A Possibilistic Petri-Nets-Based Service Matchmaker for Multi-Agents System Architecture

Jonathan Lee, Yao-Chiang Wang, Chia-Ling Wu, Shin-Jie Lee, Shung-Pin Ma and Whan-Yo Deng

Abstract

The focus of this paper is to devise a service-oriented architecture for multi-agents system (called SAM), to reach a wide adoption and routine use of agents, web services, and semantic web technologies. In order to better integrate agents, web applications and information sources in an open and distributed environment such as the Internet, a service matchmaker called middle agent is responsible for registering and spotting services in SAM. In the making of the service matchmaker there are four components involved: (1) a possibilistic Petri-nets (PPN) based service matchmaking mechanism for discovering services, (2) a service description language based on PPN for describing agent services and web services, (3) a set of ontologies for facilitating the semantic matchmaking, and (4) newly-defined extension mechanisms for bridging the linkage between web service standards and the service matchmaker through a translator.


1. Introduction

It is widely recognized that multi-agent systems (MASs) have attracted increasing attention not only as a scientific discipline, but more importantly as a software engineering paradigm for building commercially viable and innovative applications [12, 16, 21, 25]. Consequently, a considerable number of researches have reported progress towards the formation of theories on the modeling of MAS infrastructures. (e.g. IMPACT [3], InfoSleuth [7], RETSINA [25], OAA [20], DECAF [13] and etc.)

However, several key challenges remain to be addressed in order to reach the wide adoption and routine use of agent technology in the Internet - an open environment that serves as one of the main pushers behind the research and development of MAS and web applications:

- How to facilitate the discovery of Internet-based services? The amount of Internet-based services, including deployed software agents and web services, is increasing in an exponential fashion. Moreover, these services together with their communication links and information sources may appear and disappear dynamically in the Internet.

- How to increase the interoperability among Internet-based services? With the advent of web infrastructure via the development of various web service standards such as SOAP (Simple Object Access Protocol) [5], WSDL (Web Service Description Language) [8], and UDDI (Universal Description, Discovery and Integration) [4], different service applications and information sources can be composed independently. Through this interoperability, businesses should be able to utilize and compose these services more flexibly and dynamically in performing their Internet-based services.

In this paper, we propose a Service-oriented Architecture for Multi-agents (called SAM) as an attempt towards the construction of an MAS infrastructure that can better integrate agents, web applications and information sources. We begin with the devise of a service matchmaker based on the formation of possibilistic reasoning [18], and an extension of our previous work on a PPN-based agent service description language (PPN-ASDL) [19] by means of DAML-based ontology and services description, then proliferate to the integration of web infrastructure with the SAM-ASDL translator that utilizes our newly-defined tModels for UDDI registry, and finally conceive the notion of composite service agents capable of the composition of agents, web applications and information resources as an experimental scenario.

The remainder of this paper is organized as follows. Numbers of infrastructures and mechanisms designed for service matching and heterogeneous environments
integrating of various multi-agent systems are outlined and compared in the next section. In section 1, the proposed multi-agent system, namely SAM, and its constituents are briefly introduced. In section 4, the making of a PPN-based service matchmaker, which is built inside a middle agent in SAM, is delineated in detail. In section 5, the problem domain of advanced traveling information system is used as an illustrative example to demonstrate the proposed approach. Finally, we conclude and give future plans for extending the work of service matchmaking in SAM.

2. Related Work on MAS Infrastructures

Recently, a significant number of MAS infrastructures have been proposed to tackle various issues encountered in their efforts of advancing agent technology. The focus of each MAS varies, including information gathering, collaboration, and etc.. We have identified not only the focus of each MAS, but also key features, especially on service matchmaking and interoperability mechanisms, that are useful in distinguishing these MASs: agent types, services types, service description language (SDL), mediation protocol, matching algorithm, query processing, matching facilities and heterogeneous interoperability. Before looking into the detail comparisons in table 1, we briefly examine each MAS at first.

Table 1. Comparison between Service Matching and Interoperability Mechanisms on MASs

<table>
<thead>
<tr>
<th>Service Types</th>
<th>Agent Services</th>
<th>Agent Services</th>
<th>Agent Services</th>
<th>Agent Services</th>
<th>Agent Services</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAM</td>
<td>Legacy Services</td>
<td>Legacy Services</td>
<td>Legacy Services</td>
<td>Legacy Services</td>
<td>Legacy Services</td>
</tr>
<tr>
<td>Infolith</td>
<td>WWW</td>
<td>WWW</td>
<td>WWW</td>
<td>WWW</td>
<td>WWW</td>
</tr>
<tr>
<td>OAA</td>
<td>WWW</td>
<td>WWW</td>
<td>WWW</td>
<td>WWW</td>
<td>WWW</td>
</tr>
<tr>
<td>DECAF</td>
<td>WWW</td>
<td>WWW</td>
<td>WWW</td>
<td>WWW</td>
<td>WWW</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Middleware Infrastructure</th>
<th>middle agents</th>
<th>Agent Name Services</th>
<th>Agent Name Services</th>
<th>Agent Name Services</th>
<th>Agent Name Services</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMPACT</td>
<td>LARKS</td>
<td>Yellow-pages</td>
<td>Broker Agents</td>
<td>Broker Agents</td>
<td>Broker Agents</td>
</tr>
<tr>
<td></td>
<td>HTML-like</td>
<td>Services</td>
<td>Task Planning Agents</td>
<td>Ontology Agents</td>
<td>Ontology Agents</td>
</tr>
<tr>
<td></td>
<td>language</td>
<td>registration</td>
<td>Servicers</td>
<td>Facilitator Agents</td>
<td>Facilitator Agents</td>
</tr>
<tr>
<td></td>
<td></td>
<td>servers</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Matching Algorithms</th>
<th>Functional match</th>
<th>Functional match</th>
<th>Functional match</th>
<th>Functional match</th>
<th>Functional match</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Non-functional match</td>
<td>Syntactical match</td>
<td>Syntactical match</td>
<td>Syntactical match</td>
<td>Syntactical match</td>
</tr>
<tr>
<td></td>
<td>(rule preferences)</td>
<td>Semantic similarity</td>
<td>Semantic similarity</td>
<td>Semantic similarity</td>
<td>Semantic similarity</td>
</tr>
<tr>
<td></td>
<td></td>
<td>match</td>
<td>match</td>
<td>match</td>
<td>match</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(context)</td>
<td>(context)</td>
<td>(context)</td>
<td>(context)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(context)</td>
<td>(context)</td>
<td>(context)</td>
<td>(context)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Matching Features</th>
<th>Functional match</th>
<th>Functional match</th>
<th>Functional match</th>
<th>Functional match</th>
<th>Functional match</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Non-functional match</td>
<td>Syntactical match</td>
<td>Syntactical match</td>
<td>Syntactical match</td>
<td>Syntactical match</td>
</tr>
<tr>
<td></td>
<td>(rule preferences)</td>
<td>Semantic similarity</td>
<td>Semantic similarity</td>
<td>Semantic similarity</td>
<td>Semantic similarity</td>
</tr>
<tr>
<td></td>
<td></td>
<td>match</td>
<td>match</td>
<td>match</td>
<td>match</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(context)</td>
<td>(context)</td>
<td>(context)</td>
<td>(context)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(context)</td>
<td>(context)</td>
<td>(context)</td>
<td>(context)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Heterogenous Interoperability</th>
<th>Agent Name Services</th>
<th>Agent Name Services</th>
<th>Agent Name Services</th>
<th>Agent Name Services</th>
<th>Agent Name Services</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(for interacting with legacy systems)</td>
<td>(for interacting with legacy systems)</td>
<td>(for interacting with legacy systems)</td>
<td>(for interacting with legacy systems)</td>
<td>(for interacting with legacy systems)</td>
</tr>
<tr>
<td></td>
<td>WWW</td>
<td>WWW</td>
<td>WWW</td>
<td>WWW</td>
<td>WWW</td>
</tr>
<tr>
<td></td>
<td>SAM-ASDL</td>
<td>SAM-ASDL</td>
<td>SAM-ASDL</td>
<td>SAM-ASDL</td>
<td>SAM-ASDL</td>
</tr>
<tr>
<td></td>
<td>translator</td>
<td>translator</td>
<td>translator</td>
<td>translator</td>
<td>translator</td>
</tr>
<tr>
<td></td>
<td>(for consuming Web services)</td>
<td>(for consuming Web services)</td>
<td>(for consuming Web services)</td>
<td>(for consuming Web services)</td>
<td>(for consuming Web services)</td>
</tr>
</tbody>
</table>

The RETSINA [25, 26] (Reusable Environment for Task Structured Intelligent Networked Agents) multi-agent infrastructure has been developed by the Intelligent Agents Group at Carnegie Mellon University, USA. It is an open multi-agent system that provides infrastructure for different types of deliberative, goal directed agents: Interface agents, task agents, middle agents and information agents. Agents can communicate and interact among themselves with the infrastructure components, such as MAS management services, performance services, security services, agent name services (ANS), matchmaker, and RETSINA-OAA interoperator. Each agent is composed of four autonomous functional modules: a communicator, a HTN (hierarchical task network) planner, a scheduler and an execution monitor. The four modules of a RETSINA agent are implemented as autonomous threads of control to allow concurrent planning and actions scheduling and execution.

RETSINA treats individual agents as "socially aware" programs that communicate, interact among themselves with the infrastructure components, such as ACL infrastructure, MAS management services, performance services, security services, agent name services (ANS), matchmaker, and RETSINA-OAA interoperator. RESINA explicitly distinguishes between the two services: ANS and matchmaker. The matchmaker maps capabilities into agents, and the ANS maps agents to locations. Decoupling the two services is justified by the fact that agent discovery by name is a system-level service, whereas capability-based lookup is a service at the knowledge level.

To describe agents' capabilities in the matchmaking process, RETSINA have defined and implemented an ACDL (Agent Capability Description Language), called LARKS (Language for Advertisement and Request for Knowledge Sharing) [27, 28]. LARKS is wrapped in a KQML-like message and is a frame-like specification with the following slots: context keywords, variable types, input and output variables, logical constraints on input and output variables, and ontological descriptions of keywords. RETSINA with LARKS contains five different filters to take both syntactic and semantic matching into account.

The IMPACT (Interactive Maryland Platform for Agents Collaborating Together) [2] platform has been developed at the university of Maryland, College Park, USA. The principal goal of the IMPACT project is to develop both a theory and a software implementation that facilitates the creation, deployment, interaction, and collaborative aspects of software agents in a heterogeneous, distributed environment. For this purpose IMPACT provides following set of services as an infrastructure upon which different IMPACT agents may
interact with each other:

- Yellow-pages server performs basic matchmaking among service requesters and service providers based on two weighted hierarchies it maintains - a verb and a noun hierarchy of synonyms - and retrieval algorithms to compute similarities between given service specifications,
- Registration server for agent registration and maintaining an agent service index used by the yellow-pages server,
- Type server maintains a hierarchical ontology of standard data types and concepts, and
- A thesaurus and a human interface allowing a human to access all the above servers in an IMPACT system.

Any service provided by an IMPACT agent is specified in an HTML-like service description language. Agents may request services in one of the following forms: (1) k-nearest neighbor request: find the k-nearest service names such that there exists an agent who provides that service and identify this agent, and (2) d-range search: find all service names within a specified distance d. Therefore the Yellow-pages server identifies agents that provide services similar to a requested service. Similarity is computed through two technical means - through a certain kind of "k-nearest neighbor" search in a metric space, and through certain "range" queries in a metric space.

InfoSleuth [22] is an agent-based system for information discovery and retrieval in a dynamic, open environment such as the World Wild Web. It has been developed by MCC Inc. in Austin, Texas, USA. The InfoSleuth system consists of a set of collaborating agents that work together, and focuses on:

- gathering information via complex queries from a changing set of databases distributed across an internet,
- performing polling and notification for monitoring changes in data, and
- analyzing gathered information using statistical data mining techniques and/or logical inferencing.

The InfoSleuth agents are organized as core agents that provide basic information subscription, filtering and fusion capabilities, resource agents that serve as interface to external information resources, and user agents that act as proxies for individual users. The core agents (broker agent, task planning agent, multiresource query agent, data mining agent, ontology agent) represent the set of agents that work together to connect users with the information they need. These agents service requests over a set of common ontologies, accessed via the Ontology agents.

The Open Agent Architecture (OAA) [20], developed and used for several years at SRI International, USA, provides an agent architecture that can facilitate communication and cooperation between agents by one or more centralized facilitators. OAA uses a PROLOG-based ICL (Interagent Communication Language) for describing agents' needs and capabilities. The language allows service providers to describe the agents in terms of tasks they can perform and not really in terms of resources they represent. Based on ICL, the facilitator agent will make decisions about which agents are available and capable of handling sub-parts of the request, and will manage all agent interactions required to handle the complex query. The facilitator may also provide a global data store to adopt a blackboard style of agent interaction. Furthermore, OAA uses triggers to instantiate commitments within and between agents, and provides a agent library to minimize the effort involved in creating new agents and wrapping legacy applications.

DECAF (Distributed Environment Centered Agent Framework) [13], developed by University of Delaware, USA, provides a platform for extremely rapid development of agents, following a well-defined software engineering approach. This is accomplished by building an operating environment that provides an interface, internal agent scheduling and monitoring in a fashion similar to operating system primitives. The agent developer does not need knowledge of any of this structure and can thus focus on development of the agent itself. The basic DECAF architecture has been built using the Java programming language, and a set of Java classes are implemented, such as communication, coordination and reasoning services.

DECAF provides programmers with a GUI interface, called Plan-Editor, to model executable actions as basic building blocks which can be chain together in the style of an HTN (hierarchical task network). The output of the Plan-Editor is the plan file, which represents the programming of the agent. One agent consists of a set of capabilities and a collection of actions that may be planned and executed to achieve the objectives. DECAF uses RETSINA-style automated data flow for directing plan execution. Furthermore, DECAF helps in prototyping MAS by providing standardized, domain-independent middle agents, including agent name servers (ANS), matchmakers (yellow pages), brokers (white pages), web proxies, and agent management agents. DECAF also provides some value-added features to the programmers such as ANS registration, behavior libraries to interact with middle-agents.

In table 1, we summarize the comparison between MASs with a list of criteria based on the previous examinations. The detail descriptions of these criteria are as follows:

- Agent Types: The core of an MAS is various
agents reside in. Some MASs will further classify these agents into different types or roles by separating some complex tasks for agents to perform. For example, service agents, request agents, and middle agents can be defined from views of different roles involved in a service lifecycle.

- Service Types: A service can be a task or a resource delivered by a network node which is needed by others in an open environment. Although software agents are the main sources of theses services in an MAS, services can still be classified into different types according to the technologies used in different systems to provide them. For example, Web services and legacy system, such like database system, are good sources of services for an MAS.

- ASDL/ACDL: ASDL (Agent Service Description Language) or ACDL (Agent Capability Description Language) is used to provide a specification to describe capabilities of agents or other types of services and to facilitate service publishing and requesting. Almost all of MASs are their own SDL (service description language) to expose their own features based on its agent modeling, service types, matchmaking mechanism, or query processing style.

- Mediation Infrastructure: Besides agents that provides services for other ones, there should be an infrastructure to realize the mediation protocol used in the MAS. Some kinds of special agents, like middle agents, or other foundation constructs are playing the role to mediate among agents.

- Matching Feature: There could be various of matching mechanisms that exhibit different matching behaviors to find appropriate services against the requests. Mechanisms frequently used are exactly matching, structural/syntactic matching, and similarity/partitial matching.

- Matching Algorithm: Different technologies can be leveraged to realize the matching mechanisms just mentioned. Each technology has it own features to fit in with the special needs of each MAS.

- Heterogeneous Interoperability: In order to integrate with different environments beyond the targeted MASs, some MASs also tackle with the interoperability issue in the development of MASs. Interoperators, wrappers, or translators are such components that can be used to realize the interoperability between heterogenous environment.

3. SAM: A Service-Oriented Architecture for Multi-agents

As an attempt towards the construction of an MAS infrastructure to better integrate agents, web applications and information sources, we begin with the devise of a service matchmaker based on the formation of possibilistic reasoning [18], and an extension of our previous work on a PPN-based agent service description language (PPN-ASDL) [19] together with a set of ontologies for semantics matching, then gradually proliferate to the integration of web infrastructure with the SAM-ASDL translator that utilizes our newly-defined tModels in UDDI registry, and provide an easy access via a tool bar in the client browser, called SmartAgent bar, to the multi-agent system for the query processing, and finally conceive the notion of composite service agents capable of the composition of agents, web applications and information resources. Refer to figure 1 for an overview of SAM.

On the client site, a browser with SmartAgent bar is established to facilitate the users in issuing their queries and in utilizing services provided in SAM. The adoption of Internet browsers makes easy the design of SAM in a more flexible way to immerse within the existing Internet-based environments. Users can thus obtain their services through a browser as usual.

The web service infrastructure is an environment built upon industrial web standards including SOAP, WSDL and UDDI. SOAP services can be deployed in SOAP servers, described by WSDL documents, and then published in UDDI registry. We have developed a set of SOAP services in the ATIS application (see section 5), for example, E-Map, Shortest Path, Parking Lot, and etc.

Inside the multi-agent system, there are three types of agents: middle agents, request agents and service agents. Service agents provide services. Request agents work together with service agents to obtain the services requested. Agents that help find the relevant service
agents for the request agents in an open environment are considered as middle agents. Service matchmaking is the process of locating an appropriate service for a request agent through the matchmaking mechanism resided in a middle agent. A traditional matchmaking process for MAS proposed by Katia Sycara et al. [9, 30] usually includes the following steps: (1) service agents publish their advertisements to the middle agent, (2) the middle agent stores these advertisements, (3) request agents ask the middle agent to locate and connect to service agents with desired services, (4) the middle agent matches the request against the stored advertisements and returns the results, and (5) the request agent communicates with the corresponding service agents and decides whether to subscribe to their services or not.

In order to integrate the agent technology with web services environment in a seamless fashion in SAM, we have extended this framework along four dimensions:

- To integrate web services into the service matchmaking, the middle agent utilizes an ASDL translator to gather web services from the UDDI registry. This UDDI registry applies the newly-defined tModels to annotate the web services with data about the features designed in SAM. Through this mechanism, the middle agent is capable of matchmaking web services for agents.

- To increase the flexibility of SAM, the request agent itself not only can request services from other service agents but also from the web services published in UDDI registry. In the design of request agent, a special document describing the process model of operations is taken into account in helping the request agent interact with a web service automatically.

- To provide a uniform service request and delivery mechanism, a web server bridges the users and the multi agents system in SAM. Queries issued by users are converted into a service request and sent to the request agent. Service results returned from service agents or web services are then sent back by the request agent and delivered to the users by the web server.

- To improve the interoperability of services involved in SAM, service agents may be considered as wrappers to interact with the back-end web services via SOAP messages or other legacy systems. Moreover, the composite service agents can cooperate with multiple service agents and web services according to the domain knowledge to provide better service usages for the user.

In the following sections, we will elaborate in-depth the design and implementation of middle agents. Besides, the interoperability with web service standards and scenarios of demonstration are also given.

### 4. Middle Agents: PPN-Based Service Matchmakers

In an MAS, the agents called middle agents are capable of registering service profiles published by service agents and spotting appropriate service providers for the request agents to answer to a request issued by the user. After performing service matchmaking, request agents process the matched service advertisements returned by middle agent to contact with the desired services. The tenet of the proposed MAS, namely SAM, is an extended service matchmaking mechanism resided in the middle agent with the following main features (see figure 2).

![Figure 2. Overall Architecture of Middle Agent](image)

- A matchmaking engine based on Possibilistic Petri-Nets (PPN, see section 4.1): The PPN Matching Engine matches the services, published by agents or web service providers, for request agents by using the SAM-ASDL (see section 4.2). The matchmaking engine extends and implements the PPN-ASDL matching algorithm [19]. The DAML+OIL [15] related ontologies (see section 4.3) are stored inside the middle agent. An inference engine utilizes these DAML+OIL ontologies to help the matchmaking engine perform semantic matchmaking.

- SAM-ASDL service profile registries: The SAM-ASDL profile registries are made for storing the SAM-ASDL profiles. In order to distinguish the service types, service profiles advertised by service agents are stored in Agent Service Profile Registry, and service profiles found from the UDDI registry through the SAM-ASDL translator are stored in Web Service Profile Registry. When performing service matchmaking, the middle agent will search profiles registered in the SAM-ASDL service
profile registries first. If there is no service matched, the service query can further be processed through directly searching public UDDI registries and register the newly found web services back to its registries.

- **SAM-ASDL Translator** (see section 4.4) and UDDI Browser (see section 4.5): The SAM-ASDL Translator is developed to bridge the linkage between UDDI registries and the middle agent. In order to realize this linkage, a set of newly-defined tModels and APIs are elaborated. The provided APIs can help for converting service profiles between SAM-ASDL profile format and UDDI data structures. Besides, we can manipulate or search UDDI registries in WSDL or SAM-ASDL driven fashions through utilizing these APIs. By cooperating with the SAM-ASDL translator, the middle agent can advertise SAM-ASDL based service profiles to public UDDI registries and directly query web services published from them also. The UDDI Browser is an user interface to help users in browsing UDDI registries with previous fashions. Therefore, various MASs and web service infrastructure can cooperate with each other through the public UDDI registries.

### 4.1 Possibilistic Petri-Nets based Service Matchmaking

Traditional service matchmaking is based on the key words search mainly. However, the services requested are always not the same as the services advertised. Using this approach will limit the quality of matched services and doesn’t utilize the benefit of semantics. For coping with some degrees of partial and semantic matchmaking, we use Possibilistic Petri nets (PPN) [18] and PPN-ASDL matchmaking algorithm [19] as a mechanism to improve the quality of service matchmaking.

A typical interpretation of Petri nets (see figure 3) is to view a place iP(r) or OP(r) as a condition, a transition t as the causal connectivity of conditions, and a token in a place as a fact used to claim the truth of the condition associated with the place. For example, suppose there is a hotel service agent assisting users in reserving a hotel on line. The location of the hotel is in Taipei City and the room type it provides is a twin. Thus the hotel service agent and a request agent are shown in the figures 3(a) and 3(b), respectively. Two input places describe the preconditions of this service, and the output place denotes the situation after performing this service. We can put a token in each input place to signify that the preconditions are truth, and then fire the transition to perform this service. The tokens in input places are taken away and a token is put in the output places to show the results after performing this service. Note that the confidence about the connectivity of conditions and the facts can be uncertain.

![Figure 3. (a) A service agent, (b) A request agent](image)

Possibilistic Petri-nets (PPN) [18] is an extension of Petri-nets with the capability to add possibility Π and necessity N measures on the possibilistic tokens and possibilistic transitions. It uses possibility distributions over all places and tokens to display the uncertainty about possible locations of a token before receiving certain information. The calculation of (N, Π) of corresponding possibilistic tokens and possibilistic transitions is based on the possibilistic reasoning. Figure 4 is a representation of matching request with service in PPN-ASDL. The general description of the matchmaking algorithm is: (1) construct the PPN for the request and the service, (2) create new possibilistic transitions between corresponding places in these two PPNs, (3) put a possibilistic token in each input places of request, (4) fill in the initial (N, Π) of each possibilistic tokens and possibilistic transitions, and (5) fire the transitions and calculate the (N, Π) of the remaining possibilistic tokens and possibilistic transitions according to the possibilistic reasoning rule. The (N, Π) of the possibilistic token in the output place of the request will be the confidence degree that the service can satisfy the request. Refer to [19] for more detail about this algorithm.

![Figure 4. Matching request with service in PPN-ASDL](image)
4.2 SAM-ASDL: Agent Service Description Language in SAM

Agent service description language (ASDL) plays an important role in facilitating the middle agent to capture the content in a message and to match appropriate services for the request agents. Therefore, the agreement in the use of a common ASDL, which defines a syntactical specification of publishing and requesting services, is the key to understanding and automatic processing requests and services of agents. A number of recent developments of service description languages have been proposed, for example:

- A service description language in InfoSleuth [7] consists of four fragments: conversation ontology, language ontology, service ontology and domain ontology.
- LARKS [27] is a frame-like specification with the following slots: context keywords, variable types, input and output variables, logical constraints on input and output variables, and ontological descriptions of keywords.
- A service description language in IMPACT [3] comprises three parts: a service name, typed input and output variables, and attributes of services.
- The specification of CDL [29] is to some extent similar to LARKS with the following components: service name, inheritance relation, input and output variables, logical constraints on input and output variables, and a special constraint across input and output variables.
- DAML-S [1] is a DAML+OIL based web service ontology that has three main parts: the service profile, the process model, and the grounding.

To fulfill the requirements of a service description language setup for SAM, we have extended our previous work on PPN-ASDL by means of DAML-based services description ontologies. The foundation of the ASDL developed in SAM, called SAM-ASDL, is made up of two main ontologies: the generic service profile ontology and the service grounding ontology. The generic service profile ontology contains basic concepts needed to be described in a service. A specific service profile ontology specializes these concepts for a specific service domain. These specific concepts are used by service agents to advertise services and by request agents to describe the services needed. In general, the generic service profile ontology (see figure 5) defines concepts and property for describing the following data:

- Service related information: It describes the general attributes about a service, included its name, description and service category.
- Provider related information: It is used to annotate the provider of a service. The data described are its name, description, contact information, and web site URLs.
- Variables: There are three types of variables: input variable, output variable and condition variable. During the invocation of services, the service agents or web services need the input variables to perform the services and produce the output variables for the request agents. Condition variables are used as decorators to describe additional attributes of a service. A condition variable can be either an input variable or an output variable. In order to describe the services more precisely, we group the specialized variables into domains for better usage.
- Conditions: Agents or web services use the conditions to describe the services they provide or need. A condition is made up of an instance of a specialized condition variable and a value of condition type. The middle agent processes these conditions to perform PPN-based service matchmaking.

Figure 5. Generic Service Profile Ontology

- Matchmaking related information: We use a set of properties, included the message type of this document, the type of a service, searching criterion, and etc., to provide additional information used by middle agent in processing related tasks.

The service grounding ontology provides information for service invocation. After obtaining the matched services from the middle agent, request agents can utilize the information to further process requested services. The main portions of the service grounding ontology (see figure 6) are depicted below:

- Agent grounding: It is used to indicate that the service is provided by a service agent, which contains several attributes about the location of a agent, its name, the tasks it performs, and the token it needs. Tokens refer to the input or output variables used in the specific service profile
ontologies. The service agents perform the tasks through these tokens to achieve the services published.

- **WSDL grounding:** In distinguishing from the agent grounding, this is used to indicate that the service is provided by a web service. This grounding contains the following attributes: WebURL, WsdlURL and InstructionURL. WsdlURL keeps the references to the WSDL documents of the web service. WebURL is related to the front-end web site of the web service. Through the web site, user can use the backend services directly. InstructionURL stores the references of the instruction description documents of web services that prohibit a web site from being accessed directly. The instruction description is a DAML-S process model like language in helping describe how an agent can access it automatically. If the matched services are web services, the request agents can get the WSDL and the instruction description documents, parse them to find related information, and access the web services via SOAP messages.

Refer to appendix A and B for more detail about the specifications of the generic service profile ontology and the grounding ontology. Figure 7 is a part of an SAM-ASDL document that describes a hotel reservation service agent. In figure 7(a), there are two conditions. The first one is the room type provided by the hotel is Single or Twin, and the second one is the location of the hotel is in Chungli. The middle agent then can use these conditions to perform service matchmaking. On the other hand, the service grounding information is described in figure 7(b). It depicts the URL of the service agent and its name in that agent community. The task name, which is RequestForService, and related data, which are Name, CheckInDate and CheckOutDate, for requesting the task are also provided.

![Service Grounding Ontology](image)

**4.3 SAM-ASDL Related Ontologies**

The semantics used in SAM-ASDL makes easy the matchmaking process. We have leveraged on the well addressed researches on semantics web [14, 15], and developed a set of DAML+OIL based ontologies to lay the ground work for SAM-ASDL (see figure 8).

- **Category domain ontology:** This ontology defines a categorization scheme used by the middle agent to filter unrelated services of different service domains. Each service domain refers to a specific service profile ontology. Users can search the service domain they are interested in first and use the specific service profile ontology of this service domain to fill in a request form in ASDL.

- **Specific service profile ontologies:** In order to precisely describe a service, a corresponding specific service profile ontology is created for each service domain. In this ontology, basic concepts in generic service profile ontology are specialized with more specific concepts related to the service domain. These concepts may be defined in a domain specific ontology or from types defined in an XML schema. These specific concepts can then be used to describe the services requested or published in the same service.
domain.

- Domain-specific ontologies: All domain specific concepts are defined in each domain-specific ontology. This ontology contains other semantic relations [6, 11], for examples part/whole or relevancy, used by the DAML+OIL inference engine to calculate the similarity among these concepts for PPN matchmaking engine.

An HTML page can be generated dynamically by performing an ontological reasoning over the concepts defined in a specific service profile ontology to guide the user to fill in an SAM-ASDL document for requesting or publishing a service. The DAML+OIL inference engine implemented by DAMLJessKB [17] uses the domain specific ontologies to derive the similarity between concepts. Relations between concepts provide the pre-defined relevancy values. The semantic relation path is a directed path composed of the same type of relations from one concept to the other one. Calculating the similarity between two concepts is performed by multiplying the relevancy values of the relations which constitute the semantics relation path. The maximum similarity is chosen in the case that there exist multiple semantic relation paths between two concepts. Consider the hotel reservation example, a portion of domain specific ontology that describes the relations is shown in Figure 9, and the similarity between Taipei and Taichung is 0.76, which is the result of multiplying 0.95 and 0.8.

4.4 SAM-ASDL Translator

The SAM-ASDL translator is designed to bridge the linkage between the public UDDI registries and middle agents. There are three issues involved in developing this translator: (1) how to fill in the extended semantic information captured in SAM-ASDL to UDDI data specification, (2) how to utilize the filled in semantics information to facilitate the locating and converting of SAM-ASDL based services profiles registered in UDDI registries, and (3) how to represent and reason the process model of the operations resided in a web services. We have discussed the third issue in the design of service grounding ontology of SAM-ASDL in section 4.2. In this section, the translation mechanism between the middle agents and UDDI registries is fully discussed.

![Figure 9](image.png)

**Figure 9.** An example of (a) Room Type Choice, and (b) Location Choice concepts hierarchy and relevancy defined in a domain specific ontology

The data model defined in the UDDI specification is insufficient to carry semantics information for the services published [10, 24]. Fortunately, the tModel data model designed in the specification of UDDI is useful in specifying additional attributes of the entities in the UDDI registry. To publish SAM-ASDL profiles in UDDI registry, we have defined a number of tModels, see figure 10 for this mapping. These tModels help keep SAM-ASDL specific information in the UDDI registry, which are defined as follows:

- serviceCategory_tModel: specify the service category of the web service.
- InputVariable_tModel: specify the input variables of the web service.
- OutputVariable_tModel: specify the output variables of the web service.
- ConditionVariable_tModel: specify the condition variables of the web service.
- hasValue_tModel: specify the restricted type and value of the variables.
- conditionType_tModel: specify the condition type of the condition.
- WebURL_tModel: specify the url refers to the web site as the front-end user interface of the web service.
- InstructionURL_tModel: specify the url refers to the instruction document that describes the process model of operations resided in the web service.
- SAM-ASDL_tModel: information published by the service is compliant with the data model of
In figure 11, we illustrate a condition of the hotel reservation service agent with the newly-defined tModels. In this example, three tModels, included ConditionVariable_tModel, hasValue_tModel, and conditionType_tModel, are referred. The semantics are inserted as the values of the KeyValue property of the KeyedReference instances that depict a condition restricting the room type provided to be a single. We use the XML namespaces to introduce for convenient. In practice, these namespaces should be replaced by the URIs that the concepts belonged to. The SAM-ASDL translator then utilizes these tModels created, the data model structure described in UDDI specification, and WSDL documents referred to translate a UDDI service entry into an SAM-ASDL document stored in the middle agent. This mechanism helps the middle agent to handle the semantics well when performing service matchmaking based on these tModels defined.

After publishing a service by embedding SAM-ASDL related information in UDDI registry, we describe the detail process handled by the ASDL translator. First, we create the Profile portion of a SAM-ASDL document for a service. An instance of Provider concept is constructed by mapping from the BusinessEntity. The name of the service and its general descriptions are then created by gathering the data of the BusinessService. Input and output variables can be derived from the KeyedReference instances of InputVariable_tModel and OutputVariable_tModel, or from the message data elements defined in the WSDL documents referred. Conditions are built upon the KeyedReference instances of related tModels, included ConditionVariable_tModel, hasValue_tModel, and conditionType_tModel, representing the various conditions. The other information in the Profile is given the default value as defined in SAM-ASDL specification by the ASDL translator. Finally, the making of the Grounding portion of a SAM-ASDL is simplified to create an instance of the WSDLGrounding concept. The values of the WsdIURL property are the URLs of WSDL documents referred by the BusinessService entry. Other related sub-parts, like InstructionURL and WebURL, are described in KeyedReference instances of InstructionURL_tModel and WebURL_tModel, respectively.

4.5 UDDI Browser

So far, the core technologies and components that are employed to build a service matchmaking for the proposed MAS called SAM are introduced. This service matchmaking tackles the semantic and uncertain issues for matching services well. However, it will also need to develop a convenient user interface for users to searching services beyond the extent of MASs. The newly elaborated component, called UDDI Browser, is an UDDI registry independent searching tool that extends the basic query scenarios described in the UDDI specification by incorporating with the translator mechanisms introduced in section 4.4.

4.5 UDDI Browser

So far, the core technologies and components that are employed to build a service matchmaking for the proposed MAS called SAM are introduced. This service matchmaking tackles the semantic and uncertain issues for matching services well. However, it will also need to develop a convenient user interface for users to searching services beyond the extent of MASs. The newly elaborated component, called UDDI Browser, is an UDDI registry independent searching tool that extends the basic query scenarios described in the UDDI specification by incorporating with the translator mechanisms introduced in section 4.4.

4.5 UDDI Browser

So far, the core technologies and components that are employed to build a service matchmaking for the proposed MAS called SAM are introduced. This service matchmaking tackles the semantic and uncertain issues for matching services well. However, it will also need to develop a convenient user interface for users to searching services beyond the extent of MASs. The newly elaborated component, called UDDI Browser, is an UDDI registry independent searching tool that extends the basic query scenarios described in the UDDI specification by incorporating with the translator mechanisms introduced in section 4.4.

Figure 12 is the layered architecture of UDDI Browser. There are four parts made up the UDDI Browser mainly. The part at the bottom of the layered architecture is the Standard UDDI Application Interface. It provides all functionalities need to manipulate a targeted standard UDDI registry which is publicly accessible via Internet. These functionalities comply with the interfaces, which are RPC-based methods for communicating through SOAP [5] over HTTP, specified in the UDDI Speciation [4]. WSDL2UDDI Translator lies on the Standard UDDI Application Interface and implements the best practice [23] proposed by OASIS organization, which is the maintainer of UDDI Specification, to provide a recommended approach to mapping WSDL [8]
descriptions to the UDDI data structures. Any targeted
UDDI registry conforms this best practice should
register all tModels newly identified in [23] as a basis to
perform WSDL driven service searching. Though,
WSDL2UDDI Translator can be implemented by
parsing all referred WSDL documents by a registered
business service entity, this kind of searching
infrastructure can significantly decrease the load of the
service finder. SAM-ASDL Translator, on the other hand,
provides the same functionalities like WSDL2UDDI
Translator except for SAM-ASDL documents. The
mapping of SAM-ASDL to UDDI data structures and
additional identified tModels for this mapping are
specified in the section 4.4. Finally, User Interface is the
front-end interaction interface for users to manipulate the
settings of targeted UDDI registries to be searched, to
fill in the fields for different searching styles, and to
browse the searching results by cooperating with the
other underlying parts just introduced.

Although UDDI Browser can perform almost tasks relate
to service searching, it can't replace the service
matchmaker entirely. Issues involved in service
matchmaking, including uncertainty, semantic reasoning,
quality of service and et al., are much suitable and
tackled more effectively in an open and distributed
environment than a standalone tool. However, service
searching should not be the limitation of the
development of UDDI Browser. Service providers also
need a convenient way to publish services in a UDDI
registry which will be the next direction of development
of the UDDI Browser in the future.

5. An ATIS Example: Hotel Reservation Service
and Traveling Service

In this section, we give a scenario of advance
traveling information system (ATIS): to query for the
hotel information, reserve a hotel, and travel to the hotel,
as an illustrative example to demonstrate how to publish,
request, compose, match, and deliver services in SAM.

There are two service domains involved in this
scenario. Service gents in the hotel reservation domain
provide services for serving hotel by wrapping the web
site of each hotel. The composite service agent in the
traveling domain composes several simple services,
including E-Map, Shortest Path, Route Guidance and
Parking Lot information.

Let's first consider the scenario of hotel reservation
service (see figure 13). To start with, the user inputs a
query "hotel reservation" in the SmartAgent bar to query
for the service domain. After choosing the specific
service domain, the web server generates a request form
for the hotel reservation domain by reasoning its related
ontologies in this domain. In the request form, the user is
guided to fill in requested information, including
personal data, check in date, check out date and etc. The
personal data may be retrieved through the smart card
attached with SAM. He/she can also check the search
criteria for the hotel services he/she is interested in, for
example, the location of the hotel and the room type.
He/she can then send this request to the web server
which will convert it into an SAM-ASDL document, and
in the meanwhile as a trigger to instantiate a request
agent to take in the request form in the SAM-ASDL
format. The instantiated request agent brings this
SAM-ASDL document to the middle agent for
matchmaking.

If the user doesn't allow the request agent to act on
his/her behalf, the request agent will send the matched
results with their matching degrees to the message pool
of the web server and wait for the user to pick up a
service. After a desired service is clicked, the request
agent brings related information to the hotel service
agent and makes the reservation. The hotel service agent
then accesses the web site of the hotel and preprocesses
all the tasks for the user. Finally, a hotel reservation
confirmation form is returned to the web server. The
user can peruse the incoming message and decide
whether to reserve the hotel or not. If a confirmation is
made, this information will be sent to the hotel's web site
directly. An on-line hotel reservation is thus completed.

After reserving a hotel, we further describe the second
scenario of requesting for traveling information about
the hotel reserved (see figure 14). Again the user queries
for the service domain by typing "traveling" and locates
the traveling domain. Once again, the web server
dynamically generates the request form of the service
domain. In this request form, the user fills in the
destination by means of voice input, and obtains the
current position through the GPS receiver. He/she then
picks the information to make shape the query. Again
the web server converts the request form into an SAM-ASDL document and sends it to the request agent.

The request agent fetches the request to the middle agent for service matchmaking. This time, it is a composite service agent with four simple services that matches the request. The composite service agent accesses these services and composes the results according to its domain knowledge. Finally, this service result is sent to the web server, and the user gets the message to utilize the services in a Map Viewer tool which is embedded in the web page for rendering map information. The user then follows the shortest path and voice guidance to the hotel. The parking lot information is delivered automatically half mile before arriving the hotel.

6. Conclusion

In this paper, we devise a PPN-based matchmaker for the proposed multi-agent system (MAS) called SAM, Service-oriented Architecture for Multi-agents, as an attempt towards the construction of an MAS structure that can better integrate agents, web applications, and information sources.

In the making of the matchmaker resides in the middle agents for SAM, we build this construction based on the following works. First, the formation of possibilistic reasoning and the model of possibilistic petri-net (PPN) serve as the basis for representing and matching services. Second, an extension of our previous work on a PPN-based agent service description language (PPN-ASDL) called SAM-ASDL, ASDL for SAM, is developed to capture characteristics of a service involved in SAM. Third, a set of ontologies related to SAM-ASDL are designed and used for semantics matching. Fourth, an SAM-ASDL Translator that accompanies with the middle agent utilizes our newly-defined tModels for the UDDI registry in helping the integration of web service infrastructure with SAM. Finally, a standalone service searching tool named UDDI Browser leverages some works developed previously to provide a basic and convenient user interface to search services in any standard UDDI registries.

In the future, we will focus more on how to combine the quality of web services with the PPN matchmaker to increase the confidence of the matching results.

Acknowledgment

This research was sponsored by Ministry of Education in Taiwan under the grant EX-92-E-FA06-4-4.

References

Improving the functionality of UDDI registries through web services semantics. Springer-Verlag Berlin Heidelberg, 2002.


Appendix A Specification of Generic Service Profile Ontology in SAM-ASDL

The detail of each concept and property defined in the generic service profile ontology of the SAM-ASDL are as follows:

- **Profile**: The class Profile serves as a superclass of high-level description of the service published or requested. It contains a set of relations that describe different attributes of a service. These relations are expressed by the properties serviceName, serviceDescription, serviceCategory, hasProvider, hasVariable, hasCondition, messageType, source, searchBy, and requestType. When issuing a request, the properties including serviceName, serviceDescription, serviceCategory, and source can be ignored in an ASDL document for requesting. And in an ASDL document for publishing, the property searchBy is ignored.

- **serviceName** refers to the name of a service published. It is used to distinguish different attributes of a service. These relations are expressed by the properties serviceCategory, hasProvider, hasVariable, hasCondition, messageType, source, searchBy, and requestType. When issuing a request, the properties including serviceName, serviceDescription, serviceCategory, and source can be ignored in an ASDL document for requesting. And in an ASDL document for publishing, the property searchBy is ignored.

- **serviceDescription** provides a brief description of a service. Any information for human reading can be described in it.

- **serviceCategory** refers to the service category that the service belongs to. In SAM, a categorization scheme is used for maintaining service registry and performing service matchmaking easily.
- **hasProvider** collects a list of instances of Provider class.
- **hasVariable** collects a list of instances of Variable class.
- **hasProvider** collects a list of instances of Condition class.
- **messageType** indicates the purpose of an ASDL document sending to the middle agent. The possible value of this property may be RARequest, SARequest, Advertise, and Delete. RARequest and SARequest are used to handle different request messages sending by request agents and service agents, respectively. Advertise and Delete are used to publish or un-publish a service by service agents.
- **source** is used to indicate the service provided is a web service or an agent service. Two possible values are used, namely WebService and AgentService.
- **searchBy** is used by the middle agent to perform different filtering strategies in service matchmaking. The middle agent reduces the search space down by filtering out irrelevant services. The values of the property are ServiceName, ServiceCategory, and ServiceProvider. In SAM, the ServiceCategory is the default filtering strategies to find services in relevant service category that a request agent is searching.
- **requestType** is primarily used by a request agent. Two possible values can be filled, namely Computer and User. A user can assign the Computer value to grant the request agent with the authority to access the matched services automatically based on his/her preferences. Otherwise, the default value User is used to give the decision back to the user.

### Provider:
The Provider class provides information of the service provider for human reading mainly. It is composed of the properties including providerName, providerDescription, providerContacts, and providerWebURL.
- **providerName** refers to the name of the service provider.
- **providerDescription** is a brief description of the service provider.
- **providerContacts** contains a list of contact information.
- **providerWebURL** is the web site URL of the service provider.

### Variable:
The class Variable is used as a superclass of other variables. In SAM, there are three types of variables: **Input_Variable**, **Output_Variable**, and **ConditionVariable**.
- **hasValue** refers to the instances of the restricted classes defining in a subclass of Variable. The classes may be predefined elements in XML schema specification or be defined in a DAML+OIL ontology. When specifying a subclass of Variable in a specific service profile ontology, we also restrict the data types or potential values of its instances. This design will facilitate to generate a user interface of an ASDL document for publishing or requesting services.
- **conditionValue** refers to an instance of **Condition_Variable**. It is the main part of a condition.
- **conditionType** is used to provide a more powerful logical description of a condition. The possible value for this property is AND or OR. We can use this property in the case that there are multiple values of conditionValue property specified in a condition instance.

### Appendix B Specification of Grounding Ontology in SAM-ASDL

The detail of each concept and property defined in the grounding ontology of the SAM-ASDL are as follows:

- **AgentGrounding**: The class AgentGrounding is used to indicate that the service is provided by a service agent, which contains several attributes such as agentURL, agentName, and hasTasks.
- **agentURL** keeps the URL information of the service agent.
location that a service agent resides.

- **agentName** refers the name of the service agent in the agent community.
- **hasTasks** stores a list of instances of Task.

**Task**: The class Task is used to present a task performed by a service agent. When carrying out a service, the service agent may perform many tasks by communicating with the request agents. It has a property called tokens only.

- **taskName** describes the name of a task.
- **tokens** keeps a list of instances of the class Token.

**Token**: The class Token is a subclass of Input_Variable or Output_Variable. The instances of Token are related to the input or output variables used in the specific service profile ontology. The service agents perform the tasks through these tokens to achieve the services published.

**WSDLGrounding**: To distinguish from the agent grounding, the class WSDLGrounding is to indicate that the service is provided by a web service. This grounding contains the following attributes: WebURL, WsdlURL and InstructionURL.

- **WsdlURL** keeps the references to the WSDL documents of the web service.
- **WebURL** is related to the front-end web site of the web service. Through the web site, user can use the backend services directly.
- **InstructionURL** stores the references of the instruction description documents of web services that prohibit a web site from being accessed directly. The instruction description is a DAML-S process model like language in helping describe how an agent can access it automatically. If the matched services are web services, the request agents can get the WSDL and the instruction description documents, parse them to find related information, and access the web services via SOAP messages.

---

**Jonathan Lee** is a professor in the Computer Science and Information Engineering at National Central University (NCU) in Taiwan, and was the department chairman from 1999 to 2002. He is currently the director of Software Research Center at NCU. His research interests include agent-based software engineering, service-oriented computing, and software engineering with computational intelligence. He received his Ph.D in computer science from Texas A&M University in 1993. He is the president of Taiwan Software Engineering Association, a senior member of the IEEE Computer Society and a member of the ACM.

---

**Yao-Chiang Wang** is a PhD student in the Computer Science and Information Engineering at National Central University (NCU) in Taiwan. His research interests are in service-oriented computing and software engineering, including service matchmaking, quality of service, service monitoring, and agent technology. He received his B.S. degree in Computer Science and Information Engineering from National Central University, Jhongli City, Taiwan. Contact him at solo@selab.csie.ncu.edu.tw.

**Chia-Ling Wu** received his B.S. degree in Computer Science and Information Engineering from National Chiao Tung University (NCTU) in Taiwan, in 1999. He is currently a Ph.D. student in the Department of Computer Science and Information Engineering of National Central University. His research interests include agent-based software engineering, service-oriented computing, requirements engineering and development process modeling.

**Shin-Jie Lee** received his B.S. degree in Mathematics from National Changhua University of Education, Taiwan, in 2000. He is currently a Ph.D. student in the Department of Computer Science and Information Engineering of National Central University. His current research interests include agent-based software engineering, service-oriented computing, and software engineering.

**Shang-Pin Ma** is a PhD student in the Computer Science and Information Engineering at National Central University (NCU) in Taiwan. His research interests are in service-oriented computing and software engineering, focusing on software composition. He was project coordinator of SIM project (Service-oriented Information Marketplace), that is sponsored by Ministry of Economic Affairs in Taiwan, from 2003 to 2004. He received a BS in Computer Science and Information Engineering from National Central University, Jhongli City, Taiwan.

**Whan-Yo Deng** is a PhD Student in the Computer Science and Information Engineering at National Central University (NCU) in Taiwan. His research interests include service-oriented computing and software engineering with computational intelligence. He is the member of Taiwan Software Engineering Association.