A BDI Agent Approach to Healthcare Web Services

Hoon Jin and In-Cheol Kim

Kyonggi University, Department of Computer Science, Suwon-si, KOREA

Abstract— **Although there are many efforts for automated semantic web service composition, most of them are based upon the traditional AI planning techniques. Openness and dynamics of the Web environment limits the usage of previous approaches based upon the traditional AI planning techniques. This paper introduces a BDI agent system for semantic web service composition and invocation. Through some tests on healthcare web services, we found our BDI agent approach has the potential enough to improve robustness and flexibility of semantic web services.**

Keywords— **Semantic Web Service, Service Composition, BDI Agent Architecture, OWL-S, Healthcare Service**

I. Introduction

Semantic web services augment web services with rich formal descriptions of their capabilities, thus facilitating automated discovery, composition, dynamic binding, and invocation of services within an open environment. Semantic web services, therefore, have the potential to change the way knowledge and business services are consumed and provided on the Web. Enabling semantic web services needs the infrastructure such as standard service ontology and architecture. The service ontology, such as OWL-S, aggregates all concept models related to the description of a semantic web service, and constitutes the knowledge-level model of the information describing and supporting the usage of the service. Currently several tools supporting OWL-S have been developed: OWL-S Editor, Matchmaker, Broker, Composer, and Virtual Machine [4].

The current semantic web service architectures do not provide with integrated functionality of composition, dynamic binding, and invocation [1]. Although there are some efforts for automated semantic web service composition, most of them are based upon the traditional AI planning techniques. Due to openness and dynamics of the Web environment, the web services composed through off-line planning are subject to fail. In order to overcome the drawback, this paper introduces a BDI agent system for semantic web service composition and invocation. Using some examples of healthcare web services, we test the feasibility of our approach.

II. BDI Agent Architecture

The most commonly used architecture for software agents is the Belief-Desire-Intention (BDI) model. Fig. 1 shows a PRS-based BDI architecture called JAM [3]. Each JAM agent is composed of five primary components: a world model, a goal set, a plan library, an interpreter, and an intention structure. The world model is a database that represents the beliefs of the agent. The goal set is a set of goals that the agent has to achieve. The plan library is a collection of plans that the agent can use to achieve its goals. The interpreter is the agent's "brain" that reasons about what the agent should do and when and how to do it. The intention structure is an internal model of the agent's current goals and keeps track of the commitment to, and progress on, accomplishment of those goals.

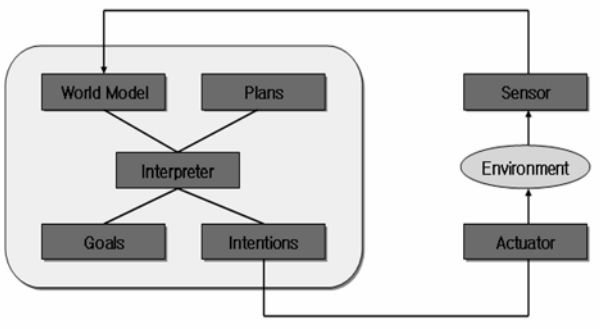

Fig. 1 JAM: a BDI agent architecture

The BDI architecture integrates traditional goal-directed reasoning and reactive behavior. Because most traditional deliberative planning systems formulate an entire course of action before starting execution of a plan, these systems are brittle to the extent that features of the world or consequences of actions might be uncertain. In contrast, the BDI architecture continuously tests its decisions against its changing knowledge about the world, and can redirect the choices of actions dynamically while remaining purposeful to the extent of the unexpected changes to the environment.

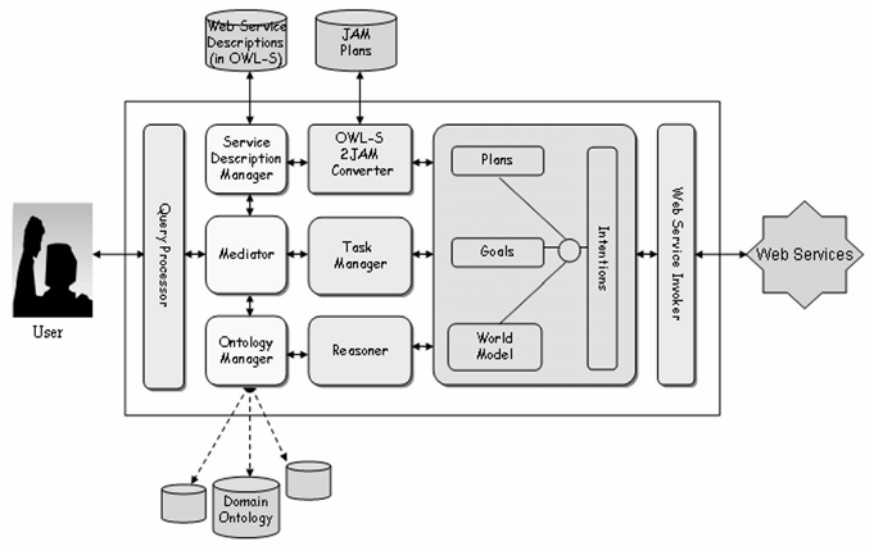

Fig. 2 Architecture of SWEEP II system

III. SWEEP II SYSTEM

SWEEP II is a BDI agent system supporting automated semantic web service composition and invocation. As shown in Fig. 2, the core component of SWEEP II is *the JAM BDI engine*. Additionally, SWEEP II includes several main components such as *service description manager*, *OWL-S2JAM converter*, *ontology manager*, *reasoner*, *query processor*, *mediator*, *task manager, web service invoker*. The service description manager retrieves and stores OWL-S descriptions from the semantic web service repository. The OWL-S2JAM converter transforms the retrieved OWL-S descriptions into the corresponding JAM primitive plans. Based upon the domain ontologies managed by the ontology manager, the reasoner provides reasoning services for service matching, validation, and mediation. The query processor takes a request from the user. With the help of the mediator, it generates a task to be accomplished by JAM. In order to accomplish the given task, JAM reactively elaborates a complex plan from primitive plans considering its goals and the world model. According to the intentions of JAM, the web service invoker calls the atomic web services in order.

Fig. 3 illustrates the mapping relationship from an OWL-S service description to the corresponding JAM plan. Outputs and effects of the OWL-S process are mapped to goals and effects of the JAM plan. Inputs and preconditions of the OWL-S process are mapped to subgoals and context of the JAM plan. The grounding WSDL operations of the OWL-S service are mapped to the external functions of the JAM plan. Many ontological concepts and properties in the OWL-S description are converted into the corresponding relations within the JAM world model. Fig. 4 shows an example of OWL-S service description and Fig. 5 represents the corresponding JAM plan.

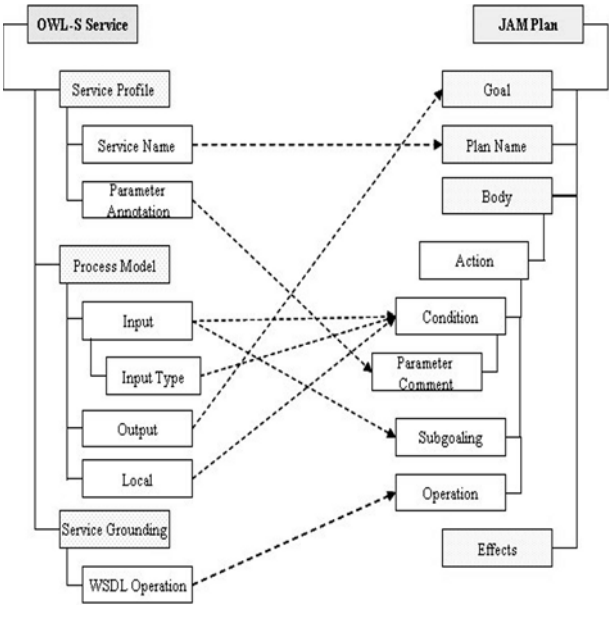

Fig. 3 OWL-S to JAM plan mapping

```
<?xml version="1.0"?>
  <owl:Ontology rdf:about="">
    ...
    <owl:imports rdf:resource="http://localhost/resources/ontology/Healthcare.owl"/>
  </owl:Ontology>
  <process:AtomicProcess rdf:ID="DiseaseSearchProcess">
    <process:hasOutput>
      <process:Output rdf:ID="disease">
        <process:parameterType rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
            http://localhost/resources/ontology/Healthcare.owl#Disease
        </process:parameterType>
      </process:Output>
    </process:hasOutput>
    <process:hasInput>
      <process:Input rdf:ID="pain_name">
        <process:parameterType rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
            http://localhost/resources/ontology/Healthcare.owl#Pain</process:parameterType>
        <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
            It's a name representing about pain.</rdfs:comment>
      </process:Input>
    </process:hasInput>
    <process:name rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >DiseaseSearchProcess</process:name>
    <service:describes>
      <service:Service rdf:ID="DiseaseSearch">
        <service:providedBy rdf:resource="#DiseaseSearch"/>
        ...
        <service:supports>
          <grounding:WsdlGrounding rdf:ID="DiseaseSearchGrounding">
            <grounding:hasAtomicProcessGrounding>
              <grounding:WsdlAtomicProcessGrounding rdf:ID="DiseaseSearchProcessGrounding">
                <grounding:owlsProcess rdf:resource="#DiseaseSearchProcess"/>
                <grounding:otherReference
                rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
                    http://localhost:8080/axis/services/UFdiseaseSearch2?wsdl
                </grounding:otherReference>
                ...
  </process:AtomicProcess>
</rdf:RDF>
```

Fig. 4 Example of OWL-S description

Due to features of the embedded JAM engine, the SWEEP II system can adapt its decisions on web service composition and invocation against the changing Web environment. In this sense, the SWEEP II system realizes *dynamic composition* of semantic web services. Fig. 6 shows a screenshot of the SWEEP II system.

In order to test the feasibility of our SWEEP II system, we developed some examples of healthcare web services. They are disease inquiry service, clinic search service, expert recommendation service, registration and appointment service, and so on. For our purpose, however, we implemented them as OWL-S semantic web services. Through some tests on these semantic web services, we found that our SWEEP II system shows high robustness against unexpected failures and delays of web services.

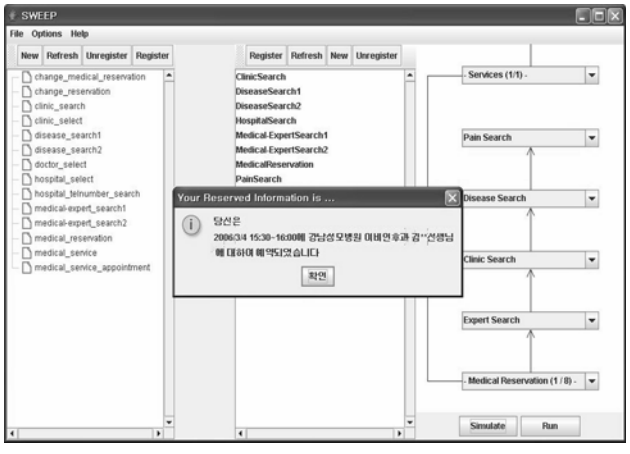

Fig. 6 Screenshot of SWEEP II system

```
plan: {
NAME: "Disease Search"
GOAL: ACHIEVE disease_known "True";
PRECONDITION: RETRIEVE disease_known $known; (== $known "False");
BODY:
    ASSIGN $service_name "Disease Search";
    ASSIGN $currServ_paramNumIn 1;
    RETRIEVE simul_mode $smode;
    OR
    {   TEST(== $smode "OFF");

        ASSIGN $cmt_param "Pain: ";
    ASSIGN $param (com.irs.jam.primitives.RequestUserInput.execute $cmt_param $param);
        WHEN: TEST (!= $param "") { ASSIGN $pain_name $param; };
        WHEN: TEST (== $param "") {
            DO{ OR{ ACHIEVE pain_name_known "True";
                    RETRIEVE pain_name_known $known; }
                  { EXECUTE println "Subgoaling Error!"; };
                  } WHILE: TEST(!= $known "True" );
                RETRIEVE pain $pain_name;  }; };
    ASSIGN $service_WSDL "http://localhost:8080/axis/services/DiseaseSearch?wsdl";
        EXECUTE com.irs.jam.primitives.ServiceCall.execute $service_name $service_WSDL
        $pain_name $disease;
        WHEN: TEST (!= $disease "") { ASSERT disease $disease; }; }
    {   TEST(== $smode "ON");
        DO{ OR{ ACHIEVE pain_known "True";
                RETRIEVE pain_known $known;  }
              { EXECUTE println "Subgoaling Error!"; };
            } WHILE: TEST(!= $known "True");
        SUCCEED; };
EFFECTS:
    EXECUTE com.irs.jam.primitives.GetCurrentPlanName.execute $service_name $planList;
    UPDATE (disease_known)(disease_known "True");
}
```

Fig. 5 The corresponding JAM plan

IV. Conclusions

We introduced SWEEP II, which is a BDI agent system for semantic web service composition and invocation. The core component of SWEEP II is the JAM BDI engine. The JAM BDI architecture continuously tests its decisions against its changing knowledge about the world, and can redirect the choices of actions dynamically. Through some tests on healthcare web services, we found that our SWEEP II system can help to improve robustness and flexibility of semantic web services.

References

1. Cabral, L., Domingue, J., Motta, E., Payne, T., Hakimpour, F. (2004) Approaches to Semantic Web Services: An Overview and Comparisons. Proceedings of the 1st European Semantic Web Symposium (ESWS2004), Springer-Verlag, 2004, pp 225-239
2. Neto, R., Udupi, Y., Battle, S. (2004) Agent-Based Mediation in Semantic Web Service Framework. Proceedings of the 1st AKT Workshop on Semantic Web Services (AKT-SWS04), 2004
3. Marcus, J. (1999) JAM: A BDI-theoretic Mobile Agent Architecture. Proceedings of the 3rd International Conference on Autonomous Agents (Agents'99), 1999, pp 236-243
4. Paolucci, M., Sycara K. (2003) Autonomous Semantic Web Services. IEEE Internet Computing, 7:34-41
5. Paolucci, M., Soudry, J., Srinivasan, N., Sycara K. (2004) A Broker for OWL-S Web Services. Proceedings of 2004 AAAI Spring Symposium on Semantic Web Services, MIT Press, 2004
6. Paolucci, M., Ankolekar, A., et al. (2003) The DAML-S Virtual Machine. Proceeding of 2nd International Semantic Web Conference (ISWC2003), Springer-Verlag, 2003, pp 290-305

Address of the corresponding author:

Author: Jin, Hoon
Institute: Department of Computer Science, Kyonggi University
Street: San94-6, Yiui-dong, Youngtong-gu
City: Suwon-si
Country: KOREA
Email: jinun@kyonggi.ac.kr