Web Service Composition based on Agent Societies and Ontological Concepts

Faezeh Ensan1, Mohsen Kahani2, Ebrahim Bagheri1
Faezeh.ensan@unb.ca, kahani@um.ac.ir, E.Bagheri@unb.ca
1 Department of Computer Science, University of New Brunswick, Canada
2Computer Engineering Department, Ferdowsi University of Mashhad, Iran

Abstract.
Pervasive computing environments have overwhelmed humans with a vast amount of information resulting in a lost in the hyperspace problem. Different methods have been proposed to structure and provide better information retrieval. The most recent developments have been the creation of the semantic web and the web services. The new challenge is to establish the basis for semantic web services that are able to be organized into a chain which satisfies the desired user functionality. In this paper we propose a framework in which multi-agent societies have been used to create an environment in which user requests are received (in a restricted English grammar format) and a suitable web service composition is formed. The framework targets distributed environments where no central web service registry is available.

Keywords:
Logical Reasoning, First Order Logic, Automated Web Service Composition, Intent Verbalization, Multi-Agent Societies.

1. Introduction

The World Wide Web has been experiencing great evolution since its early date of birth in the 90’s. Having an unstructured growth of the current web is one of the most important reasons in having a decline in the usage interest. On the other hand, the lack of suitable building blocks to construct the information on the web has allowed anarchy in the spread of information all over the web.

In this situation finding suitable information, services and deriving the appropriate results from the available content has turned into a dilemma. Using web crawlers and search engines to help finding the proper data has alleviated the situation; although the continuation of such growth will deprive the classification power from the search engines leaving them in a jam.

One of the main solutions to this problem is to create meaning for the current structures and content available on the web so that each information substance can describe itself through the metadata that it is carrying. XML and related schema creation and validation methods were initially created and ongoing research in this area has been followed. Later the idea of taxonomy to classify concepts and to explain the principles underlying the classification was born.

The broader concept of ontology was established to formulate a domain structure containing its entities, the available relationships and the governing rules. Ontologies are usually hierarchically formed and are mostly described for a specific domain. The created ontologies depending on the computer ability to process them can be classified as weak or strong. Ontologies described in languages such as OWL [1], OIL [2], DAML [3], and DAML+OIL [4] can be categorized as strong ontologies because they are completely machine interpretable. The before mentioned structures create the basis of what is now known as the semantic web [5] which aims to increase reachability and connectivity along with understandability to the current available resources.

Although the move towards the semantic web is paving the way for a better structure of information, the lack of a proper programming structure to provide services was also a great deficiency. Creating packed logic, as in software components, that can work in the World Wide Web can enhance the current situation and interoperability through interface unification and open standards. Web services have been developed to answer such a need and provide suitable capability for software boxing and distribution. The web service technology has a protocol stack that consists of WSDL, UDDI, and SOAP and all messages passing is done through XML documents. WSDL is used to describe the communication principles of the web service. UDDI allows other applications to look up specific web services functionality and allows a uniform description and discovery model for the web services. The last functionality, SOAP, provides the ability to pass XML message on the web over the HTTP protocol. The web service model is now supported by many programming languages such as Java, Mirossoft .NET, PHP, Python, and etc.

The need for exploring the web has created many different algorithms. One of the technologies that have been of much attention is the use of agent societies. An entity should have different features to satisfy the agent definition which are autonomy, intelligence, flexibility, and rationality [6]. There have been different definitions for an agent that can be applied in different circumstances.

In this paper we aim to create a framework, to allow the user to interact with the provided interface and express his will in a human understandable language (e.g. English
Grammar and Vocabulary). The framework will have the ability to translate user intent into first order logic axioms, and detect the users’ will. The user objective will then be defined by a tuple composed of the user inputs, outputs and his desired constraint.

The paper is composed of 4 sections. Section 2 will detail the methodology used in the framework creation and explains the details of the design. Section 3 concludes the paper and summaries the main points.

2. The Proposed Framework

Although there are a wealth of available techniques, frameworks, tools, and algorithms available for web service composition [7, 8, and 9], there still exist many reasons to motivate researchers to tackle the problem and provide new solutions. The main stimuli for our proposed framework can be addressed in four categories; however there are still open areas that will be considered in the next versions of the framework.

The main concern in creating this framework was to create an interface that can interact with the user and transfer user language commands into a logic based language that could allow reasoning. Such interface could not be found in any other systems and allows users to easily communicate with the planner and declare their needs.

The second problem is dealing with partial solutions. There are times when the planner (Logical Reasoner) can not infer any suitable solutions based on the preliminary state for the users’ needs. The planner will in these situations provide partial solutions to the users in our proposed framework. Finding appropriate web services according to the ontological domain of the problem was also the other aim of the framework. Our platform also provides the means to check parameter ranges in the obtained solutions. For example if the user requests a trip schedule from his home in Tehran to Montreal, the planner should only use web services which have the related records to buy a ticket from Tehran to London and from London to Montreal, and a web service which only support US domestic flights will be of no use.

The proposed framework consists of 3 main parts: User edge, WSComposer, Ontology Handler. The user edge is a component which has the most interaction with the user. The main responsibility of the user edge is receiving user commands in a verbalized form. The inputs are then transformed into first order logic axioms and the user intent, input and outputs are detected. The user inputs are imagined to be the initial state and can used to describe the user world. The required outputs are used as his desired final state and his intent is the high level plan that should be minutely decomposed using logical inference. The third component is used to create and manage ontology based agents that keep track of new web services based on an ant routing algorithm and classifies the web services in an ontology hierarchy. The framework has been depicted in Figure 1.

2.1 Web Service Registration

Our model only handles the syntactic features of a web service. The composability is thus only checked on the syntax of the available web services: Binding, Operation Mode, Messages [10]. Upon the addition of the newly attached web service to a UDDI repository no special event will occur. The only tasks that must be performed are the ordinary registration process therefore avoiding extra execution time on the whole system. The web service semantic conceptualization will be explained later on in the paper.

2.2 Web Service Composition

The automatic web service composition process comprises of different stages that will be minutely explained. The framework that is proposed for WSC does not only deal with complexities related to the discovery process but also handles user interaction. The purpose of a
Definition 1. 
A target $T$ can be defined as a tuple $(I, O, W)$ where:
- $I$ is the set of bound variables in the FOL clause.
- $O$ is the set of all unbound attributes marked as user intent in the FOL clause
- $W$ is the set of all distributed and disperse web services scattered around the web in different repositories where the framework agents have access to.

Domain Specific Agents

As the proposed framework does not push extra processing on the servers at the registration process time, the need to find suitable web services in the environment exists. Different web services have been located on different web service repositories based on the creators decision and are introduced through web service registries. In this model, the use of a single repository for web services has been avoided. By omitting the registration process, mobile agents should be used to discover new web services.

Our methodology formulates a very complex problem to solve for the designers of the framework but on the other hand having created this framework the least configuration for the WSC procedure would be needed. The problem has been formulated in Definitions 2 and 3.

Definition 2. 
Apt Rule Chain: Suppose there are $n$ predicates aggregately available in $m$ different rule bases. If at least one path exists from the initial state to the final conditions, we name the best chain of rules to satisfy the Target $T$ as an Apt Rule Chain (ARC).

Definition 3. 
Apt Rule Chain Process: The process of finding the most suitable chain of rules to satisfy the constraint or prove the theory is called the Apt Rule Chain Process (ARCP).

The web service composition process can suitably be mapped to the Apt Rule Chain Process where the rules in the ARCP are the web services in our environment. The web services provide an interface where the input will receive the user request and variables and the output will present the effect of WS functionality on the received values. A web service can in this regard be modeled as a rule in the rule base. The preconditions of the rule are mapped to the inputs of the web service, while the web service outputs would resemble the consequences of the rule (Figure 2).
provided. The exploiters of the framework can create models related to their own domain of interest. As we indicated before there are one abstract type of DSA for each concept root. For example if we are conceptualizing the Library domain, a DSA can be created for the Books, Personnel, and etc. Books themselves can have Authors, Content, and so forth. The main point is that only root concepts can have relating DSAs.

The population of domain specific agents is primarily user configured but is dynamically adapted based on the environment needs. Every DSA is responsible for finding and recording the web services related to its field of interest (the agents’ field of interest is supposed to be the same as the concepts in the category he belongs to.). The DSAs’ life cycle consists of three phases:

1. Birth
2. Lifetime
   - Search and Index Domain Specific Web Services
   - Have Regular Homo-Meetings
   - Have occasional Hetro-Meetings
3. Death

Figure 3. The Lifecycle of Each DSA

**DSA Lifecycle**

As it had been previously mentioned the lifecycle of DSA consists of three phases: Their birth that can be any time, lifetime, and finally death. Their death occurs to control the population of the agents that are active in the system. Besides the agents were initially created at the commencement of system activity, the rest of newly born agents are formed based on a novel agent reproduction system. We will explain a DSA’s behavior in detail in the following paragraphs.

DSA agents search for the web services that are related to the same context of their ontological background. For example if an agent belongs to the class of Book in the ontology he will be interested in web services that may have indexed books, sell or even rent electronic books, but on the other hand they have no interest in the web services that try to provide rental cars. As the DSA agents are mobile and move around the environment (based on a given algorithm) they tend to find new web services. Upon an arrival at a new web service they will map the syntactic attributes of the WC as shown in Figure3. The location of the encountered web service along with a timestamp is also stored. In this way the knowledge base of all agents is gradually filled but having a complete knowledge base is never guaranteed in this approach. Every DSA will have as much information related to its context as possible. As different DSAs of the same type traverse different paths they will eventually have different knowledge bases.

The other feature that the DSAs benefit from is the regular homo-DSA meetings. Every homo-DSA meeting is held after a cycles have passed. These homo-DSA meetings are held locally for homogeneous DSA agents and provide the means of rule base exchanges. Each type of DSA can have different simultaneous meetings for DSAs which are spread in different locations of the network.

Every DSA agent has a Meeting Radius (MR) which identifies the radius in which the attending agents will participate. For example if the MR is 5, then only the agents residing on the machines within this agents 5 hops will receive his message for participating in a meeting. Agents will based on the messages received decide on the meeting they would like to attend.

The local meeting will be held at the machine where the oldest DSA resides. During the meeting the rule bases will be exchanged and updated. If a web service has got down and many of the agents don’t know about it, they will update their rule base based on their peers rule base and its newer timestamp. The other point that the agents will decide upon in the homo-DSA meetings is the value of MR. If enough number of rule exchanges has been done, MR will remain unchanged, however if the number of exchanges is too low, MR will increase. The eventual value of MR will be based on the average MR value of all agents attending the meeting. The main benefits of the homo-DSA meetings are:

1) The agents will add new rules without having to traverse a specific machine which at times can be very far away.

2) Rule bases of all DSAs are frequently updated based on newer information.

The other main question is that what routes will the DSAs follow? The idea behind the domain specific agents movement is based on the swarm intelligence. The behavior of one kind of ants were observed and used to model the next hop selection strategy of the DSAs. The collaboration of fireants[13] in defending or even attacking an enemy was used to model DSAs movement behavior.

The sting of a single fireant is not nearly as painful as a single sting from a wasp or centipede. The pain and danger lies in the multiple stings delivered by a single ant and most important the fact that fireants rarely attack alone. Their powerful pheromones tell their colony members that help is needed. The real pain of fireants comes from the combination of hundreds of angry insects and each one may sting numerous times. In our algorithm DSAs follow similar behavior. Having moved on to
another machine the DSAs will send back reports of how useful this machine was. The usefulness of the machine is determined by the number of new web services that the DSA has found. As machines are not client machines and are actual servers on the web, this activity shows the servers activity in adding new services and can be used for future references, so as the DSA reaches a useful machine it sends back a message to the previous machine indicating a good choice for routing other DSAs of the same type.

One other type of agents existing on each machine is actually immobile. They form a routing table for each specific type of DSA and manage all the messages that have been sent from the DSAs. As the importance of the sent back messages should wear off after some time, these agents will decrease the effect of an old message on a specific route by decrementing its value. So as time goes by the effect of older messages is decreased and new messages have greater effect. By this mechanism if a server had been previously inactive but has started extensive operation now will have the chance to survive and receive DSA agents that will explore it.

The other technique that is utilized to increase path selection diversity is the use of a probabilistic path selection. In this method every path will be selected by a degree which will be calculated based on the routing tables. So even if DSAs have a low interest in navigating a path that path still has a low chance of being selected. This mechanism was to a great extent inspired by the roulette wheel technique in genetic algorithms.

D. Web Service Composition Plan

After the user submits his request to the system, it will be analyzed and the intent (user inputs and desired outputs) are detected and are shaped in first order logic clauses. The FOL clause is then passed to the Web Service Composer (WSComposer) component which will handle the case and provide the ARC. The first step in this module is that the final desired outputs are parameterized. For example if a user has requested a red car, the WSComposer will change the request into the ARC concept based on the available ontology. The abstract FOL clause is then sent to the Planner to create the desired ARC.

Planning Through DSAs' Meetings

Once the abstract FOL clause has been received by the planner it will start to explore the possible solution space to find the most optimal solution; however it does not guarantee to find the best solution. The planner will call one of the DAS agents that are related in concept with the abstract FOL provided from the previous module. If there is more than one concept that can be mapped to the requested service, one DSA agent will be created for each of those concepts. The summoned DSA agents will then form a hetero-DSA meeting.

The hetero-DSA meeting will comprise all of the DSA agents that conceptually have some sort of relation with inputs or outputs of the request. The WSComposer will start the rule chaining process based on the rule bases of all the present DSAs which will be aggregated to form one unique rule base. The inference process is followed from top to bottom and vice versa, by this we mean that the planner tries to reach a plan both starting from the outputs to reach the inputs. While on the other hand the inputs are thought as initial conditions and the chaining process tries to gain the desired outputs. Both of the techniques which respectively are called backward and forward chaining are utilized. In other words, backward chaining starts with a list of goals and works backwards to find out if there are suitable inputs available to support the desired goals or not but forward chaining will do the reverse.

After the chaining process two probable states will occur. The first state is when the WSComposer has been successful in finding an ARC. In this situation the ARCs – multiple ARCs can be calculated when there is more than one chain which will reach the goal, we will talk more on this in the future works section – are sent to the next module to be checked for parameter value consistency. However if the WSComposer is absolutely futile in returning a suitable result the Hint module is consulted.

In this phase the list of all unbound variables are listed and a request is sent to the user. The user will then have the choice to fill in some the required variables in order to provide the planner with more choices for creating the chain. If the user refuses to provide more information or in any case does not have more information to offer the framework will return a list of partial solutions that it was able to find. Although the partial solutions may be off track, but they will benefit the user in two ways: 1. The user will learn more on how the inference procedure of the system is so the next time he will be more precise in defining his requirements and 2. The partial solution may help the user gain some information, although incomplete, about the path to getting to the information he needs.

On the other hand if the user provides the framework with the required information or even part of the required information the chaining process will again start. This time the old rule base is still consulted however the DSA agents which where used the previous time have surely left this server, so new DSAs are called for. This change of agents will increase the possibility that some key missing rules will be added and allow the chain to complete or at least allow more rules to be added to the rule base outdated the previous old rules and resulting in an updated rule base which will ease the inference process.

If the WSComposer is victorious in providing the user with the required ARC, two subsequent actions will be triggered. The first action is that the list of successful
chains is stored in a Proof Dictionary for later use. The next time a similar request is made the WSComposer will automatically check the proof dictionary for available entries that might fit the problem on hand. The proof dictionary on the other hand has a specific timeout value for every entry. Whenever a timeout occurs the specific record will be deleted. The reason that the records are regularly deleted is that new web services might be brought into function or old services might have gone down.

The second action that is automatically triggered is the birth of new DSA agents. If the chaining process is successful, new agents of the type of the DSAs involved in the chaining process are born. This is like the reproduction in real world in the way that if fertility is achieved new children will be born. On the other hand, as the DSAs get older the probability of their death increases. Being involved in success or failures in chaining will increase or decrease the chance of survival. Having a heavy rule base will also decrease DSA mortality.

E. Parameter Range Control

Having created the abstract ARC, the WSComposer will send the abstract ACR to the parameter range control module. The parameter range control component checks to see although the requests and inputs conceptually match the deduced chain, if there parameter values are in the scope of the discovered web services or not. Suppose some one is looking for web service to buy a red car in London, and the WSComposer provides him with a web services that manages a car seller but that car seller is located in Los Angeles. So the Parameter range control component is there to check for any inconsistencies. The set of ARCs that have satisfied the control of this stage will then be ready made for the end-user to select from. The user will eventually have the choice to select from the list of web service compositions provided by the framework.

3. Conclusion

In this paper we have tried to provide the basis of a complete framework for web service composition. We try to provide the means of easier and simpler user interaction through the usage of a restricted English language to enable the user to communicate with the framework in a much easier way. The user input is then transformed into first order logic to allow rule chaining using both forward and backward chaining. User intent and the sketch of the outside world is made through bound and unbound variables. Features like using hints to unlock a deadend in the chaining process and the creation of a proof dictionary to speed up the answering process have also been introduced in this framework.

An agent society has also been introduced in the system where the agents have a lifecycle and a reproduction system. Two types of different agents have been created. The domain specific agents are mobile agents which allow the web service discovery and definition and the other type of agents are used to guide the DSAs while routing in the network. The routing process was to a great extent inspired by the fire ants’ defence system.

4. References