A Hybrid Framework for Automated Onto-Logical Web Service Composition based on Agent Societies

Faezeh Ensan, Mohsen Kahani, Ebrahim Bagheri†

Department of Computing, Ferdowsi University of Mashhad, Iran
†*University of New Brunswick, Department of Computer Science*

Abstract.
The world has experienced a great evolution through the information technology era. Finding suitable vital industrial resources may have been the aim of the previous decades; however these goals have now changed. The 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.

Introduction

The World Wide Web has been experiencing great evolution since its early date of birth in the 90's. The number of users which have created an immense internet traffic based on the usage of the content provided through the web servers is quite astonishing. The number of web sites hosted around the world shows great enthusiasm in using the potential abilities of the world changing technology.
Although the early days of the internet were the adaptation days which required more cultural adaptation and social acceptance; it is now the time for productivity and hyper breeding. Based on the four layers of web usage, many organizations and governments are providing their information and in many cases, their services to their customers or their citizens in an electronic manner. This trend inspires many researchers to build upon the current structures and expand web's ability. However, statistics show surprising details on the decline in the growth of the number of web servers being set up throughout the world. These figures depict the fact that the number of web servers world wide that was doubling every 53 days in 1995 has had a decrease to 173 days in 1997. Many different reasons can be imagined for the situation.
Having an unstructured growth of the current web can be assumed as 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 e.g. RELAX [1], DTD [2], TREX [3] 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. Different systems such as the DELTA [4] were designed to allow the usage of toxonomic description in computer processing.
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 [5], OIL [6], DAML [7], and DAML+OIL [8] 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 [9] 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. Server-side scripts were usually written in some sort of programming language such as PHP, Perl and etc leaving the clients access to sole web browser interface access. 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, Mirosoft .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. The main purpose of this model is to simulate the real world by its real role-players in that the complexity is added to final outcome eliminating the burden in initial design intricacies. Models for building a resource discovery algorithm have been proposed in the text like RADMA [10]. These algorithms use mobile agents to identify suitable dispersed resources for allocation to available tasks in a pervasive computing environment. On the other hand agents can be used to monitor user behavior for user modeling purposes [11]. Agents have varied application in which many researchers have had different contributions. An entity should have different features to satisfy the agent definition which are autonomy, intelligence, flexibility, and rationality [12]. 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 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 framework identifies possible solutions to the request based on an automated web services composition in a distributed web service registry environment. The web services are registered at dispersed service banks or the so called UDDI repositories which are spread world wide. The WSDL definition of the web services which mainly consists of protocol bindings, message formats, and input types (the web service syntax) are converted to first order logic. The web service composer (WSComposer) module with the help of the Domain Specific Agents (DSA) tries to produce the best solution to the problem by applying forward and backword chaining rules. Partial solutions are given to the user if the actual request could not be fully answered. The user will then provide hints to help the WSComposer fulfill the actual need. The final solution is then checked for ontological and boundary consistency. The approved solutions will be handed to the user.

The paper is composed of seven sections. The next section will give a brief summary on the available solutions to web service composition. Section 3 will detail the methodology used in the framework creation and explains the details of the design. Section four will then cover a sample scenario followed by some ideas for future work. Section five concludes the paper and summaries the main points.

2. Web Service Composition Models

Web services are logic packages that provide functionality to the user based on his request. The creation of web services has removed the burden of redesigning a lot of code and functionality. Each user can select a specific web service according to his needs and based on the services' description. Although having many different web services around in scattered repositories allows users to search for the appropriate services; but on the other hand ensuring the user that he will find the finest web service for the task or an optimal sequence of web services for a job is not possible. For this reason models such as the previously mentioned model for resource discovery [13] (taking in mind that web services are also resources which are available in the web) have been created to find the appropriate services. Many different algorithms have also been devised for web service compositioning that can mainly be categorized in two main classes: Work-flow based models and the Logic based algorithms.

The workflow based models assume that the web services are known to the user and it is the omniscient's chore to find the users needs and create apt composite web service templates so that the users exploit them as the need arises. The templates are only abstract models of the available web services, so a search to find a real web service would be needed to create an instantiated model. The second model of web services is based on reasoning on the available web service functionalities. Most of the models translate the web service definition to a first order logic language and use the reasoning features of that language to create a solution. To our belief these models have a main deficiency which is the lack of consideration for the fact that the web services are distributed among different web service registration repositories and ordinary inference models are not suited for these cases. In this section we give a brief introduction to the 3 models: BPEL4WS [14], SWORD [15], and SHOP2 [16].

2.1.BPEL4WS

BPEL4WS is an XML-based language designed to enable task-sharing for a distributed computing environment through the combination of multiple web services. The model was first proposed by BEA Systems, IBM, and Microsoft. BPEL4WS has a structural building that can support various functionalities which is shown in figure1.

<table>
<tr><td rowspan="5">Constraints (States)</td><td colspan="2">Scopes</td><td>Partners</td></tr>
<tr><td colspan="2">Activities</td><td>Service Links</td></tr>
<tr><td>Basic</td><td>Structured</td><td>Reference</td></tr>
<tr><td colspan="3">Message Correlation</td></tr>
<tr><td colspan="3">Message Properties</td></tr>
</table>

Figure1. The Structure of BPEL4WS

The BPEL4WS provides enough technical functionality for the users to design a complex process that will be executed across the web in such a way that any cooperating entity can perform one or more steps in the process. BPEL4WS can be used to describe a business protocol that formalizes what pieces of information a specific procedure consists of, and what exceptions may have to be handled. However the BPEL program itself, does not involve the internal process of web service execution.

2.2. SWORD

SWORD is another rule based framework for creating web service compositions designed at the Stanford University which does not use the standard service descriptions such as WSDL, SOAP, RDF and DAML. The services are modeled by their pre and post conditions and are exploited only if the preconditions are met. In this way by defining the initial state and the user intended goals, the system will create a suitable plan. An execution model has also been provided with SWORD that allows generated plan instantiation and execution. One of the main features that the developers of SWORD point out is that it can be integrated with currently available web services.

2.3. SHOP2

SHOP2 is a hierarchical task network planner which creates which creates plans based on task decomposition. The HTN procedure iteratively decomposes the bigger tasks until undividable (atomic) tasks have been created. It will then generate a sequence of the atomic services as the plan. A planning problem in SHOP2 can be defined as a triple (S, T, D) where S is the initial state, T is a goal task list, and D is a domain description. SHOP2 will return a plan that will result in the goals starting from the initial state in the specific defined domain. The proposed model provides a sound and complete algorithm to convert OWL-S descriptions to the SHOP2 domain. OWL-S was initially created to support the automatic web service discovery, composition, execution, and monitoring.

3. Proposed Framework

As could clearly be seen from the short survey given on the main systems available in the field of automated web service composition, although the research ground is a fresh one, but there are a wealth of available techniques, frameworks, tools, and algorithms available. 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 other main point that was considered in this framework was 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 our 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 to 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 2.

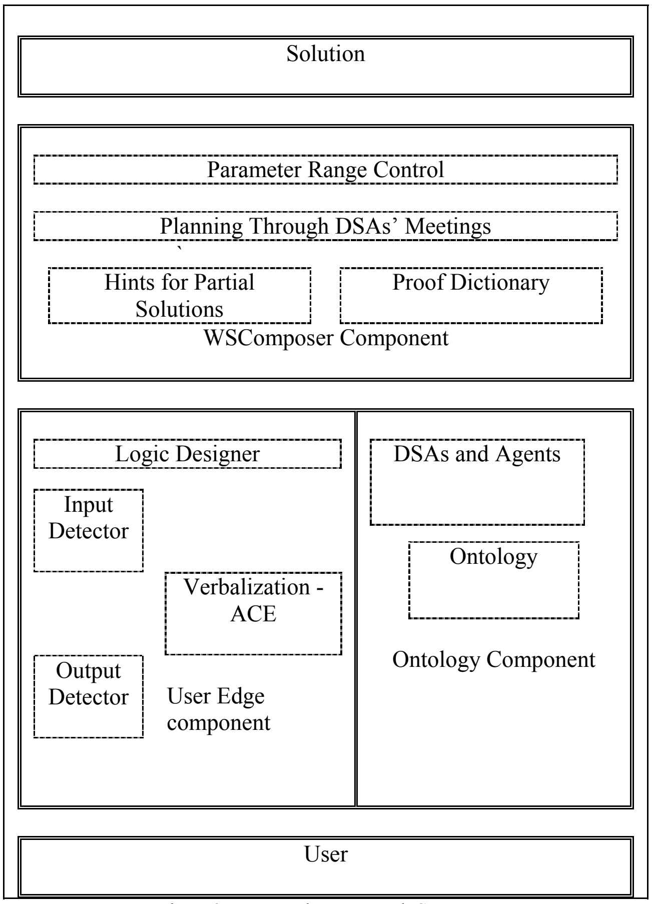


Figure2. Proposed Framework Structure

One the main phases of our web service composition framework is the functionality that provides the mechanisms that adapt the newly registered web services into the environment. This process is referred to as Web Service Registration (WSR).

3.1. 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 [17]. 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.

3.2. 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 built-in user interaction handler is to bring better understanding of the user intent based on iterative queries to the WSC framework. In this way the intent of the user is extracted form his/her request based on the features that are required for the algorithms to find the best chain of web services for composition.

A. User Interface

The use interface is primarily responsible for interacting with the user through different sorts of interfaces. The basic interface provided to receive information from the user can be thought of as a GUI interface that can be implemented using a web or windows based application. Although this approach seems to be sufficient but another layer of abstraction has also been envisaged for better interoperability. The system capability can be provided to business firms as a web service it self. In this way, legacy applications or currently under development applications can easily interact with this framework to find their required operations based on the automatic web service framework; hence promoting interoperability.

B. Verbalization - ACE

The process of verbalization is placed in the framework to allow the users to interact with framework through their usual grammatical structures. The framework does not enforce the users to use a specific form of request coding which is used in many systems. For example sending a request to a search engine involves creating a query sentence which is made up of + for inclusion or - for exclusion of a certain term in the search. These two operators do not make the state that complicated; however a user requiring more specific search will need to learn more details of the search engine specific language that might include peculiar formats. This problem complicates the user-system interaction process and at times may dissatisfy many users. The ACE language (Attempto Controlled English) was therefore chosen as the basis of the verbalization process [18].

ACE is a natural language which is a controlled subset of the English language grammar that can be unambiguously translated into the first order logic. An ACE written sentence is translated into a discourse representation structure (DRS) using the Attempto Parsing Engine (APE). DRS can then be turned into pure FOL through slight modifications [19].

For example having entered the sentence:

```
John wants a car. The car is green.
```

Into the ACE, it parses the two sentences trying to find missing verbs or nouns. If no such missing word is found it then classifies the objects into variables and produces an output that is the paraphrased version of the original sentence:

```
John wants a car C. The car C is green.
```

C has been defined as a variable to identify the car that John wants. The paraphrased sentence is then translated into the DSR format which is to a great extent close to the First Order Logic. The Discourse Representation Structure created for the previously mentioned sentence would be as:

```
[A, B, C, D, E, F, G]
Named (A, John)-1
Structure (A, atomic)-1
Quantity (A, cardinality,
  count_unit, B, eq, 1)-1
Object (A, named_entity, person)-1
Object(C, car, object)-1
Quantity(C, cardinality, count_unit,
  D, eq, 1)-1
Structure(C, atomic)-1
Predicate (E, unspecified, want, A,
  C)-1
Property (F, green)-2
Predicate (G, state, be, C, F)-2
```

The provided syntax can easily be transformed into the first order logic. A sample first order representation of a more complex sentence that will be further used in the paper to provide different examples has been shown later in the paper. The ACE languages does not have a dictionary of words as a built in function so a dictionary should be created as an add-on by the systems that benefit from it, or the type of words used in the sentence structure should be specified when entering the sentence. For example as the parser does not recognize London as a proper name the letter 'p' has been placed behind it and an 'a' before the word blue specifies that blue is an adjective.

Original Sentence:

```
John wants a CAR. The car is green.
John lives in a CITY. The city's name
is p:London.
```

Paraphrased Sentence:

```
John wants a CAR E. the car E is green.
John lives in a CITY J. A name N of the
city J is London.
```

C. Logic Design and Intent Extraction

Having formulated the DRS form of the user input sentence, we will then calculate a first order logic based form. One the main reasons of choosing the first order logic as the final representation of the internal framework interaction was that it allows easy inference based on current diverse methods like the forward or backward chaining algorithms. On the other hand the first order logic predicates allow us to easily extract the unbound variables and classify them as the user intent/request. The bound parameters along with their associated values would then be imagined as the inputs to the system. The user intention is then defined as: 1. an initial state which is formed by the inputs, 2. a final state which is the output or the unbound variables. The model is then defined as follows:

Definition1. *Web Service Composition Target*: 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.

To show a sample output of this stage we illustrate the first order logic that can be inferred from the logic design module; however the format is simplified for the sake of clarity and readability and to allow more understandable reasoning in the future parts of the paper. Suppose that we are forming the target T for the previously introduced sentence. The words written in capital letter are related to an ontology will be explained later in the paper.

Input Set (Axioms):

```
Man (John).
Is (CAR, E).
Color (E, Green).
Is (CITY, J).
Name (J, London).
¬Have (John, E).
```

Output Set (intent):

```
?- Have (John, E).
```

Rule Set:

```
All the distributed web services which
are accessible by the implemented
framework form the rule set.
```

D. Agent Groups and Their Behavior

There are typically two types of agents incorporated into the structure of the framework. The first group of agents which consists of the so called Domain specific agents (DSA) is designed to be of mobile in nature, multi purpose and perform different tasks. The second type of agents is fixed and helps the Domain specific agents find their best route based on the previous inferences which form the positive or negative experiences. The DSA movement is simulated based upon the behavior of a particular insect which will be explained thoroughly in its part. Our aim in using this type of routing is to provide indirect agent collaboration through the changes they make on the outside world; hence different means of direct communication or message passing between agents is not required. The two types of agents used in our framework are located in the ontology component of the framework allowing them to search for the required web services based on an ontological perspective.

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 our model we aim to avoid the use of a single repository for web services. The simplest approach would be to create a central registry for the web services in the way that the owner of the web service would have to register his web service in our central repository in order to allow its web service use in the web service composition process. To omit the registration process mobile agents could be used to discover new web services. Although this approach will ease the chaining process in the way that the inference process should be done on a single rule base but it will create a bottleneck in the system. Failure in the central repository will result in a disturbance in the total system functionality.
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.

Definition2. *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).*

Definition3. *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 3).

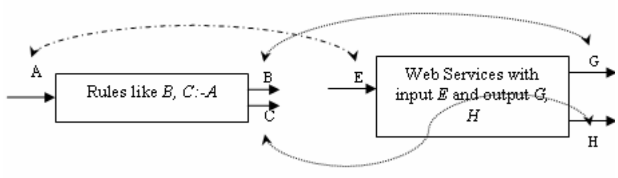


Figure3. Mapping a Web Service to a First Order Logic Rule

The framework uses an ontology based component that maps different concepts under different categories. For each of these categories, there exists a tree like topology in which various concepts can sit. For each of the main concepts in the root, that can vary depending on the type of domain that has been modeled, on abstract type of DSA is created. This framework provides the means to create different concepts in the ontology component but no specific conceptualization for a specific has been 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

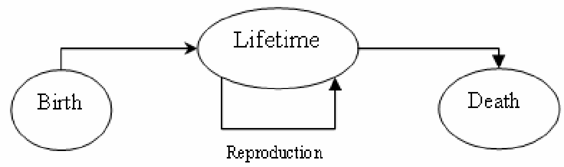


Figure4. The Lifecycle of Each DSA

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 α 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 (as was explained before, there may be different homo-DSA meetings for the same ontological concept at the same time) 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 gone 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[20] 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.

E. 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 Compser (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 *hetro-DSA meeting*.

The hetro-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 outdating 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.

F. 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 deducted 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.

4. Case Study: Trip Planner

In this section we will provide a case study on the proposed framework to clarify all of its internal functionality. The case study will encompass different parts of the proposed algorithms to show different work flows. A sample flow chart of a regular framework activity has also been provided in Figure 5.

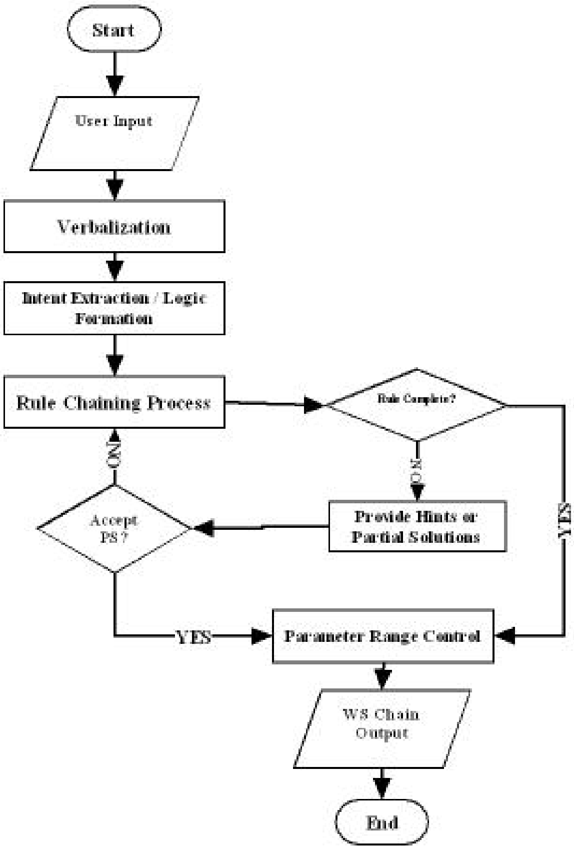


Figure5. Flowchart of the Framework Activity

Suppose that a user wants to submit the following request to the framework either through the provided GUI or the web service. Today is Saturday and John is currently in London. He wants to be in Amsterdam on Tuesday. So he needs to schedule a taxi from his house to the airport, an airplane from London to Amsterdam. The scenario has been kept to a very simple one to avoid confusion during different steps. So he gives such a request to the framework:

```
p:Saturday is a day.
p: Tuesday is a day.
p:Today is a:Saturday.
John travels by an n:airplane.
p:London is a city.
p:John is  in p:London.
p:John is at n:home.
p: Amsterdam is a city.
p:John goes to p: Amsterdam a:tueday.
```

As it can be clearly seen the syntax has far from been true human language, but it obviously alleviates the problems which where previously stated. The system will then paraphrase the sentences into:

```
Saturday is a DAY M.
Tuesday is a DAY M.
Today is Saturday.
John J travels by an airplane U.
London L is a CITY C.
John J Lives in London L.
Home H is a PLACE P.
John J is at home H.
Amsterdam A is a city E.
John J goes Tuesday to Amsterdam A.
```

The paraphrased sentence would then be used to create the first order logic clauses. As the machine created FOL clauses are two complicated to be demonstrated a FOL clause for a simple sentence like the sentence used in the verbalization – ACE part would be:

exists(A, exists(B, exists(C, exists(D, exists(E, exists(F, exists(G, (named(A, John)-1) ∧ (structure(A, atomic)-1) ∧ (quantity(A, cardinality, count_unit, B, eq, 1)-1) ∧ (object(A, named_entity, person)-1) ∧ (object(C, car, object)-1) ∧ (quantity(C, cardinality, count_unit, D, eq, 1)-1) ∧ (structure(C, atomic)-1) ∧ (predicate(E, unspecified, want, A, C)-1) ∧ (property(F, green)-2∧predicate(G, state, be, C, F)-2)))))))

A more comprehendible format will be given here for the sake of clarity.

Input Set (Axioms):

```
Day (Saturday, D1).
Day (Tuesday, D2).
Now (D1).
Man (John, J).
Airplane (U).
Travels (J, U).
Is (CITY, L).
Name (L, London).
Lives (J, L).
Place (Home, H).
At (J, H).
Is (CITY, A1).
```

```
Name (A, Amsterdam).
Go (L, A, D2).
```

Output Set (intent):

```
?- At (J, A), Now (D2).
```

Rule Set:

All the distributed web services which are accessible by the implemented framework form the rule set described in the following lines.

Suppose we only have two web service registries named *WSR1* and *WSR2*. The following web services reside on each of the web service registries respectively (Table1).

Table1. List of Web Services Registered at each WSR

Web Service Registry 1		Web Service Registry 2	
Name	Description	Name	Description
Airport Taxi	Input : Home Output: Airport	Air Transfer	Input: Airport Output: Airport
Money Exchange	Input: Dollars Output: Euros	News Server	Input: Credit Card Info Output: RSS based News
Bus Transfer	Input: Bus Terminal Output: Bus Terminal	Our Framework!	Input: String Request Output: Chains of WSC solutions

Having created an environment with six web services we can also devise the appropriate ontology. It is obvious that the detail of the ontology depends on the describers attitude towards modeling the environment. We can create very complex definitions for the six web services but every thing has been simplified. So we initially create four concepts: Transfer, Banking, News, WSC. For this reason four types of DSA agents will be created. The number of agents related to each type is arbitaray and is based on the framework configurations. We assume that 4 DSA agents will be created for each WSR. So in the first phase (before the first DSA host transfer) the knowledge base of the agents will look like Table2.

Table2. DSA Knowledge Base

Web Service Registry 1			Web Service Registry 2		
Name of DSA	Rule base	TS	Name	Description	TS*
Transfer	Airport Taxi:: Home->Airport	1	Transfer	Air Transfer:: Airport->Airport	1
	Bus Transfer:: Bus Terminal-> Bus Terminal	1			
Banking	Money Exchange:: Dollars->Euros	1	Banking	-	-
News	-	-	News	News Server:: Credit Card Info-> RSS based News	1
WSC	-	-	WSC	WSC:: String Request-> Chains of WSC solutions	1

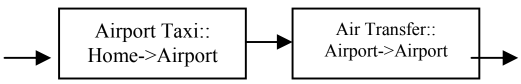
* TS is Timestamp

The request in the form of FOL clauses will be directed to the WSC web sevice located on the WSR2. the web service starts the chaining process in both a backward and forward way. As there are no useful webservices to match the inputs of the request the forward chaining process will be a total failure. The backward chaining suggests a user takes the Transfer webs service to buy a ticket from London to Amsterdam but the user has to deal with the rest of the path. The framework will also provide some questions (based on the hint module) to the user like "*Please provide: Credit Card Info",* to see if it could chain in the News Server web service as useful or not, if the user either refuses to provide credit card information or accept the solution, the framework will wait until the new DSA agents have moved into the repository. The formed table inside the planner will then look like Table3.

Web Service Registry 1			Web Service Registry 2		
Name of DSA	Rule base	TS	Name	Description	TS
Transfer	Air Transfer:: Airport->Airport	2	Transfer	Airport Taxi:: Home->Airport	2
	Airport Taxi:: Home->Airport	1		Bus Transfer:: Bus Terminal->Bus Terminal	2
	Bus Transfer:: Bus Terminal-> Bus Terminal	1		Air Transfer:: Airport->Airport	1
Banking	Money Exchange:: Dollars->Euros	1	Banking	Money Exchange:: Dollars->Euro	2
News	News Server:: Credit Card Info-> RSS based News	2	News	News Server:: Credit Card Info-> RSS based News	1
WSC	WSC:: String Request-> Chains of WSC solutions	2	WSC	WSC:: String Request-> Chains of WSC solutions	1

Table3. DSA Knowledge Base in the Second Time Phase

Having built a complex table based on the new information the planner can now propose a complete chain to the user:



The chain added to the proof dictionary while it is also passed to the parameter range control module. The framework functionality is much more complex than what was depicted in the scenario, but simplifications were made to bring about better understanding of system dynamics.

5. Future Work

There are a wealth of work been done the field of adaptive systems which try to customize the user interaction with computer based on hbis interests, previous actions, goals, or his level of knowledge[]. We will further investigate how the two fields will converge either by implementing the adaptive features into the web service composition framework to enable both the system and end user to make right decisions. Adaptiveness will allow the framework to alleviate its performance through the reduction of hint requests and better rule selection. On the other hand, benefiting from the web service composition techniques will allow adaptive systems to provide new functionality to end user that might not have been available in the past. With this insight we anticipate a lot of close research in both fields in order to make novel models and create new opportunities.

6. 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.

7. References

1. Relax NG Technical Committee: RELAX NG home page.Online in Internet, 2005
2. B. Ludaescher, Y. Papakonstantinou, P. Velikhov, and V. Vianu. View definition and DTD inference for xml. In Workshop on Semistructured Data and Nonstandard Data Formats, January 1999
3. TREX: Tree Regular Expressions for XML, J Clark - URL http://www. thaiopensource. com/trex, 2001
4. G. Cobena, S. Abiteboul, and A. Marian. Detecting changes in xml documents. In Proc. Int. Conf. on Data Engineering, 2002.
5. Michael K. Smith, Chris Welty, and Deborah McGuinness, Owl web ontology anguage guide. http://www.w3.org/TR/owl-guide/, 2003.
6. D. Fensel, F. van Harmelen, I. Horrocks, D. L. McGuinness, and P. F. Patel-Schneider. OIL: An ontology infrastructure for the semantic web. IEEE Intelligent Systems, 16(2):38-45, 2001.
7. J. Hendler and D. McGuinness, "The DARPA Agent Markup Language," IEEE Intelligent Systems, vol. 15, no. 6,Nov./Dec. 2000, pp. 72–73.
8. I. Horrocks. DAML+OIL: A Description Logic for the Semantic Web. IEEE Data Engineering Bulletin, 25(1):4–9, 2002.
9. P. Traverso and M. Pistore. Automated Composition of Semantic Web Services into Executable Processes. In Proc. of ISWC 2004.
10. Ebrahim Bagheri, Mahmood Naghibzadeh, Mohsen Kahani: A Novel Resource Dissemination and Discovery Model for Pervasive Environments Using Mobile Agents. HPCC 2005: 1043-1048.
11. Felix Hernandez-del-Olmo, Elena Gaudioso, Jesus Boticario: A Multiagent Approach to Obtain Open and Flexible User Models in Adaptive Learning Communities. User Modeling 2003: 203-207.
12. Franklin, S and Graesser, A, 1997, ``Is it an agent, or just a program?: A taxonomy for autonomous agents" In: MuÈ ller, JP, Wooldridge, MJ and Jennings, NR, eds, Intelligent Agents III: Proceedings of the Third International Workshop on Agent Theories, Architectures and Languages (Lecture Notes in Artifcial Intelligence, 1193) Springer-Verlag, 21-35.
13. Ebrahim Bagheri, Mahmoud Naghibzadeh, Mohsen Kahani, Faezeh Ensan, A Novel Resource Advertisement and Discovery Model for Ubiquitous Computing Environments using Mobile Agents, Proceedings of the 10^{th} IEEE Tencon-2005, Melbourne, Australia, November 21-24,2005.
14. Mandell, D.J. and McIlraith, S.A. Adapting BPEL4WS for the Semantic Web: The Bottom-Up Approach to Web Service Interoperation. In Proceedings of the Second International Semantic Web Conference, 227-241, 2003.
15. S. R. Ponnekanti and A. Fox. SWORD: A Developer Toolkit for Web Service Composition. In Proc. of the Eleventh International World Wide Web Conference, Honolulu, HI, 2002.
16. D. Nau, T. Au, O. Ilghami, U. Kuter, J. Murdock, D.Wu, F. Yaman, SHOP2: An HTN planning system, Journal of Artificial Intelligence Research 20 (2003) 379–404.
17. B. Medjahed, A. Bouguettaya, and A. K. Elmagarmid. Composing Web Services on the Semantic Web. The VLDB Journal, 12(4):333–351, 2003.
18. Fuchs, Norbert E., Uta Schwertel, Rolf Schwitter (1999) Attempto Controlled English (ACE), Language Manual, Version 3.0, Technical Report ifi-99.03, University of Zurich.
19. Stefan Hoeer. The Syntax of Attempto Controlled English: An Abstract Grammar for ACE 4.0. Technical Report I-2004.03, Department of Informatics, University of Zurich, Zurich, Switzerland, 2004.
20. The Red Imported Fire Ant, http://www.antcolony.org/fire_ants.htm, 2002