Possibilistic Petri nets as a basis for agent service description language

Jonathan Lee\textsuperscript{a,∗}, K.F.R. Liu\textsuperscript{b}, Y.-C. Wang\textsuperscript{a}, W. Chiang\textsuperscript{c}

\textsuperscript{a}Department of Computer Science and Information Engineering, National Central University, Chungli 320, Taiwan
\textsuperscript{b}Department of Environmental Engineering, Da-Yeh University, Changhua 51542, Taiwan
\textsuperscript{c}Department of Civil Engineering, National Central University, Chungli 32054, Taiwan

Abstract

In the making of a service-oriented multiagents framework, two pivotal issues need to be addressed: agent service description language (ASDL) and its service matchmaking mechanism. ASDL provides a specification of publishing and requesting services for agents, and matchmaking is the process of finding an appropriate service for a request through a medium. In this paper, a possibilistic Petri net-based agent service description language (PPN-ASDL) is proposed as an advanced ASDL with four key features: possibilistic transitions to represent a service or a request; input places to denote preconditions expected to hold before performing the services, output places to denote postconditions expected to hold after performing the services; and possibility and necessity measures to quantify the confidence levels that an agent can provide the relevant service for a request. A matchmaking mechanism, permitting a relaxed match for close semantics, is also developed to search for the possible services among agents for a request.

© 2003 Elsevier B.V. All rights reserved.

Keywords: Possibilistic Petri nets; Agent service description language (ASDL); Ontology

1. Introduction

In the making of a service-oriented multiagents framework, two pivotal issues need to be addressed: agent service description language (ASDL) and its service matchmaking mechanism. ASDL is a basic need for agents to describe themselves to other agents. It plays a crucial role in service matchmaking among agents in an open information environment because it helps agents to locate
other agents that can provide them with services to accomplish specific requests at a given time. The increasing variety of services provided in the cyberspace makes the issues of ASDL and service matchmaking significant. An expressive and flexible ASDL is in need, and an effective matchmaking mechanism is essential for a service-oriented framework meant for multiagent systems. As a result, service description languages and matchmaking mechanisms have been discussed and worked on within the context of several agent systems, for example, InfoSleuth [5], IMPACT [1], LARKS [29], CDL [32], and XML-based service description framework such as RDF [16], and DAML+OIL [12].

Three types of agents can be identified in such a framework: service-agents, request-agents, and middle-agents. Service-agents provide services such as finding information, or performing a particular domain-specific problem solving. 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 finding an appropriate service-agent for a request-agent through the matchmaking mechanism embedded in a middle-agent [29]. There are several important issues involved in this matchmaking process: an agent service description language that provides a specification of publishing and requesting services [32], an ontology to interpret the semantics of terms used in the descriptions of agent services [13], a matchmaking mechanism to search for the relevant services for request based on their semantics, and a service composition approach to satisfy a request by integrating several different services [25].

In this paper, we propose:

- the use of possibilistic Petri nets as a basis for ASDL: possibilistic transitions to represent a service or a request; input places to denote preconditions expected to hold before performing the services, output places to denote postconditions expected to hold after performing the services; and possibility and necessity measures to quantify the confidence levels that an agent can provide the relevant service for a request.
- the adoption of DAML+OIL ontology markup language for sharing domain knowledge: to enrich the semantics of agent services and requests, we choose one of the ontology markup languages—DAML+OIL [10] to annotate the terms used in the services and requests. DAML+OIL provide a specification consisting of definition of classes, properties, constraints and their relations. The meanings of terms are annotated by DAML+OIL and the relations are represented by hierarchies of concepts (classes) in a domain. The locations of services and requests in the ontological hierarchies of classes are annotated by DAML+OIL. The possibility and necessity measures between services and requests in the hierarchies are then computed in a manner of similarity.
- a service matchmaking mechanism that is capable of dealing with uncertainty: a matchmaking mechanism is also proposed in this paper to calculate the overall confidence level about a service-agent satisfying a request.

The organization of this paper is as follows. The service-oriented framework for multiagent systems is introduced in the next section. In Section 3, the background of possibilistic Petri nets is discussed. In Section 4, possibilistic Petri net-based agent service language accompanied with DAML+OIL ontology markup language is proposed. A matchmaking mechanism is also developed in this section. Finally, a summary of our approach and its potential benefits are given in Section 5.
2. A service-oriented framework for multiagent systems

In a service-oriented framework for multiagent system, agents are distinguished into three categories: service-agents that provide services such as resources, information, products and problem-solving capabilities; request-agents that request some specific services; and middle-agents that provide means of meaningful coordination activities to facilitate searching for relevant service-agents for request-agents. There are several important features involved in this framework (see Fig. 1):

- service-agents publish their services to middle-agents,
- middle-agents store these services information,
- request-agents request middle-agents to locate and connect to service-agents with desired services,
- the corresponding middle-agent processes this request through a matchmaking mechanism and returns the results, and
- the request-agent communicates with the corresponding service-agents and decides whether to subscribe to their services or not.

The focus of this framework is on how to perform the service matchmaking mechanism embedded in the middle-agents, that is, how to search for the relevant service-agents. To implement the framework, we will need:

- an ASDL to provide a specification of publishing and requesting services [32],
- an ontology language to interpret the semantics of terms used in the descriptions of agent services [13],
- a matchmaking mechanism based on the agent service description language and on the ontology language [31], and
- a service composition approach to satisfy a request by integrating several separated services [25].

2.1. Agent service description language

The key to understand and automatically process requests and services of agents is the agreement in the use of a common agent service description language, which defines a syntactical specification of publishing and requesting services. We sum up a set of requirements for establishing an ASDL
to cope with an open information environment that is usually imperfect in nature:

- Equipped with expressive power: it is desirable for an ASDL to be able to describe the preconditions that are expected to hold in the situation before services are performed, and the post-condition that are expected to hold in the situation after services are performed. Moreover, it will be easier to understand the contents of services and requests if an ASDL can be displayed graphically.
- Capable of handling uncertainty: an ASDL with the capability of quantifying the possibility of providing services by service-agents is essential for tackling uncertainty in an open environment. The possibility arises from the uncertainty about fulfilling preconditions of providing services.
- Easy to integrate: A task which is demanded by a request-agent sometimes needs several service-agents to accomplish coordinately. Therefore, it is crucial for an ASDL to provide a basis for service composition.
- Powered with semantics: ASDL itself is a syntactical specification for representing services and requests, therefore, in order to offer the power of semantics, a markup language to serve the purpose of annotation is vital for the interpretation of terms used in describing services.

Recently, progresses have been reported towards the development of ASDLs, for example, InfoSleuth [5], IMPACT [1], LARKS [29], CDL [32], RDF [16], OIL [8,9] and DAML+OIL [12]. An agent service description in InfoSleuth consists of four fragments: conversation ontology, language ontology, service ontology and domain ontology. The content language supported by InfoSleuth is KIF, which is basically treated like Horn clauses, and the deductive database language is LDL++ that is similar to Prolog. An agent service description in IMPACT comprises three parts: a service name, typed input and output variables, and attributes of services. The content language used in IMPACT is an HTML-like language. 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. The specification of CDL, which is to some extent similar to LARKS and also wrapped in a KQML-like message, including 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. The content language in CDL is supported by first-order predicate logic. In a nutshell, an ASDL should provide a specification for describing not only a service name but also a description of what the service-agent needs for performing its service and what changes are made by performing its service.

2.2. Ontologies

The semantics of agent service is the focus in matchmaking process. The explicit representation of the semantics of agent services, accompanied with domain theories (that is, ontology), will enable a middle-agent to spot a relevant service-agent. The issue of representing the semantics of data is addressed well in the area of semantic webs. A set of requirements for an ontology language are outlined in [31]:

- High degree of expressiveness: the language should be expressive to allow agents to annotate complex terms.
• Ability to express semi-structured data: the language should allow agents to mention some details of services they provide or request.
• Support for types and subsumption: the language should support the construction of the relations of service types.
• Ability to express constraints: the language should help express constraints over the parameters of the services.

Some markup languages and frameworks are developed to annotate the meanings of data used in web pages, such as XML [33], RDF Schema [28], and DAML+OIL [10]. Extensible markup language (XML) is a specification for computer-readable documents. It is actually a metalanguage: a mechanism for representing other languages in a standardized way. However, XML does not provide any interpretation of the data beforehand, therefore it does not contribute much to the “semantic” aspect of the semantic web. Resource description framework (RDF) is a mechanism to tell something about data, which contains an object (a resource), an attribute (a property), and a value (a resource or free text). Basically, RDF Schema is a simple-type system for RDF. It provides a mechanism to define domain-specific properties and classes of resources to which you can apply those properties. RDF provides a standard model to describe facts about web resources, which gives sorts of interpretation to the data. RDF Schema extends those interpretation possibilities somewhat more. DAML+OIL (DARPA agent markup language + ontology inference layer) is an expressive ontology description language for markup. Building on top of RDF and RDFS, and with its roots in AI description logics, DAML+OIL overcomes many of the expressiveness inadequacies plaguing RDFS and most importantly, has a well-defined model-theoretic semantics as well as an axiomatic specification that determine the language’s intended interpretations [19].

2.3. Matchmaking

To search for relevant service-agents, a service matchmaking mechanism embedded in middle-agents should take the following requirements into account:
• Semantics-based matching: a matchmaking mechanism should be performed based on the semantics of services and requests rather than their syntax.
• Partial matching: a matchmaking mechanism should be able to find the related services with the possibility to fulfill the request when the middle-agent cannot locate the exact service-agents.

Both the agent-based markup language and the ontology provide a basis for semantics-based matchmaking. But not all of the existing matchmaking mechanisms consider the issue of ontology. The matchmaking in InfoSleuth takes both syntax and semantics into account. Syntactical matching refers to the process of matching requests on the basis of the syntax of services. Semantic matching refers to the process of matching requests with services within the hierarchical data structures. Searching for appropriate services in IMPACT relies on the assignment of a distance from a request in a given weighted verb and noun hierarchies. The RETSINA multiagent infrastructure [30] with an agent service description language (LARKS) contains five different filters to take both syntactic and semantic matchmaking into account. It allows users to configure these filters to achieve the desired tradeoff between matching efficiency and matching quality. Logic-based reasoning over descriptions in CDL relies on the notion of capability subsumption.
3. Possibilistic Petri nets

Possibilistic Petri nets (PPN) [18] is a graphical formalism to serve as a bridge that brings together the possibilistic entailment [17] and Petri nets [24] into a hybrid approach to modeling possibilistic reasoning, which is different from the work done by Cardoso et al. [4]. They propose a possibilistic Petri nets model combining possibility theory and Petri nets theory to lead to a tool for qualitative representation of uncertain knowledge about a system state. They use possibility distributions over all places and tokens to display the uncertainty about possible locations of a token before receiving certain information.

Our PPN offers several benefits for an advanced ASDL:

• PPN is a logical formalism which contains preconditions and postconditions. Furthermore, PPN also has a graphical expression which allow one to visualize the contents of services and requests. Therefore, PPN meets the requirement of expressiveness to be an ASDL.
• PPN is able to cope with uncertainty of providing services because PPN considers partial matching. That is, PPN takes the requirement of flexibility into account.
• The graphical nature of PPN makes easy to integrate several descriptions of services together.
• PPN accompanied with an ontology language also facilitates a semantics-oriented matchmaking because the graphical expression of PPN enables us to easily understand the mechanism.

Before developing possibilistic Petri net-based agent service description language (PPN-ASDL), the background of possibilistic Petri nets is introduced first in the section.

3.1. Possibilistic reasoning

The distinction between imprecise and uncertain information can be best explained by the canonical form representation (i.e. a quadruple of attribute, object, value, confidence) proposed by Dubois and Prade [6,7]. Imprecision implies the absence of a sharp boundary of the value component of the quadruple; whereas, uncertainty is related to the confidence component of the quadruple which is an indication of our reliance about the information. Dubois and Prade [6] have proposed the possibility and necessity measures as an uncertainty model for classical propositions. In this paper, we propose a representation for possibilistic information and an inference mechanism called possibilistic entailment based on Dubois and Prade’s possibility and necessity measures.

To represent uncertain information, we have proposed a representation of possibilistic propositions as follows [17]:

\[ r_1 \in (N_r, \Pi_r), \]

where \( r_1 \) denotes a classical proposition, \( N_r \) denotes the lower bounds of necessity measures, and \( \Pi_r \) denotes the upper bounds of possibility measures (i.e. \( N(r_i) \geq N_r \) and \( \Pi(r_i) \leq \Pi_r \)). A possibilistic proposition \( (r_1, (N_r, \Pi_r)) \) means that the degree of ease to say that \( r \) is false is at most equal to \( 1 - N_r \); meanwhile, the degree of ease to say that \( r \) is true is at most equal to \( \Pi_r \). By definition of possibility measures, \( 1 = \Pi_{r \lor \neg r} = \max(\Pi_r, \Pi_{\neg r}) \) and by duality \( \Pi_{\neg r} = 1 - N_r \), so in this framework \( N_r \) and \( \Pi_r \) are restricted by \( \max(1 - N_r, \Pi_r) = 1 \).
A general formulation of the inference rule which includes multiple antecedents of a rule is expressed as follows:

\[
(r_1 \land r_2 \land \cdots \land r_n) \rightarrow q, (N_{(r_1 \land r_2 \land \cdots \land r_n)} \rightarrow q, \Pi_{(r_1 \land r_2 \land \cdots \land r_n)} \rightarrow q)
\]

\[
r_1, (N_{r_1}, \Pi_{r_1}) \quad r_2, (N_{r_2}, \Pi_{r_2})
\]
\[
\vdots
\]
\[
r_n, (N_{r_n}, \Pi_{r_n})
\]
\[
q, (N_q, \Pi_q)
\]

where \( r_i (i = 1 \sim n) \) and \( q \) are classical propositions, and \((r_1 \land r_2 \land \cdots \land r_n)\) is viewed as a classical proposition \( p \); \( N_r, N_q \) and \( N_{(r_1 \land r_2 \land \cdots \land r_n)} \rightarrow q \) are the lower bounds of necessity measures; and \( \Pi_n \), \( \Pi_q \) and \( \Pi_{(r_1 \land r_2 \land \cdots \land r_n)} \rightarrow q \) are the upper bounds of possibility measures. To infer \( N_q \) and \( \Pi_q \), we have proposed an approach called possibilistic entailment [17], inspired by Nilsson’s probabilistic entailment [21] that helps one deduce a new proposition with associated probability from a base set of propositions with associated probabilities. After performing the possibilistic entailment, we can derive the conclusions below:

\[
N_q = \min\{\max[N_{(r_1 \land r_2 \land \cdots \land r_n)} \rightarrow q, 1 - \Pi_r], \max[1 - \Pi_{(r_1 \land r_2 \land \cdots \land r_n)} \rightarrow q, N_r]\},
\]
\[
\Pi_q = \max\{\min[\Pi_{(r_1 \land r_2 \land \cdots \land r_n)} \rightarrow q, \Pi_r], \min[\Pi_{(r_1 \land r_2 \land \cdots \land r_n)} \rightarrow q, 1 - N_r]\},
\]

where \( N_r = \min[N_{r_1}, N_{r_2}, \ldots, N_{r_n}] \) and \( \Pi_r = \min[\Pi_{r_1}, \Pi_{r_2}, \ldots, \Pi_{r_n}] \). In the case that \( \Pi_r < 1 \) and \( \Pi_{(r_1 \land r_2 \land \cdots \land r_n)} \rightarrow q < 1 \) do not exist simultaneously, Eq. (3) then becomes

\[
N_q = \min\{N_{(r_1 \land r_2 \land \cdots \land r_n)} \rightarrow q, N_r\},
\]
\[
\Pi_q = \Pi_{(r_1 \land r_2 \land \cdots \land r_n)} \rightarrow q.
\]

When several rules having a same conclusion are fired, these inferred conclusions with different confidence levels, for example \((q, (N'_q, \Pi'_q)) (i = 1 \sim n)\), should be aggregated as a conclusion \((q, (N'^{n+1}_q, \Pi'^{n+1}_q))\). This aggregation can be viewed as a disjunction; we then obtain \( N'^{n+1}_q = \max[N'^1_q, N'^2_q, \ldots, N'^n_q] \) and \( \Pi'^{n+1}_q = \max[\Pi'^1_q, \Pi'^2_q, \ldots, \Pi'^n_q] \).

Possibilistic logic has been advocated by Dubois and Prade for more than 10 years. In their approach, they viewed a possibility measure \( \Pi(p) \) of proposition \( p \) as the degree of possibility that \( p \) is true, and a necessity measure \( N(p) \) as the degree of necessity that \( p \) is true. A comparison between their approach and ours is discussed in our previous work [17].

### 3.2. Petri nets

A Petri net is a directed, weighted, bipartite graph consisting of two kinds of nodes, called places \((p_i)\) and transitions \((t_i)\), where arcs are either from a place to a transition or from a transition to a place [23]. Murata has formally defined Petri nets as a 5-tuple [20]: \( \text{PN} = (P, T, A, W, M_0) \), where \( P = \{p_1, p_2, \ldots, p_m\} \) is a finite set of places, \( T = \{t_1, t_2, \ldots, t_n\} \) is a finite set of transitions, \( A \subseteq (P \times T) \cup (T \times P) \) is a set of arcs, \( W : A \rightarrow \{1, 2, 3, \ldots,\} \) is a weight function, and \( M_0 : P \rightarrow \{0, 1, 2, 3, \ldots,\} \) is the initial marking. A marking \( M \) is an \( m \)-vector, \( \{M(p_1), \ldots, M(p_m)\} \), where \( M(p_i) \) denotes
the number of the tokens in place \( p_i \). The evolution of markings, used to simulate the dynamic behavior of a system, is based on the firing rule, such as a transition \( t \) is enabled if each input place \( t \) is marked with at least \( w(p, t) \) tokens, where \( w(p, t) \) is the weight of the arc from \( p \) to \( t \); an enabled transition may or may not be fired; and a firing of an enabled transition \( t \) removes \( w(p, t) \) tokens from each input place \( p \) of \( t \), and adds \( w(t, p) \) tokens to each output place \( p \) of \( t \), where \( w(t, p) \) is the weight of the arc from \( t \) to \( p \). A place having two or more output transitions is referred to as a conflict.

3.3. Possibilistic Petri nets (PPN)

The simple view of a system concentrates on two primitive concepts: events and conditions. An event is an action which takes place in the system. A condition is a predicate or logical description of the state of the systems [23]. A typical interpretation of Petri nets is to view a place as a condition, a transition as the causal connectivity of conditions (an event), and a token in a place as a fact used to claim the truth of the condition associated with the place. However, the confidence about the connectivity of conditions and the facts could be uncertain. To take the uncertain situations into account, we formally define a possibilistic Petri nets as follows.

**Definition 1.** A possibilistic Petri net (PPN) is defined as a five-tuple,

\[
\text{PPN} = (P, PT, A, W, M_0)
\]

\( P = \{P_1(r_1), P_2(r_2), \ldots, P_m(r_m)\} \) is a finite set of places, where place \( P_i \) represents a classical proposition \( r_i \). \( PT = \{t_1(N_1, \Pi_1), t_2(N_2, \Pi_2), \ldots, t_n(N_n, \Pi_n)\} \) is a finite set of possibilistic transitions, where \( t_j \) represent the connectivity of places, \( N_j \) denotes the lower bounds of necessity measures, and \( \Pi_j \) denotes the upper bounds of possibility measures to represent the uncertainty about the connectivity between places. \( A \subseteq (P \times PT) \cup (PT \times P) \) is a set of arcs. \( W : A \rightarrow \{1, 2, 3, \ldots, \} \) is a weight function. \( M_0 = \{M(P_1), M(P_2), \ldots, M(P_m)\} \) is the initial marking, where \( M(P_i) \) is the number of tokens in \( P_i \).

Each token is associated with a pair of necessity and possibility measures \((N_i, \Pi_i)\) (called a possibilistic token) and inserted into the related place that represents a classical proposition. To simulate the dynamic behavior of a possibilistic system, a marking in a PPN is changed according to the firing rule: a firing of an enabled possibilistic transition \( t_j \) removes the possibilistic tokens from each input place \( P_i \) of \( t_j \), adds a new token to each output place \( P_k \) of \( t_j \), and the necessity and possibility measures attached to the new token will be computed based on the possibilistic reasoning. A simple example of PPN is illustrated in Fig. 2.

3.4. Possibilistic reasoning and PPN

Many researchers have devoted themselves into modeling rule-based reasoning via Petri nets [2,14,27] or extensions to Petri nets [3]. There are several rationales behind to base a computational paradigm for rule-based reasoning on Petri net theory:

- Petri nets achieve the structuring of knowledge within rule bases, which can express the relationships among rules and help experts construct and modify rule bases.
• Petri net’s graphic nature provides the visualization of dynamic behavior of rule-based reasoning.
• Petri nets make it easier to design efficient reasoning algorithm.
• Petri net’s analytic capability provides a basis for developing knowledge verification technique.
• Petri nets can model the underlying relationship of concurrency among rules activation, which is an important aspect where real-time performance is crucial.

3.4.1. Knowledge representation

The three key components in uncertain rule-based reasoning: propositions, uncertain rules and uncertain facts, can be formulated as places, possibilistic transitions and possibilistic tokens, respectively. The mapping between possibilistic reasoning and PPN is described as follows:

• **Places**—Places correspond to classical propositions. The classical propositions that are attached to places represent conditions.

• **Possibilistic tokens**—A possibilistic token represents an uncertain fact. A pair of necessity and possibility measures are attached to possibilistic tokens to represent our confidence level about the observed facts.

• **Possibilistic transitions**—Possibilistic transitions are classified into four types: inference, aggregation, duplication, and aggregation-duplication transitions. The inference transitions represent the uncertain rules, the aggregation transitions are designed to aggregate the conclusion parts of rules which have the same classical propositions, the duplication transitions are used to duplicate possibilistic tokens to avoid the conflict problem, and the aggregation-duplication transitions link the same classical propositions. These are formally defined below.

**Type 1 (Inference transition (ti))**: An inference transition serves as modeling of an uncertain rule. An uncertain rule having multiple antecedents is represented as

\[(r_1 \land r_2 \land \cdots \land r_n) \rightarrow q, (N_1, \Pi_1),\]

where \(r_i\) and \(q\) are classical propositions. In Fig. 3, after firing the inference transition \(ti\), the tokens will be removed from the input places of \(ti\), a new token will be deposited into the output place.
Fig. 3. Modeling possibilistic reasoning through PPN: (a) before firing inference transition $t^i$, (b) after firing $t^i$.

Fig. 4. Modeling the aggregation of conclusions by an aggregation transition: (a) before firing the aggregation transition $t^{a}_{m+1}$, (b) after firing $t^{a}_{m+1}$.

of $t^i$, and a pair of necessity and possibility measures attached to the new token are derived by the possibilistic reasoning (see Section 3.1).

Type 2 (Aggregation transition ($t^a$)): An aggregation transition is used to aggregate the conclusions of several uncertain rules which have a same classical proposition, and to link the antecedent of an uncertain rule which also have the same classical proposition. For example, there are $m$ uncertain rules with the same classical proposition $q$ in the conclusions, denoted as

$$ (r_1 \rightarrow q, (N_1, \Pi_1)), (r_2 \rightarrow q, (N_2, \Pi_2)), \ldots, (r_m \rightarrow q, (N_m, \Pi_m)). $$

(7)

In Fig. 4, after firing the aggregation transition $t^{a}_{m+1}$, the tokens in input places of $t^{a}_{m+1}$ will be removed, a new token will be deposited into the output place of $t^{a}_{m+1}$, and a pair of necessity and possibility measures attached to the new token are derived by disjunction (see Section 3.1). It should be noted that $t^{a}_{m+1}$ is dead if one of its input places never received a token. To avoid deadlock in aggregation transitions, we assume that for each source place $P_i$, a token will be inserted into $P_i$, 


Fig. 5. Modeling the duplication of a possibilistic token through PPN: (a) before firing the duplication transition \( t^d_1 \), (b) after firing \( t^d_1 \).

Fig. 6. Modeling the aggregation–duplication of possibilistic tokens through PPN: (a) before firing the aggregation–duplication transition \( t^{ad}_1 \), (b) after firing \( t^{ad}_1 \).

and a pair of necessity and possibility \((0,1)\) is attached to represent ignorance if no fact refers to the proposition in place \( P_i \).

**Type 3 (Duplication transition \((t^d)\))**: The purpose of duplication transitions is to avoid the conflict by duplicating the token. For example, there are \( l \) uncertain rules with a same classical proposition \( r \) in the antecedents, denoted as

\[
(r \rightarrow q_1,(N_1,\Pi_1)), (r \rightarrow q_2,(N_2,\Pi_2)), \ldots, (r \rightarrow q_l,(N_l,\Pi_l)).
\]  

They are linked by a duplication transition shown in Fig. 5. After firing the duplication transition \( t^d_1 \), the tokens in the input place of \( t^d_1 \) will be removed, new tokens will be added into the output places of \( t^d_1 \), and a pair of necessity and possibility measures attached to the new tokens are not changed.

**Type 4 (Aggregation–duplication transition \((t^{ad})\))**: An aggregation–duplication transition is a combination of an aggregation transition and a duplication transition (see Fig. 6). It is used to link all the same classical propositions. For example, there are \( m \) uncertain rules with a same classical proposition \( q \) in the conclusions and \( l \) uncertain rules with the same classical proposition \( q \) in the antecedents, denoted as

\[
(r_1 \rightarrow q_1,(N_1,\Pi_1)), (r_2 \rightarrow q_1,(N_2,\Pi_2)), \ldots, (r_m \rightarrow q_1,(N_m,\Pi_m)),
\]

\[
(q \rightarrow s_1,(N_{m+1},\Pi_{m+1})), (q \rightarrow s_2,(N_{m+2},\Pi_{m+2})), \ldots, (q \rightarrow s_l,(N_{m+l},\Pi_{m+l})).
\]  

(9)
They are linked by an aggregation–duplication transition shown in Fig. 6. After firing the aggregation–duplication transition \( t_{\text{ad}}^1 \), the tokens in the input places of \( t_{\text{ad}}^1 \) will be removed, new tokens will be deposited into the output places of \( t_{\text{ad}}^1 \), and the pair of necessity and possibility measures attached to the new tokens are derived by disjunction.

4. Using PPN as an agent service description language

In order to implement the service-oriented framework (outlined in Section 2), we propose the use of PPN as a basis for ASDL, the adoption of DAML+OIL ontology markup language for sharing domain knowledge, and a service matchmaking mechanism that is embedded in the middle-agents and capable of dealing with uncertainty. An overview of the proposed approach is shown in Fig. 7.

4.1. Specification

Possibilistic Petri net-based agent service description language (PPN-ASDL) is a specification that guides agents to publish, request and match services in an open information environment. PPN-ASDL has the following features (see Fig. 8):

- **Confident transition** \( (t^i) \): This is an inference transition associated with complete confidence, that is, both the degrees of necessity and possibility are 1. In Petri nets, a transition is considered as an event or action that can derive the postconditions once its preconditions hold. Therefore, performing a service is viewed as an event or action that can achieve the postconditions (services) once its preconditions hold.

Fig. 7. Service-oriented framework for multiagent systems using PPN-ASDL.
• **Input variables** (⟨x⟩): An expression contains variables that the actual inputs will need to match and is attached to input arcs of a possibilistic transition.

• **Output variables** (⟨x⟩): An expression contains variables that the actual outputs will need to match and is attached to output arcs of a possibilistic transition.

• **Input places** (IP(r)): The preconditions are represented in a logical expression and expected to hold in this situation before a service can be performed (free variables in these conditions can only be from the inputs).

• **Output places** (OP(q)): The postconditions are represented in a logical expression and expected to hold in this situation after the services can be performed (free variables in these conditions can only be from the outputs).

A simple example is illustrated to demonstrate the use of PPN-ASDL as follows. Supposed that there is a mechanic who works in Taipei City and specializes in fixing vans. His problem-solving capability can be represented by a service-agent shown in Fig. 9(a). The two input places describe the preconditions of providing his service, and the output place denotes the situation after performing his service. Let us suppose there is a man whose truck is broken in Taoyuan City (i.e. about 30km away from Taipei City) and requests a service to repair his truck. His request is represented by a request-agent shown in Fig. 9(b). In service-oriented framework for multiagent systems, the messages that different agents need to exchange are expressed in a high-level communication language, knowledge query and manipulation language (KQML). In this example, an ASDL embedded in the content part of KQML is illustrated in Fig. 10: Fig. 10(a) is the message issued by a service-agent to publish its service to the middle-agent, and Fig. 10(b) is the message sent by a request-agent to request for a service to the middle-agent. Once the middle-agent receives the request, it will search for the
relevant services within its service registry for the request through its service matchmaking mechanism with also class hierarchies.

4.2. Ontology for ASDL

An ontology defines a common vocabulary for users who need to share information in a domain. It includes machine-interpretable definitions of basic concepts in the domain and relations among them. We usually attribute the notion of ontology to the specification of a conceptualization, that is, defined terms and relations between them, usually in formal and preferably machine-readable manner. There are a number of reasons why ontologies need to be developed [11,22]: (1) to help a matchmaking mechanism find the most appropriate services based on their semantics, (2) to share common understanding of the structure of information among people or software agents, (3) to enable reuse of domain knowledge, (4) to make domain assumptions explicit, (5) to separate domain knowledge from the operational knowledge, and (6) to analyze domain knowledge. In this paper, by creating ASDL associated with an ontological language, we are able to use hierarchy (and property restrictions) to find matches through class and subclass properties or other semantic links.

4.2.1. DAML+OIL ontology markup language

An ontology language could be easily used to define an ontology—not of services but of the terms needed to describe the invocation of services. In this paper, DAML+OIL is chosen as an ontological language to encode the classes and subclasses of concepts and relations pertaining to services and users constraints. This is mainly because it is built on top of RDF and RDFS, and with its roots in AI description logics. It also overcomes many of the expressiveness inadequacies plaguing RDFS and most important, has a well-defined model-theoretic semantics as well as an axiomatic specification that determines the language’s intended interpretations [19]. Let us consider the previous example again. As was shown in Fig. 11, we use DAML+OIL to annotate its ontology because the semantics of van varies in different domain. In Fig. 11, we know that the class Van is a subclass of Automobile and disjoint from the class Truck, and its primary property is to have an enclosed box-like look.
Fig. 11. The ontology of van via DAML+OIL.

Fig. 12. (a) A class hierarchy of transportation. (b) A class hierarchy of world.

4.2.2. Class hierarchy

In practical terms, developing an ontology not only includes using ontology markup language but also arranging the classes in a taxonomic (subclass-superclass) hierarchy [22]. There are several possible approaches in developing a class hierarchy [22]. First, a top-down development process starts with the definition of the most general concepts in the domain and subsequent specialization of the concepts. Second, a bottom-up development process starts with the definition of most specific classes, the leaves of the hierarchy, with subsequent grouping of these classes into more general concepts. Finally, a combination development process is a combination of the top-down and bottom-up approaches. The more salient concepts are defined first and then generalized and specialized appropriately. None of these three methods is inherently better than any of the others. The approach to take depends strongly on the personal view of the domain. If a developer has a systematic top-down view of the domain, then it may be easier to use the top-down approach. The combination approach is often the easiest for many ontology developers, since the concepts in the middle tend to be the more descriptive concepts in the domain. In the previous example, there are two class hierarchies needed to represent the relations of classes related to transportation and geography (see Fig. 12).

4.3. The service matchmaking mechanism for PPN-ASDL

The matchmaking mechanism, which is embedded in the middle-agents, is used to determine whether a service provided by a service-agent matches a request issued by a request-agent. To
permit a relaxed match for close semantics, we use degrees of possibility and necessity to represent our confidence levels that a service-agent can provide the relevant service for request-agent.

4.3.1. Defining possibility and necessity measures between two classes

To compare service and request based on their semantics, the matchmaking mechanism quantifies the confidence levels (i.e. the degrees of possibility and necessity) of matching two classes by computing a similarity between two classes in a class hierarchy [15]. Before introducing the formal definition of similarity, a similarity between two adjacent superclass and subclass is first described below.

**Definition 2 (The similarity between two adjacent superclass and subclass).** Let the similarity between two adjacent superclass $c_1$ and subclass $c_2$ be denoted by $S(c_1, c_2)$. We have

$$S(c_1, c_2) : S \times S \rightarrow [0, 1]$$

assigning to each pair of superclass and subclass $(c_1, c_2)$ a unique degree of similarity between 0 (corresponding to maximum dissimilarity) and 1 (corresponding to maximum similarity).

For example, the similarity between the classes Automobile and Van is assigned as 0.7 (see Fig. 12(a)).

The most specific common superclass of two classes $c_1$ and $c_2$ indicates the most specific class that subsumes $c_1$ and $c_2$. Its formal definition is described below.

**Definition 3 (The most specific common superclass of two classes).** Let the most specific common superclass of two classes $c_1$ and $c_2$ be denoted by $G_t(c_1, c_2)$. We have

$$G_t(c_1, c_2) = \{ g \mid \text{general}(g, c_1) \land \text{general}(g, c_2) \land (\forall c \neq g)(\text{general}(c, c_1) \land \text{general}(c, c_2)) \Rightarrow \text{general}(c, g) \},$$

where $\text{general}(x, y)$ is a predicate to indicate that $x$ is more general than $y$ (i.e. $x$ is a superclass of $y$).

It should be noted that we cannot find the most specific common superclass if two classes $c_1$ and $c_2$ are located in two different class hierarchies, that is, $G_t(c_1, c_2)$ does not exist. For example, in Fig. 12(a), the most specific common superclass of the classes Van and Truck is the class Automobile (i.e. $G_t($Van, Truck$) =$Automobile), and the most specific common superclass of the classes Van and Taipei does not exist.

In class hierarchies, the similarity between two classes is evaluated by multiply similarities from each class to their most specific common superclass in the class hierarchy.

**Definition 4 (The similarity between two classes).** Let the similarity between two classes $c_1$ and $c_2$ be denoted by $S(c_1, c_2)$. $S(c_1, c_2)$ is defined as the similarity of the shortest path from $c_1$ to $c_2$ in the class hierarchy. We have:

1. $S(c_1, c_2) = S(c_1, G_t(c_1, c_2)) \land S(c_2, G_t(c_1, c_2))$. 
2. \( S(c_1, c_2) = 1 \) if \( c_1 = c_2 \).

3. \( S(c_1, c_2) = 0 \) if \( G_t(c_1, c_2) \) does not exist.

For example, in Fig. 12(a), the similarity of the classes Van and Truck is \( S(\text{Van}, \text{Automobile}) \times S(\text{Truck}, \text{Automobile}) \), that is, 0.7 \( \times \) 0.8.

Inspired by the concepts of degrees of consistence and implication proposed by Ruspini [26], we define our degrees of consistence and implication between two classes to serve as a measure of the confidence levels that a service can fulfill a request.

**Definition 5** *(Degree of consistence between two classes)*. Let the degree of consistence of class \( c_1 \) implying class \( c_2 \) be denoted by \( C(c_1 \rightarrow c_2) \). We have:

1. \( C(c_1 \rightarrow c_2) = 1 \) if \( G_t(c_1, c_2) \) exists.
2. \( C(c_1 \rightarrow c_2) = 0 \) if \( G_t(c_1, c_2) \) does not exist.

It should be noted that the degree of consistence between two classes is 1 if we can find their most specific common superclass. However, the degree of consistence between two classes is 0 if they are located in two different class hierarchies. For example, in Fig. 12(a), the degree of consistence \( C(\text{Truck} \rightarrow \text{Van}) \) is 1 and the degree of consistence \( C(\text{Truck} \rightarrow \text{Taiwan}) \) is 0. In fact, the degree of consistence between two classes can be considered as the possibility measure [26]. Therefore, we define the possibility measure between two classes as the degree of consistence, that is, \( \Pi(c_1 \rightarrow c_2) = C(c_1 \rightarrow c_2) \).

**Definition 6** *(Degree of implication between two classes)*. Let the degree of implication of class \( c_1 \) implying class \( c_2 \) be denoted by \( I(c_1 \rightarrow c_2) \). We have:

1. \( I(c_1 \rightarrow c_2) = 1 \) if \( G_t(c_1, c_2) = c_2 \).
2. \( I(c_1 \rightarrow c_2) = S(c_1, c_2) \) if (a) \( G_t(c_1, c_2) \neq c_2 \) and (b) \( c_1 \) and \( c_2 \) are in the same class hierarchy.
3. \( I(c_1 \rightarrow c_2) = 0 \) if \( c_1 \) and \( c_2 \) are located in two different class hierarchies.

It should be noted that the degree of implication between two classes is 1 if subclass implies super-class, otherwise, the degree of implication between two classes is the similarity of the two classes. At the same time, the degree of implication between two classes is 0 if they are located in two different class hierarchies. For example, in Fig. 12(a), the degree of implication \( I(\text{Van} \rightarrow \text{Automobile}) \) is 1, the degree of implication \( I(\text{Automobile} \rightarrow \text{Van}) \) is 0.7, and the degree of implication \( I(\text{Van} \rightarrow \text{Truck}) \) is \( S(\text{Van}, \text{Truck}) = 0.56 \). In fact, the degree of implication between two classes can be considered as the necessity measure [26]. Therefore, we define the necessity measure between two classes as the degree of implication, that is, \( N(c_1 \rightarrow c_2) = I(c_1 \rightarrow c_2) \).

### 4.3.2. Matching algorithm

Consider a service and a request in Fig. 13, in which the request has \( m \) input places describing the current situation before receiving services and the output place representing the need of service. A service, which has \( n \) input places and \( n \) is not greater than \( m \), is chosen to match the request by the following matching algorithm.
Fig. 13. Matching request with service.

**Algorithm 1** (Matching requests with services).

1. Find the appropriate locations in class hierarchies for the classes \(cq_1\) and \(cq_2\) mentioned in the output places of service \(OP_1(q_1)\) and request \(OP_2(q_2)\) based on both syntactic and semantic matchings. (i.e. the syntactic matching searches for the same class name in class hierarchies and then the semantic matching further checks whether the found class has the same properties with \(cq_1\) or \(cq_2\).)

2. Select another service to match the request and go back to Step 1 if the classes \(cq_1\) and \(cq_2\) are located in two different class hierarchies, that is, they do not belong to the same ontology.

3. Compute the degrees of the possibility \(\Pi\) and necessity \(N\) that the output place of service \(OP_1(q_1)\) can infer the output place of request \(OP_2(q_2)\). That is, \(\Pi = C(cq_2 \rightarrow cq_1)\) and \(N = I(cq_2 \rightarrow cq_1)\). (i.e. it should be noted that the service-agent can provide more services than what the request-agent needs if \(cq_2\) is a subclass of \(cq_1\). This step is to calculate our confidence level that a service-agent is capable of fulfilling the request if all the preconditions of performing the service hold.)

4. Find the appropriate locations in class hierarchies for the classes \(cr_k\) and \(cr_l\) mentioned in the input places of service \(IP_k(r_k)\) and request \(IP_l(r_l)\) (i.e. \(k = 1 \sim n\) and \(l = n + 1 \sim n + m\)) based on both syntactic and semantic matchings.

5. Select another service to match request and go back to Step 1 if any pair of \((cr_k\) and \(cr_l)\) does not have the same class hierarchy, that is, the present service cannot be satisfied.

6. Compute the degrees of the possibility \(\Pi_k\) and necessity \(N_k\) that the input place of request \(IP_l(r_l)\) can infer the input place of service \(IP_k(r_k)\), where \(k = 1 \sim n\), \(1 \leq l \leq m\), and \(cr_k\) and \(cr_l\) have the same class hierarchy. That is, \(\Pi_k = C(cr_l \rightarrow cr_k)\) and \(N = I(cr_l \rightarrow cr_k)\). (i.e. this step is to calculate our confidence level about the satisfying of the preconditions to perform the service.)

7. Insert certain tokens (i.e. \((N, \Pi) = (1, 1)\)) into all input places of the request to represent the existing preconditions and then perform the possibilistic reasoning through the inference transitions \(t_4 \sim t_{n+3}\) (their input places are the input places of the request and their output places are the input places of the service), where the inference transitions are attached with the degree of possibility and necessity derived from Step 6.
8. Perform the possibilistic reasoning through the confident transition of the service $t^i_1$. The derived degrees of possibility and necessity represent our confidence level to make sure that the service can be performed under partially matched preconditions. Then carry out the possibilistic reasoning through an inference transition $t^i_3$ (its input place $OP_1(q_1)$ is the output place of the service and its output place $OP_2(q_2)$ is the output place of the request), where the inference transition is attached with the degree of possibility and necessity derived from Step 3. The derived degrees of possibility and necessity are considered as the confidence that service can fulfill the request.

It should be noted that the algorithm makes each input place of the confident transition of the request conflict (i.e. a place with two output arcs) and then guides the certain tokens in these input places to fire the inference transitions from $t^i_4$ to $t^i_{n+3}$ rather than $t^i_j$. The design of “bypass” has the following features: (1) to calculate our confidence level that a service-agent is capable of fulfilling the request if all the preconditions of performing the service hold, (2) to infer our confidence level about satisfying the preconditions to perform the service, (3) to derive our confidence level that the service can be performed under partially matched preconditions, and (4) to compute our confidence level that service can fulfill the request.

Let us consider the previous example (see Fig. 14). First, we find the appropriate locations in class hierarchies in Fig. 12 for the classes Van and Truck mentioned in the output places of service $OP_1(q_1)$ and request $OP_2(q_2)$ based on the properties of the classes. Second, the degrees of the necessity and possibility for $(OP_1(q_1) \rightarrow OP_2(q_2))$ are calculated as $(0.56, 1)$, which means the confidence level that the mechanic is capable of fixing the broken truck if all the preconditions of performing the service hold. Third, we find the appropriate locations in class hierarchies in Fig. 12 for the classes Van, Truck, Taipei and Taoyuan, which are mentioned in the input places of the service and the request based on the properties of the classes. Fourth, the degrees of the necessity and possibility for $(IP_3(r_3) \rightarrow IP_1(r_1))$ and $(IP_4(r_4) \rightarrow IP_2(r_2))$ are computed as $(0.56, 1)$ and $(0.63, 1)$, respectively, which denote the confidence levels about the satisfying of the preconditions to perform the service. Finally, the possibilistic transitions $t^i_4$, $t^i_5$, $t^i_1$ and $t^i_3$ are triggered sequentially, and we thus obtain that the degrees of necessity and possibility to serve the request by the service-agent is $(0.56, 1)$. Namely, there is a possibility that the mechanic will go to Taoyuan City and repair the broken truck.
although he is good at fixing vans. However, the degree of necessity that he will accomplish this request is only 0.56.

Consider a more complex situation in this example: four possible services for the request. Besides the previously mentioned mechanic (say mechanic A) who works in Taipei City and specializes in fixing vans, a mechanic B works in Taichung and is good at fixing truck, a mechanic C works in Taoyuan and can repair car, and a tow truck serves in Taoyuan. In the case of the mechanic B, the degrees of the necessity and possibility for \((\text{OP}_1(q_1) \rightarrow \text{OP}_2(q_2))\) are calculated as \((1,1)\) because the mechanic’s capability fulfills the request exactly. The degrees of the necessity and possibility for \((\text{IP}_3(r_3) \rightarrow \text{IP}_1(r_1))\) and \((\text{IP}_4(r_4) \rightarrow \text{IP}_2(r_2))\) are computed as \((1,1)\) and \((0.56,1)\), respectively. The possibilistic transitions \(t_4^1, t_5^1, t_1^1\) and \(t_2^1\) are triggered sequentially, and we thus obtain that the degrees of necessity and possibility to serve the request by the service-agent B are \((0.56,1)\). Namely, there is a possibility that the mechanic will go to Taoyuan City and repair the broken truck. However, the degree of necessity that he will go to Taoyuan City is only 0.56. In the case of the mechanic C, the degrees of the necessity and possibility for \((\text{OP}_1(q_1) \rightarrow \text{OP}_2(q_2))\) are calculated as \((0.72,1)\) because the similarity between Truck and Car is 0.72. The degrees of the necessity and possibility for \((\text{IP}_3(r_3) \rightarrow \text{IP}_1(r_1))\) and \((\text{IP}_4(r_4) \rightarrow \text{IP}_2(r_2))\) are computed as \((0.72,1)\) and \((1,1)\), respectively. The possibilistic transitions \(t_4^1, t_5^1, t_1^1\) and \(t_2^1\) are triggered sequentially, and we thus obtain that the degrees of necessity and possibility to serve the request by the service-agent C are \((0.72,1)\). In other words, there is a possibility that the mechanic will repair the broken truck. The degree of necessity that he will accomplish this request is 0.72. As for the tow truck, it will not be considered here because it provides irrelevant service for the request. Finally, the mechanic C will be chosen because it has the highest degree of necessity measure to accomplish this request.

Let us consider another scenario: adding a car repair shop located in Taoyuan to this example. The request will also be fulfilled if the tow truck and the car repair shop serve this request in a collaborative manner, which is related to the issue of coordination. The coordination can be roughly divided into three categories: chain-type, parallel-type, and a hybrid one. The chain-type coordination means that several service-agents accomplish a request by performing their services sequentially, that is, a service-agent’s output place is another service-agent’s input place to form a chain of service-agents. The parallel-type coordination represents that several service-agents fulfill a request collaboratively and simultaneously, that is, each service-agent deals with part of the whole request. The third type of coordination is a hybrid of the previous two types, which will be addressed in our future work.

5. Conclusion

Possibilistic Petri net-based agent service description language (PPN-ASDL) is proposed in this paper as a specification of publishing and requesting services for service-oriented framework for multiagent systems. PPN-ASDL offers several benefits:

- Equipped with expressive power due to the use of a logical formalism with preconditions and postconditions, and its graphical expression to visualize the contents of services.
- Capable of handling uncertainty through the PPN to represent the uncertainty about fulfilling preconditions of providing services.
Easy to integrate a number of service together because of the inherited nature of Petri nets.

Power with semantics realized by adopting DAML+OIL ontology markup language to facilitate a semantics-based service matchmaking.

A service matchmaking mechanism embedded in the middle-agents, with class hierarchies is also developed to search for services among service-agents for request-agents. The proposed matchmaking mechanism permits a relaxed match for close semantics through the use degrees of possibility and necessity to represent the confidence levels that a service-agent can provide a particular service for a request-agent.

We leave the issue of service coordination in a service-oriented framework for multiagent systems (see Section 2) to our future work. That is, our future research plan will consider the service composition of several service-agents to collaboratively perform a task of the request.

Acknowledgements

This research was sponsored by the Ministry of Education (Taiwan, ROC) under Grants EX-91-E-FA06-4-4.

References