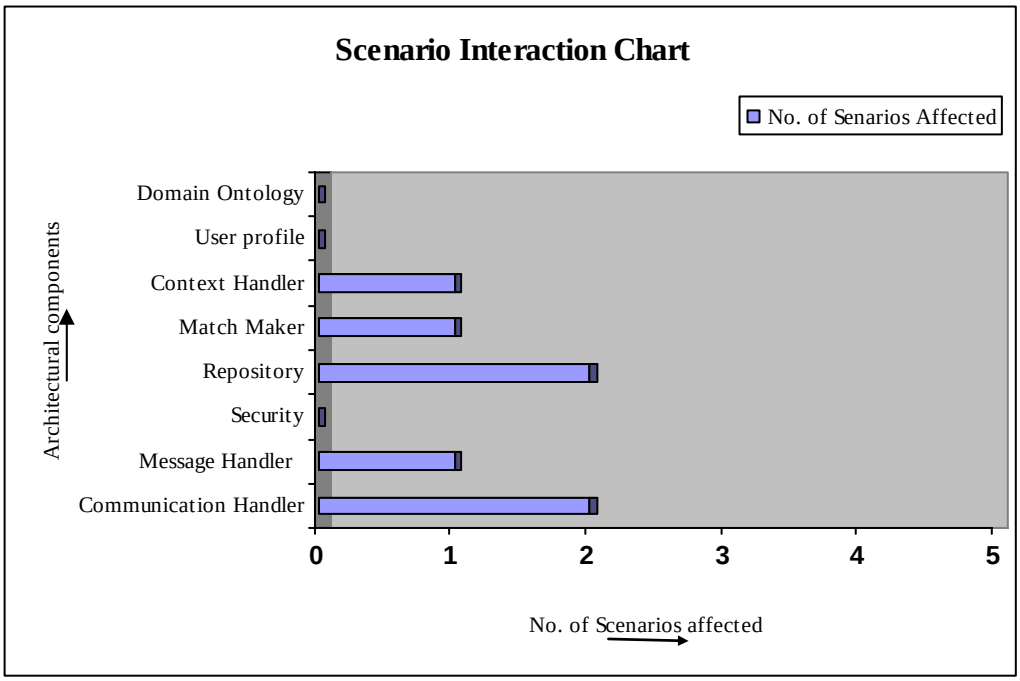

Figure 19: Scenarios Interaction

Scenario Evaluation

The scenario interaction reveals that the number of components affected by each scenario is not more than two. This shows that the applicability of the proposed architecture to various applications does not need much change. Hence the proposed architecture is more flexible. The maintainability of the architecture is also high. All these qualities prove that the proposed architecture is having high quality.

7 CONCLUSION

Agents are widely used in many domains. Hence in order to reduce the development time of agents pertaining to a domain, reusability of architecture becomes an absolute necessity. DSSA is a field of study which focuses upon developing domain-specific architecture which could be reused. Developing DSSA for agents has not received enough focus in the agent research. And the available works are of preliminary in nature. Hence neither a methodology nor a complete DSSA for agents is available.

In this paper, a new methodology for developing a DSSA for agents has been proposed. Using this methodology the development of DSSA for agents pertaining to service discovery domain has been described. The evaluation of the methodology as well the resulting architecture gives promising results in that a complete methodology and a well defined DSSA for software agents have been obtained as an outcome.

REFERENCES

[Ahmed 00] Ahmed E. Hassan and Richard C. Holt, "A Reference Architecture for Web Servers, IEEE, 2000

[Alfanet 01] ALFANET, "Deliverable D31 –Existing Standards Analysis", Active Learning for Adaptive Internet, 2001

[Baojin 99] Baojin Li Guagzhou Zeng Zongkai Lin, "A Domain Specific Software Architecture", ACM, 1999.

[Chia-En 07] Chia-En Lin, Krishna M.Kavi, Pokahr and Winfried Lamersdorf, Frederick
T. Sheldon, Kris M. Daley and Robert K. Abercrombie, "A Methodology to Evaluate Agent-Oriented Software Engineering Techniques", Proceedings of the 40th Hawaii International Conference on System Sciences, 2007

[Craig-1] Craig Thompson, Strawman, "Agent Reference Architecture", http://www.objs.com/agility/tech-reports/9808-agent-ref-arch-draft3.ppt.

[Craig-2] Craig Thompson, "Agent Technology White Paper and RFP Roadmap", OMG Agent Working Group, 2000

[Craig-3] Craig Thompson, Tom Bannon, Paul Pazandak, Venu Vasudevan, "Agents for the Masses", Object Services and Consulting, Inc. http://www.objs.com

[Davide 00] Davide Brugali, Politecnico di tirino Katia Sycara, "Towards Agent Oriented Application Frameworks", ACM, 2000.

[FIPA] Foundation for Intelligent Physical Agents, "Fipa Abstract Architecture Specification", 1993 http://www.fipa.org/

[Jeffrey 97] Jeffrey M. Bradshaw, Stewart Dutfield, Pete Benoit and John D. Wolley, "KAoS: Toward An Industrial-Strength Open Agent Architecture", AAAI Press, 1997

[Joaq-1 06] Joaquin Pena, Michael G. Hinchey and Antonio Ruiz Cortes, "Building the Core Architecture of a Multiagent System Product Line: With an example from a future NASA Mission", 2006.

[Joaq-2 06] Joaquin Pena, Michael G. Hinchey and Antonio Ruiz Cortes, "Multi-Agent System Product Lines: Challenges and Benefits", ACM, 2006.

[Joaq-3 06] Joaquin Pena, Michael G. Hinchey, Manuel Resinas, Roy Sterritt and James L. Rash, "Managing the Evolution of an Enterprise Architecture using a MAS-Product-Line Approach", 2006

[Jose 02] Jose M. Vidal, Paul Buhler, "A Generic Agent Architecture for Multiagent Systems", USC CSCE, 2002

[Josh 05] Josh Dehlinger, Robyn R. Lutz, "A Product-Line Requirements Approach to Safe Reuse in Multi-Agent Systems", ACM, 2005

[Klaus 05] Klaus Bergner, Andreas Rausch, Marc Sihling, Thomas Ternite, "DoSAM – Domain-Specific Software Architecture Comparison Model", 2005

[Marius 05] Marius, Lucian-lonel, "Multi-criterion Analysis of Reference Architectures and Modeling Languages used in Production Systems Modeling, IEEE, 2005

[Mugurel 03] Mugurel T. Ionital, Deiter K. Hammer, Henk Obbink, "Scenario-Based Software Architecture Evaluation Methods: An Overview", Technical University, Eindhoven, 2003

[Pragnesh 06] Pragnesh Jay Modi, Spiros Mancoridis, William M. Mongan, William Regli, Israel Mayk, "Towards a Reference Model for AgentBased Systems", ACM, 2006

[Rem 02] Rem William Collier, "Agent Factory: A Framework for the Engineering of Agent-Oriented Applications", 2002

[Richard 95] Richard N.Taylor, Will Tracz, Lou Coglianese, "Software Development Using Domain-Specific Software Architecture", ACM, 1995

[Roberto] Roberto A. Flores-Mendez, "Towards a Standardization of Multi-Agent System Frameworks"

[Sankar 95] Sankar Virdhagriswaran, Damian Osisek and Pat O'Connor, "Standardizing Agent Technology", OMG 1995.

[Steven 02] Steven P. Fonseca, "An Internal Agent Architecture for Dynamic Composition of Reusable Agent Subsystems – Part 1: Problem Analysis and Decomposition Framework", Hewlett Packard Company, 2002

[Sturm 03] Sturm, A and Shehory O, "A Framework for Evaluating Agent-oriented Methodologies", 5th Int'l Bi-Conference Workshop on Agent-Oriented Info Sys (AOIS), Springer LNCS 3030, 2003

[Sudeikat] Jan Sudeikat, Lars Brauback, Alexander Pokahr and Winfried Lamersdorf, "Evaluation of Agent-Oriented Software Methodologies – Examinnation of the Gap Between Modeling and Platform"

[Tracz 95] Will Tracz, "DSSA (Domain-Specific Software Architecture) Pedagogical Example", ACM, 1995.

[Wood 01] Wood, MF and DeLoach, SA. "An Overview of the Multiagent Systems Engineering Methodology in Agentoriented Software Engineering," First Int'l Wkshp (AOSE 2000) on Agent-oriented Software Engineering, Springer-Verlag New York (LNCS 1957), 2001, pp. 207-222

[Woold 00] Wooldridge, M, Jennings, NR and Kinny, D. "The Gaia Methodology for Agent-oriented Analysis and Design," *Autonomous Agents and Multi-Agent Systems Jr., Kluwer Academic Publishers.*, 2000(3): 285-312.

About the authors

Miss A. Suganthy is studying M.Tech in Computer Science and Engineering (2005-2007 batch) in Department of Computer Science, Pondicherry University, Pondicherry. Email: asugan@gmail.com

Ms. T. Chithralekha is currently working as a Reader in Dept. of Banking Technology, Pondicherry University. She has completed her M.Tech(Computer Science & Engg.) from Dept. of Computer Science, Pondicherry University. Her area of research pertains to Software Agents and Multilingual systems. She has published about 15 papers in Conferences and Journals. Her area of specialization includes Distributed Systems, Agent Technology, Information Security and Data Warehousing.

Email: tchitu@yahoo.com