# An MAS Infrastructure for Implementing SWSA based Semantic Services

Önder Gürcan[1], Geylani Kardas[2], Özgür Gümüs[1], Erdem Eser Ekinci[1], and Oguz Dikenelli[1]

[1]Ege University, Department of Computer Engineering, 35100 Bornova, Izmir, Turkey
{onder.gurcan,ozgur.gumus,oguz.dikenelli}@ege.edu.tr
erdemeserekinci@gmail.com
[2]Ege University, International Computer Institute, 35100 Bornova,Izmir
geylani.kardas@ege.edu.tr

**Abstract.** The Semantic Web Services Initiative Architecture (SWSA) describes the overall process of semantic service execution in three phases: discovery, engagement and enactment. To accomplish the specified requirements of these phases, it defines a conceptual model which is based on semantic service agents that provide and consume semantic web services and includes architectural and protocol abstractions. In this paper, an MAS infrastructure is defined which fulfills fundamental requirements of SWSA's conceptual model including all its sub-processes. Based on this infrastructure, requirements of a planner module is identified and has been implemented. The developed planner has the capability of executing plans consisting of special tasks for semantic service agents in a way that is described in SWSA. These special tasks are predefined to accomplish the requirements of SWSA's sub-processes and they can be reused in real plans of semantic service agents both as is and as specialized according to domain requirements.

## 1 Introduction

The Semantic Web Services Initiative Architecture (SWSA) committee, which has been contributed by the Semantic Markup for Services (OWL-S)[1], Web Service Modeling Ontology (WSMO)[2] and Managing End-to-End Operations-Semantics (METEOR-S)[3] working groups, has created a set of architectural and protocol abstractions that serve as a foundation for Semantic Web service

[1] The OWL Services Coalition: Semantic Markup for Web Services (OWL-S), 2004, http://www.daml.org/services/owl-s/1.1/
[2] Web Service Modeling Ontology (WSMO) Working Group, http://www.wsmo.org/
[3] Managing End-to-End Operations-Semantics (METEOR-S) Working Group, http://lsdis.cs.uga.edu/projects/meteor-s/

technologies[1]. The proposed SWSA framework builds on the W3C Web Services Architecture working group recommendation[4] and attempts to address all requirements of semantic service agents: dynamic service discovery, service engagement, service process enactment and management, community support services, and quality of service (QoS). This architecture is based on the multi-agent system (MAS) infrastructure because the specified requirements can be accomplished with asynchronous interactions based on predefined protocols and using goal oriented software agents.

The SWSA framework describes the overall process of discovering and interacting with a Semantic Web service in three consecutive phases: (1) candidate service discovery, (2) service engagement, (3) service enactment. The SWSA framework also determines the actors of each phase, functional requirements of each phase and the required architectural elements to accomplish these requirements in terms of abstract protocols. Although it defines a detailed conceptual model based on MAS infrastructure and semantic web standards, it does not define the software architecture to realize this conceptual model and does not include the theoretical and implementation details of the required software architecture.

There have been a few partial implementations to integrate web services and FIPA compliant agent platforms. WSDL2Jade [2] can generate agent ontologies and agent codes from a WSDL input file to create a wrapper agent that can use external web services. WSDL2Agent [3] describes an agent based method for migrating web services to the semantic web service environment by deriving the skeletons of the elements of the Web Service Modeling Framework (WSMF) [4] from a WSDL input file with human interaction. WSIG (Web Services Integration Gateway) [5] supports bi-directional integration of web services and Jade agents. WS2JADE [6] allows deployment of web services as Jade agents' services at run time to make web services visible to FIPA-compliant agents through proxy agents. But these tools only deal with the integration of agents and external web services and do not provide any mechanism to realize the entire architectural and protocol abstractions addressed by the SWSA framework. It's clear that there must be environments which will simplify the development of SWSA based software systems for ordinary developers.

The main contribution of this paper is to define a software platform which fulfills fundamental requirements of SWSA's conceptual model including all its sub-processes. Then, these sub-processes are modeled as reusable plans for development of semantic service agents. And the second contribution of this paper is to develop a planner that has the capability of executing these kinds of plans. So, the developed planner has the innovative features listed below:

- Definition of reusable template plans that includes abstract task structures for SWSA's sub-processes and usage of these templates for generating agents' real plans by specializing these abstract tasks

[4] W3C Web Services Architecture Working Group, Web Services Architecture Recommendation, 11 February 2004, http://www.w3.org/TR/ws-arch/

– Support for recursion on the plan structure
– Constitution of composite services by using reusable semantic service agent plans

The paper is organized as follows: in section 2, the proposed architecture of the Software Platform for the SWSA framework is given. Planning requirements of SWSA is discussed in section 3. Section 4 introduces the planner component of the platform. Conclusion and future work are given in section 5.

## 2 Semantic Service Platform Architecture

It is apparent that SWSA describes the architecture extensively in a conceptual base. However it doesn't define required details and theoretical infrastructure to realize the architecture. Hence, we propose a new software platform in which above mentioned fundamental requirements of all SWSA's sub-processes (service discovery, engagement and enactment) are concretely fulfilled.

The software architecture of the proposed Semantic Service Platform is given in Figure 1. The platform is composed of two main modules called *Semantic Service Kernel* and *External Service Agent*.

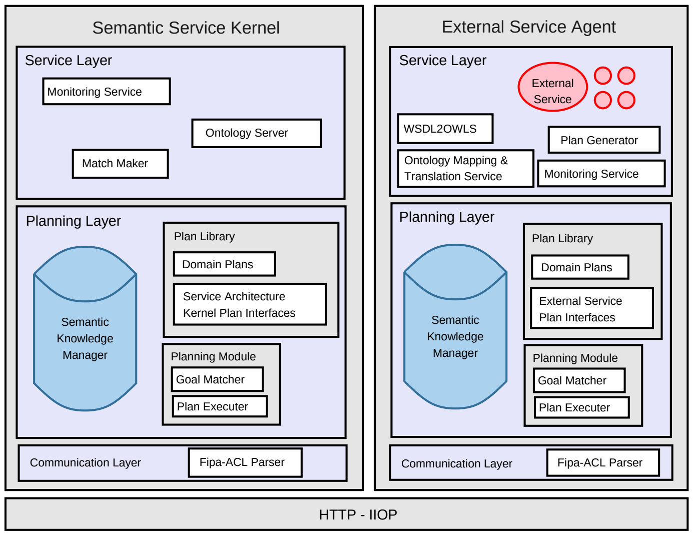

**Fig. 1.** The software architecture of the Semantic Service Platform: Two main models are Semantic Service Kernel and External Service Agent. The kernel provides architectural components for an agent to execute sub-processes of SWSA. On the other hand, External Service Agent provides integration of external services (either purely defined in WSDL or OWL-S) into the agent platform. Communication takes place via the well-known HTTP - IIOP (Internet Inter-ORB Protocol).

The Semantic Service Kernel includes the required infrastructure and architectural components for an agent to execute sub-processes of SWSA. The agent's actions, to be used in semantic service discovery, engagement and enactment, are modeled as reusable plans and will be executed in a composite fashion by a planner. The sub-tasks, which compose the plan, will execute SWSA's sub-processes by invoking the related services. Invocation will be realized via predefined execution protocols.

The External Service Agent converts either WSDL or OWL-S defined external services into agents that are able to execute SWSA's defined processes. This module includes inner services like WSDL to OWL-S Converter, Ontology Mapper and Translator -that provides mapping of services into the platform's ontologies, stores those mapping ontologies and serves ontology translation- and Monitor service that monitors quality of service parameters. Planner component of the External Service Agent realizes registration of the related service into the platform and executes interaction plans concerning service engagement and enactment. Those plans are formed automatically during the creation phase of the External Service Agent and stored in the plan library as domain plans.

In the following subsections, we discuss details of how our proposed semantic service platform meets the requirements of the SWSA taking into consideration predefined SWSA sub-processes.

### 2.1 Realization of service discovery process

In order to realize semantic service discovery, the platform services should be registered to a matchmaker and service clients should query on this matchmaker and have ability to interpret resultant service advertisements.

The Matchmaker service of the Semantic Service Platform stores capability advertisements of registered services as OWL-S profiles. As previously implemented in SEAGENT environment [7], the capability matching of the requested and registered service advertisements herein, is also based on the algorithm given in [8] and deals with semantic distances between input/output parameter concepts of related services. The details of the implemented capability matching may be found in [9] and [10]. However, in addition to the above mentioned matching algorithm, Matchmaker service of our proposed platform also supports semantic match only on types of services (excluding input/output parameter match). Therefore, a client may indicate his/her preferred capability matching approach to the matchmaker and matchmaker performs capability matching upon this client's preference.

Based on domain knowledge of the related application, the Semantic Service Platform provides a *meta-profile* definition for platform services those to be registered, discovered and invoked within the platform. Hence, in this approach, Semantic Service Kernel plans, that include client invocation codes, may be prepared easily by only using those predefined meta-profiles of services. However this naturally exposes an ontology mapping requirement when an outer service, that is needed to be included in the platform, has a different profile model than platform's meta-definitions. It is aimed to bring a solution to this problem by using capabilities of the ontology mapping and translation service of the External Service Agent. So, advertisement plan of the External Service Agent supports platform administrators to be able to map platform's meta-profile with the related external service's ontology by using mapping service via a user interface.

It should be noted that discovery process of the client, has already been defined as a reusable plan template in the Semantic Service Kernel. So, the content of this plan template is determined during domain based application development and this creates the application dependent plan of the discovery process. The client agent, which uses the related created discovery plan, first sends the required service's profile to the matchmaker service and receives advertisement profiles of semantically appropriate services. The suitable communication protocol and content language for the client has already been designed and implemented for OWL-S services [9].

Another important task of the discovery process is the service selection policy of the requester client. The Matchmaker of the platform may return a collection of suitable service profiles for the client's requests in many cases and client should apply a policy into the result collection to select the service(s) for further engagement and enactment processes. The Semantic Service Platform provides extensible service selection policy structures for plan designers to add various selection criteria into the service user agent plans.

### 2.2 Realization of service engagement process

After completion of the service selection, the client-service engagement process begins. The engagement process has 2 stages: (1) negotiation on quality of service metrics between client and service agents and (2) agreement settlement.

The first stage of the engagement process includes determination of the exact service according to quality of service (QoS) metrics. Currently, there exists no standard for the service quality metrics. However, during the exact service determination, our proposed service platform utilizes some QoS parameters (like service cost, run-time, location, etc.) defined in various studies [11,12] which address this issue. When both sides (client and service) agree on the quality metrics, the first stage of the process is finished.

The engagement process is completed after determined service's OWL-S process ontology and QoS parameters are sent to the Monitoring Service for being monitored during service execution.

### 2.3 Realization of service enactment process

Conceptually, enactment can be defined as the invocation of the engaged service by the client agent. However, in fact, enactment includes more than just invocation and it should take into consideration of monitoring, certification, trust and security requirements of service calls.

Execution of composite semantic services (modeled by using OWL-S) is maintained in the platform by means of a planning approach. The approach herein, provides definition of service templates for each atomic service of the composite service and realizes composition of the service by linking those atomic subprocesses.

Service execution also requires monitoring of the invocation according to the engagement between client agent and the server. Monitoring services of the Semantic Service Kernel and External Service Agent both monitor execution of services and control whether current interaction conforms into the predefined QoS metrics and engagement protocol or not. Hence, the Monitoring service of the External Service Agent informs the platform's monitoring service about the produced values of the quality metrics during service execution. According to the state of the ongoing interaction, the informed client agent may change his task execution behavior as defined in his enactment plan.

## 3 Planning Requirements of SWSA

The Semantic Service Kernel includes the required infrastructure and architectural components for an agent to execute subprocesses of SWSA. Such an environment simplifies the overall process of executing semantic web services for ordinary developers. Client agent(s) in this environment, must provide plans to be used in semantic service discovery, engagement and enactment. Hence, in our platform, we modeled these plans as reusable plans that are defined using the well known Hierarchical Task Network (HTN) planning structures. HTN Planning is an AI planning methodology that creates plans by task decomposition. This decomposition process continues until the planning system finds primitive tasks that can be performed directly. The basic idea of HTN planning was in the mid-70s [13,14], and the formal underpinnings were developed in the mid-90s [15]. In an HTN planning system, the objective is to accomplish a partially-ordered set of given tasks (plan) and a plan is correct if it is executable, and it accomplishes the given tasks. That is, the main focus of an HTN planner is to perform tasks, while a traditional planner focuses on achieving a desired state.

The planner of our MAS development framework, called SEAGENT, is based on the HTN planning framework presented by Sycara et. al [16] and DECAF architecture [17]. In SEAGENT, tasks might be either complex (called behaviours) or primitive (called actions). Each plan consists of a root task (behaviour) which is a complex task itself consisting of sub-tasks to achieve a predefined goal. Behaviours hold a 'reduction schema' knowledge that defines the decomposition of the complex task to the sub-tasks and the information flow between these subtasks and their parent task. The information flow mechanism is as follows: each task represents its information need by a set of *provisions* and the execution of a task produces *outcomes*, and there are links that represents the information flows between tasks using these provision and outcome slots. Actions, on the other hand, are primitive tasks that can be executed directly by the planner. Also, each task produces an outcome state after its execution. Default outcome state is "OK" and usually it is not shown. This outcome state is used to route the information flow between tasks.

Figure 2 shows the task tree of the HTN plan for service execution of client agents. "Execute Service" task represents the service execution process which was proposed by SWSA and is the root task of the plan that needs abstract characterizations of candidate services in order to be executed. It is composed of three sub-tasks: "Discover Candidate Services", "Engage with a Service" and "Enact Service" (decomposition of these tasks are not shown in this figure). "Discover Candidate Services" task inherits the abstract characterization of candidate services from its parent task and produces service profiles of candidate services after its execution completes. Then these service profiles are passed to "Engage with a Service" task via a provision link and then the execution of this task begins. "Engage with a Service" task finishes by producing two outcomes: selected service provider and service agreement. These outcomes are consumed by "Enact Service" task in order to complete the final part of the plan. This task executes selected service and passes the output list of its execution to its parent task.

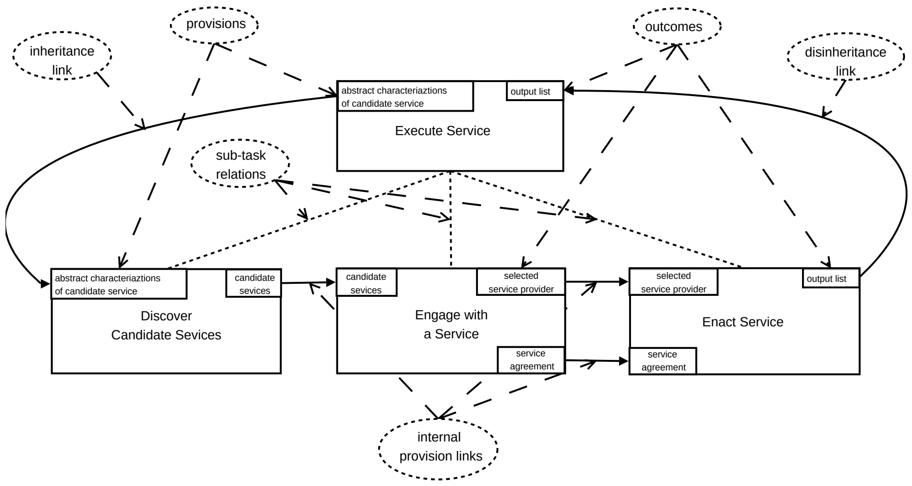

**Fig. 2.** HTN plan for SWSA based Semantic Service Execution

As discussed above, SWSA processes are controlled and monitored by Semantic Service Kernel. Planner is at the heart of the architecture which controls the workflows of the SWSA processes. Each of these processes are modeled as reusable plans and the developed planner must have the innovative features in order to have the capability of executing these kind of plans. These features are listed below:

– *Definition of reusable template plans that includes abstract task structures for SWSA's subprocesses and usage of these templates for generating agent's real plans by specializing these abstract tasks.*

  Some parts of processes introduced by SWSA is abstract because, they change according to domain. Hence, it must be possible to generate a template plan for SWSA and realize it on specific domains. Such template plan must include variable (or abstract) tasks which can be specified based on the domain requirements. So, the plan structure must provide mechanisms to define and specify such variable task constructs for developers. Without such a planning mechanism, it is impossible to define reusable plan templates for executing SWSA processes. For example, in service discovery, the client agent, first forms a query using the required service's profile, sends it to the matchmaker service, receives advertisement profiles of semantically appropriate services and finally applies a service selection policy to return a collection of suitable profiles. The reusable template plan for this service discovery process is illustrated in Figure 3. As shown in the figure, "Form a Query for Service Discovery" task and "Select Service(s)" task are abstract. Former is abstract because the query could be formed either according to service type or input/output parameter types or etc. Latter is abstract because developers should be able to use various service selection criteria.

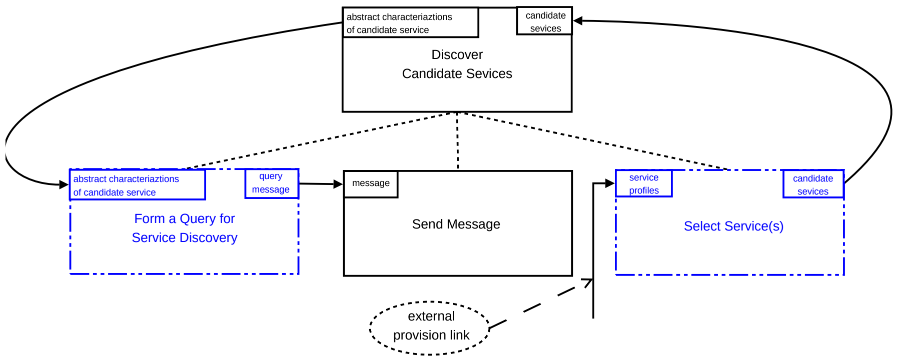
abstract characteriaztions of candidate service
candidate sevices
Discover Candidate Sevices
abstract characteriaztions of candidate service
query message
Form a Query for Service Discovery
message
Send Message
service profiles
candidate sevices
Select Service(s)
external provision link

**Fig. 3.** Decomposed view of the task for service discovery

– *Support for recursion on the plan structure.*

By a recursion we mean a situation where one instance of a task is an ancestor in the planning tree of another instance of the same task. Consider the engagement process of the client agent given in section 2.2. At first, one of the candidate services are chosen and then completeness of all service invocation parameters is assured. After the assurance, negotiation on QoS metrics and agreement settlement are performed. If either assurance or negotiation tasks fail, the engagement process will be restarted for unselected services (Figure 4). To provide this iteration the plan structure must have support for recursion. Such support needs the ability for a task to contain itself as a sub-task (in Figure 4, "Engage with a Service" contains itself as a sub-task).

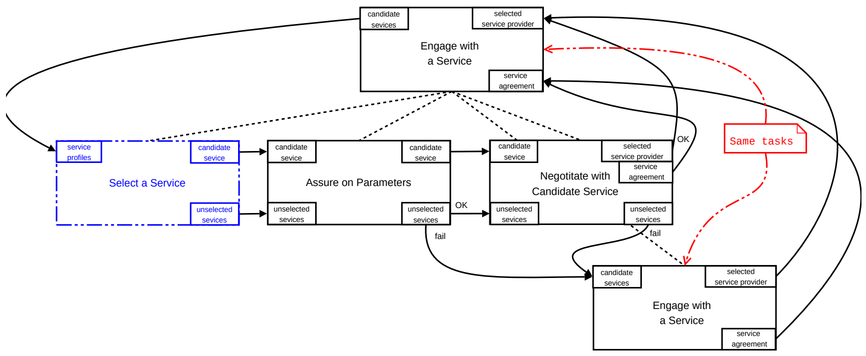

**Fig. 4.** Decomposed view of the task for service engagement

– *Constitution of composite services by using semantic service execution plans in agent's real plans.*

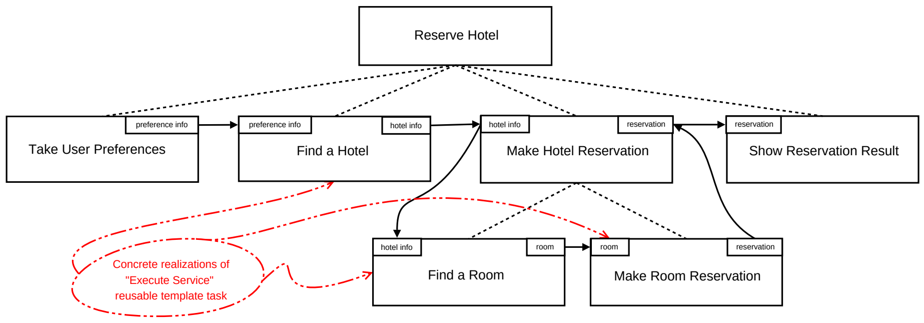

**Fig. 5.** Hotel reservation plan

To generate domain dependent service execution plans, developer must realize "Execute Service" template plan shown in Figure 2. Consider the HTN plan for reserving a hotel room shown in Figure 5. Reserving a hotel is as follows: first user preferences are taken, and according to these preferences a hotel is found, then it is tried to make reservation with that hotel and finally results are shown to the user. In this plan, "Find a Hotel", "Find a Room" and "Make Room Reservation" sub-tasks of the plan are concrete realizations of "Execute Service" task. They are connected with their provisions and outcome slots, and because they are domain dependent plans they know what input parameters they will take.

## 4 SEAGENT Planner

### 4.1 Internal Architecture

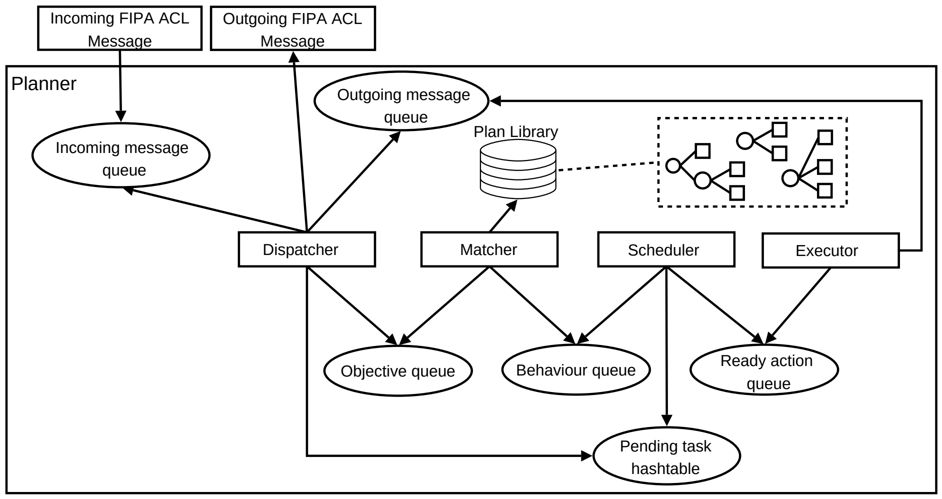

**Fig. 6.** Overall structure of SEAGENT planner

The overall structure of the planner architecture (Figure 6) is designed to execute HTN structure(s) that includes complex and primitive tasks. In order to execute a plan, the planner dynamically opens the complex root task using the 'reduction schema' knowledge, and this reduction continues until the planner finds actions (directly executable tasks), and then the planner executes these actions. Propagation of output values of the executed task to other dependent task(s) is handled by the "plan structure" itself. The planner is composed of four functional modules: dispatcher, matcher, scheduler and executor. Each module runs concurrently in a separate Java thread and uses the common data structures. All together, they match the goal extracted from the incoming FIPA-ACL message to an agent plan, schedule and execute the plan following the predefined HTN based plan. In the following, we briefly explain responsibilities of each module during plan execution.

The planner is responsible for processing incoming and outgoing messages, matching (finding), scheduling and executing tasks. When a FIPA-ACL message is received by the agent, it is enqueued to incoming message queue by the communication infrastructure layer. Dispatcher checks incoming message queue for the existence of an incoming message, if so, it parses the message and checks whether it is part of an ongoing conversation or not. If so, then the dispatcher finds out the task waiting for that message from pending task hash table, and sets the provision(s) of that task. If the incoming message is not part of an ongoing conversation, then the dispatcher creates a new objective, puts it to the objective queue.

Matcher is responsible for matching the incoming objective to a pre-defined plan by querying the plan library which is constructed using plan ontology and match ontology. When a new objective is occurred, matcher gets message information from it, and tries match a plan from the plan library for that objective. If match succeeds, Matcher creates an instance of the plan and enqueues it to behaviour queue.

Scheduler's role is to determine the execution time of each behaviour. Scheduler gets behaviours from the behaviour queue and prepares them for execution. If all of the provisions of the behaviour are set, Scheduler reducts that behaviour by calling its `reduct()` method - decomposes the behaviour to its sub-tasks. All sub-tasks are checked whether their all provisions are set or not. If provisions of a sub-task are set, then, if it is an action it is put to the ready action queue, else (if it is a behaviour) it is put to the behaviour queue. If not, the sub-task is put to the pending task hash table to wait for its provisions to be set. They may be action or behaviour.

Executor checks the ready action queue and if there are pending actions on that queue, it gets and executes them by invoking their `Do()` method. After execution of an action, produced outcomes are passed to the dependent task's provisions internally by the plan itself.

## 4.2 SEAGENT Plan Structure

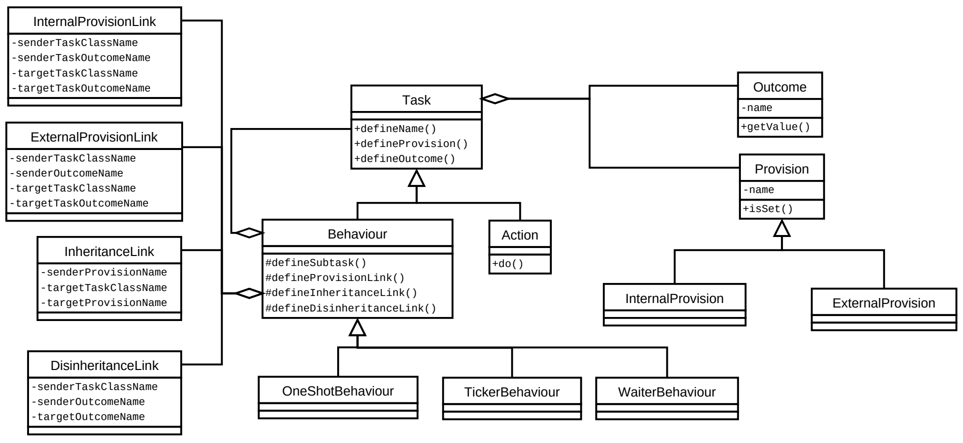

**Fig. 7.** Components of SEAGENT plan structure

Components of SEAGENT plan structure are shown in Figure 7 (only key attributes and operations are shown). Tasks are represented with `Task` class and have a name describing what it is supposed to do and have zero or more provisions and outcomes. Provisions might be of two types: internal provision and external provision. Internal provisions are provisions whose value's are internally set within the plan, in other words, value of internal provisions is determined by outcome of another tasks. External provisions, on the other hand, is set externally, with an incoming FIPA-ACL message. Incoming messages from another agents are routed to the external provisions of pending tasks. This routing is done according to *conversation-id* and *in-reply-to* fields of the incoming message. There are various types of behaviours, such as `OneShotBehaviour` which is executed only once, `TickerBehaviour` which is executed periodically and `WaiterBehaviour` which is executed after an amount of time is passed. All specifications about behaviours and actions is hold by themselves and, these specifications are described by using `defineXXX()` methods.

To write an action, it is enough to extend `Action` class, define provisions and outcomes using `defineProvision()` and `defineOutcome()` methods and finally implement `do()` method. Writing behaviours is a bit different because, behaviours have no `do()` method, and they only hold the specifications about its sub-tasks, and relationships between these sub-tasks. Also they may contain provision and outcomes, and, definition of provisions and outcomes is just as in actions. To define sub-tasks, `defineSubtask( String className )` method is used by giving the absolute class name of the task to be added to this behaviour. For example, "Enact Service" task in Figure 2 can be defined in "Execute Service" parent task as `defineSubtask( EnactService.class.getName() )`. And also provision links must be defined in order to satisfy information flow by using provision link definition methods[5].

### 4.3 SWSA Support in Planning Level

As mentioned in section 3, SEAGENT planner has innovative features to handle the workflows of the SWSA processes. This section shows how these features are satisfied by SEAGENT planner. The built-in support makes implementing overall processes of SWSA easier for developers.

**Template plan support** In SEAGENT it is possible to create template (generic) plans. The basic idea resembles abstract class logic in object orientation. That is, to construct a generic plan, describe the main characteristics of a plan leaving some special pieces (tasks) unimplemented. So that in the future they may be specialized and used (remember the service discovery plan in Figure 3). Tasks defined in reusable template plan can be extended (by redefinition of abstract tasks) to concrete plans. The important point here is the confliction of the specifications of the abstract task to be extended (provision and outcome definitions) and specifications of the extended concrete task. To implement template plans, construct the plan as a normal plan but use abstract task definitions where you want to make abstract tasks.

[5] `defineInternalProvisionLink()` and `defineExternalProvisionLink()`

In SEAGENT, to define sub-tasks, `defineSubtask()` method is used by giving the class name of the sub-task as a parameter. If we want to define an abstract sub-task, all we need to do is to give the name of the abstract sub-task indirectly. This can be done by using an abstract method that returns the name of the sub-task, and passing this abstract method as parameter to `defineSubtask()` method. For example, "Select Service(s)" abstract task in Figure 3 can be defined in "Discover Candidate Services" parent task by using `getSelectServicesTaskName()` abstract method ( `defineSubtask( getSelectServicesTaskName() )`. Concrete realization of this plan is made by extending this plan and implementing the abstract methods that return the class names of the concrete tasks.

**Recursion support** Recursion is another capability of SEAGENT planner. By a recursion we mean a situation where one instance of a task is an ancestor in the planning tree of another instance of the same task. This is usually used when we cannot satisfy something in a task and want to execute this task again but with different provisions. Since there is no restriction on defining sub-tasks, a task may contains itself as a sub-task. But the important point here is that the decomposition of the successor task must be in control, just like in recursive methods in traditional programming. In SEAGENT, the decomposition of a task is controlled via its provision's state, that is, if all provisions are set or there is no provision then the task decomposes. So, in a recursive HTN plan, recursion tasks must contain at least one provision and the value of this provision must be different from its ancestor's value (see Figure 4). Otherwise an infinite loop arises and our agent crashes.

**Constitution of Composite Services support** Constitution of composite services is simple in SEAGENT, because there are predefined reusable template plans for service execution (see Figure 2) in plan library and, developers can use these plans to construct their own domain dependent service execution plans and compose them just as building an ordinary plan (see Figure 5). The important point here is the satisfaction of input/output compatibility and this is easily handled by the correct concrete realizations of the abstract tasks.

## 5 Conclusion

The SWSA is currently a working group recommendation and describes abstract infrastructures and related processes for semantic web services and agents interaction in a conceptual base. We believe that this architecture brings a comprehensive model of software agents which utilize and provide semantic web services. However this architecture is a product of an initiative study and most of its components are only theoretically defined, not implemented. In this paper, a new MAS software platform, which aims to concretely fulfill fundamental requirements of the SWSA, has been introduced. We have modeled subprocesses of

SWSA as reusable plans by HTN approach and provided a framework in which those plans can be executed in a composite fashion by agent planners. Hence, platform agents can accomplish execution of discovery, engagement and enactment processes for semantic web service interaction by employing those reusable and predefined HTN plans. We have also discussed necessary properties of an agent planner which can execute those defined plans. Such a planner has been implemented based on the SEAGENT platform.

In the paper we focused on the requirements of the planner for execution of SWSA subprocesses. But we are also working on the other parts of the software architecture. For example, service discovery mechanisms for the platform are fully operational. Semantic capability matching of services has already been implemented and platform agents are currently able to invoke semantic web services in proper to OWL-S standards.

Perhaps our major weakness, considering both the software and reusable agent plans, is seen in definition and design of the service engagement subprocess. QoS topics are currently being studied and they weren't addressed in detail by the service oriented computing community. Hence our QoS support during the engagement process is extremely primitive and only composes monitoring service. That support is also in its initial state. Security and trust mechanisms have not been considered yet in our implementation.

## References

1. Burstein, M., Bussler, C., Zaremba, M., Finin, T., Huhns, M., Paolucci, M., Sheth, A., Williams, S.: A semantic web services architecture. IEEE Internet Computing **Volume 9 Issue 5** (2005) 72 – 81
2. Varga, L.Z., Ákos Hajnal, Werner, Z. In: Engineering Web Service Invocations from Agent Systems. Volume 2691. Lecture Notes in Computer Science (2003) 626 – 635
3. Varga, L.Z., Ákos Hajnal, Werner, Z. In: An Agent Based Approach for Migrating Web Services to Semantic Web Services. Volume 3192. Lecture Notes in Computer Science (2004) 381 – 390
4. Fensel, D., Bussler, C.: The web service modeling framework wsmf. Electronic Commerce Research and Applications **1**(2) (2002) 113–137
5. Greenwood, D., Calisti, M.: Engineering web service - agent integration. In: SMC (2), IEEE (2004) 1918–1925
6. Nguyen, T.X., Kowalczyk, R.: Ws2jade: Integrating web service with jade agents. In: AAMAS'05 Workshop on Service-Oriented Computing and Agent-Based Engineering (SOCABE'2005). (2005)
7. Dikenelli, O., Erdur, R.C., Özgür Gümüs, Ekinci, E.E., Önder Gürcan, Kardas, G., Seylan, I., Tiryaki, A.M.: Seagent: A platform for developing semantic web based multi agent systems. In: AAMAS, ACM (2005) 1271–1272
8. Paolucci, M., Kawmura, T., Payne, T., Sycara, K.: Semantic matching of web services capabilities. In: First Int. Semantic Web Conf. (2002)
9. Dikenelli, O., Gümüs, Ö., Tiryaki, A., Kardas, G.: Engineering a multi agent platform with dynamic semantic service discovery and invocation capability. In Eymann, T., Klügl, F., Lamersdorf, W., Klusch, M., Huhns, M.N., eds.: MATES. Volume 3550 of Lecture Notes in Computer Science., Springer (2005) 141–152

10. Kardas, G., Gümüs, Ö., Dikenelli, O.: Applying semantic capability matching into directory service structures of multi agent systems. In: ISCIS. Volume 3733 of Lecture Notes in Computer Science., Springer (2005) 452–461
11. Zeng, L., Benatallah, B., Dumas, M., Kalagnanam, J., Sheng, Q.Z.: Quality driven web services composition. In: WWW '03: Proceedings of the 12th international conference on World Wide Web, New York, NY, USA, ACM Press (2003) 411–421
12. Cardoso, J., Sheth, A.P., Miller, J.A., Arnold, J., Kochut, K.: Quality of service for workflows and web service processes. J. Web Sem. **1**(3) (2004) 281–308
13. Sacerdoti, E.: The nonlinear nature of plans. In: International Joint Conference on Artificial Intelligence. (1975)
14. Tate, A.: Generation project networks. In: International Joint Conference on Artificial Intelligence. (1977) 888 – 893
15. Erol, K., Hendler, J.A., Nau, D.S.: Complexity results for htn planning. Ann. Math. Artif. Intell. **18**(1) (1996) 69–93
16. Sycara, K., Williamson, M., Decker, K.: Unified information and control flow in hierarchical task networks. In: Working Notes of the AAAI-96 workshop 'Theories of Action, Planning, and Control'. (1996)
17. Graham, J.R., Decker, K., Mersic, M.: Decaf - a flexible multi agent system architecture. Autonomous Agents and Multi-Agent Systems **7**(1-2) (2003) 7–27