TRANSFORMATION OF ONTOLOGICAL REPRESENTED WEB SERVICE COMPOSITION PROBLEM INTO A PLANNING ONE

Zoltán ĎURČÍK, Ján PARALIČ
Department of Cybernetics and Artificial Intelligence, Faculty of Electrical Engineering and Informatics,
Technical University of Košice, Letná 9, 042 00 Košice, Slovak Republic, tel.: +421 55 602 4128,
e-mail: zoltan.durcik@tuke.sk, jan.paralic@tuke.sk

ABSTRACT

This article deals with automated web service composition (AWSC). We present an approach utilizing a combination of artificial intelligence planning methods with knowledge-based approaches to AWSC. For primary composition problem description we used OWL and OWL-S ontologies, which are than in next step translated into a planning problem. This planning problem is represented in PDDL language and may be solved in arbitrary PDDL planner. For the translation process we introduced several original algorithms, which are implemented as components of a prototype system for AWSC.

Keywords: *web service, composition, planner, planning, transformation, ontology, OWL, OWL-S, PDDL*

1. INTRODUCTION

Nowadays the service oriented approach to design and implementation of information systems is very hot area of research and development. One of the most interesting and discussed issues in this area is a problem of web service composition, especially automated web service composition [3]. Web services composition deals with workflow creation and instantiation. The workflows are composed from available web services with some dataflow. If it is not possible to fulfill the user's request by one web service, there is a possibility to accomplish this request by web service composition.

A very simple language translation problem may be presented as the example of AWSC. Let us assume that there exist several web services, from which each is able to translate a word from one language into another one (e.g. from English to Slovak, from English to German etc.). Now let us assume, that it is needed to translate word from language X to language Y, but there is not a web service available for this direct translation. But there are two web services available, from which first translates a word from language X to language Z, and second translates a word from language Z to language Y. From this can be seen, that with composition of these two web services (i.e. output from first service will be fed as input into the second service) we may translate a word from language X to language Y (accomplish the original users' request).

Web services (WS) are distributed programs located on a computer network (most frequently on internet) and using standard protocols for communication (most frequently HTTP and SOAP). The concept of WS was introduced by major IT Corporations as Microsoft, IBM and Sun and was proposed as alternative to object-oriented distributed standards as CORBA and Java RMI.

There are two main properties of WS: they must be self-descriptive and they must be interoperable together regardless to the environment (i.e. also the program language), in which they were created. There are several standards and technologies, which are related with WS and are important in terms of the AWSC problem:

- WSDL (Web Services Description Language) - is the most used language for WS description. It makes it possible to describe the offer of available operations in a WS, parameters of these operations, as well as the way of communication with WS.
- SOAP (Simple Object Access Protocol) - is the XML based protocol. It serves for interaction and information exchange over the network interface by using XML messages. It is platform and programming language independent.
- OWL (Ontology Web Language) [1] - is a language designed for ontology description on the internet. It belongs to languages, which serve for knowledge representation, and it is approved by W3C consortium. In the computer science ontology represents a formal knowledge representation by a set of concepts and relationships among these concepts in some domain. OWL is semantic language for publishing and sharing ontologies.
- OWL-S (Semantic Markup for Web Services) [2] - may be characterized as ontologically descriptive language for WS. The base WS descriptive languages (as e.g. WSDL) are in terms of semantic web insufficient (mainly for automation of their activity on the internet). But with incoming ontologies it was shown, that there is a possibility to describe WS by ontologies, and connect these descriptions to existing ontologies. These descriptive ontologies for WS were called OWL-S (for reason that they are OWL ontologies and they are used for service description). Main motivation for creation of OWL-S were the following tasks: automated WS discovery, automated WS invocation and automated WS composition.

Automated web service composition is a special case of web service composition [3], where the process of composition creation is automatic. There exist various methods for AWSC (e.g. situation calculus [6], Petri nets, artificial intelligence methods [8] etc.).

2. AUTOMATED COMPOSITION OF WEB SERVICES

The process of automated web services composition may be divided into several parts. In a simple general system architecture for an AWSC system the following main parts can be identified:

- user – a consumer of the system, who provides requests to the system,
- translator – translates users' requests in a form suitable for processing by the process generator,
- process generator – it solves user's request, i.e. creates a workflow composed from abstract WS,
- evaluator – it selects the best solution if there are available multiple possibilities,
- executor – it executes selected workflow via composition of concrete WS and provides the results to the user.

The composition problem is in AWSC system represented by two specifications:

- external specification and
- internal specification.

External specification is used for the primary composition problem definition. There are usually used semantic web technologies. External specification then contains user's request and WS specifications.

Internal specification defines composition problem in relation with selected process generator. One of the most suitable choices for AWSC is use of artificial intelligence planning methods for solving the composition problem. In such a case the internal specification is represented as a planning problem. Planning problem is defined by some formal planning language.

3. PROPOSAL OF AN AWSC SYSTEM

Our proposal for an AWSC system architecture is presented on the Fig. 1. This proposal is a specification of the general AWSC system architecture, which can be found in [3].

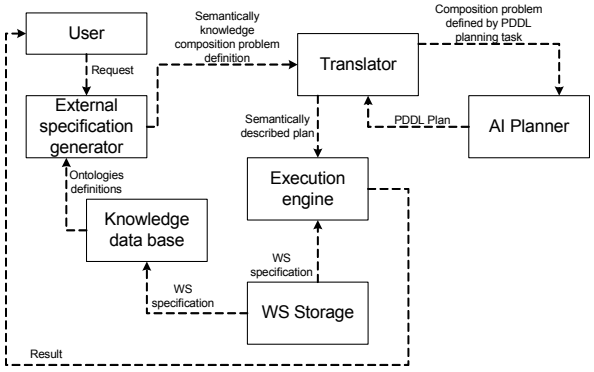

Fig. 1 Proposal for an AWSC system architecture

AI (Artificial Intelligence) Planner has been chosen as a process generator. This means that our internal specification is represented as a planning problem. In general the planning problem is represented as a quintuple $<S,S0,G,A,\Gamma>$ [8], where:

- S represents a set of all possible states in given model of the world,
- $S0$ is a subset of S, and represents the initial state,
- G is a subset of S, and represents the goal state,
- A is a set of all available actions. Each of them changes the world state by passing world state from one state to another,
- relation Γ is a subset of $S \times A \times S$ and defines preconditions and effects for each action.

In the system we use knowledge approaches. Every web services composition is performed in some domain (e.g. language translation domain, travel domain, crisis management etc.). It is important to have this domain described in a formal way. In our system we assume that this domain is described by means of OWL ontologies and these ontologies will be further referred as domain ontologies. We are able to create some state of the world in given domain by creating OWL individuals in the domain ontology. Therefore by this domain ontology we can create initial and goal state of a composition problem.

For WS description we will use ontology as well. This ontology for WS is formalised in OWL-S and represents WS operations as processes. Processes from OWL-S WS description then will represent actions in planning task. In OWL-S are processes described by IOPE – Input, Output, Preconditions and Effects. Therefore OWL-S may be used also for representation Γ of PDDL actions.

Domain, initial and goal OWL ontologies together with OWL-S web services description create external specification, and in the system are called as semantic knowledge composition problem definition. This definition must be translated into a planning problem by a translator. The process of translation from external specification to an internal one is the main topic of our work presented in this article.

After creation of a planning problem we are able to solve it using suitable external planner. In our system we can use available AI planners [8] like GraphPlan[1] or FF – Fast Forward planner[2] [5]. Besides these there is possibility to use arbitrary AI planner. The only condition is to use planner, which is able to work with PDDL language [10]. Solved problem is afterwards represented by a plan, which consists of available PDDL actions. Now there is the necessity for mapping these PDDL actions into related WS described by OWL-S. This task is realized in translator again. Finally, we can execute obtained WS and provide results to the user. This process of transformation PDDL plans back into semantic representation and execution of the translated PDDL plan is not discussed in this work. See [13] for more details.

3.1. Creation of PDDL description of a planning problem from semantically described WS composition problem

As it has been mentioned above, there exist two problem specifications in AWSC system: external and internal specification. In our case it holds:

[1] GraphPlan – http://sourceforge.net/projects/pdd4j/
[2] FF - http://personal.cis.strath.ac.uk/~ac/JavaFF/

- external specification is represented by means of available domain ontology, initial and state ontologies created from domain ontology, together with web services described in OWL-S,
- internal specification is represented as PDDL planning problem.

PDDL (Planning Domain Definition Language) language [10] arose is a result of standardisation efforts in the planning domain. It has been proposed and further developed for international planning competition requirements ICP and this language is very popular in the planning community.

The process of translation between the external and internal problem specification in our system is showed on Fig. 2.

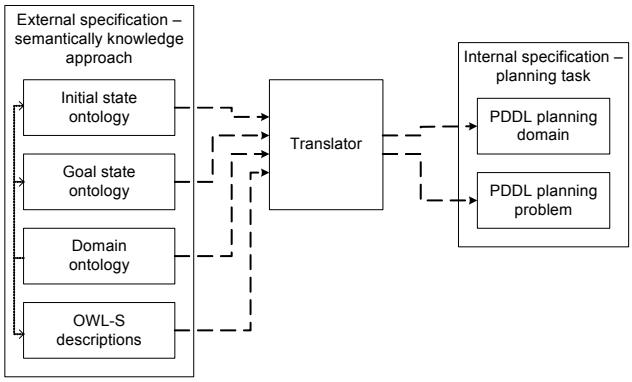

Fig. 2 Planning task creation problem

The OWL ontology structure may be divided into three parts:

- namespace definition,
- ontology header,
- ontology elements.

For every element, which is used in the ontology, it has to be specified, from where it comes from. This is either from some other already defined ontology, referencing it via the namespace, or it can be defined in the last part of the ontology. In the header of the ontology additional information can be provided, like e.g. its creator, date of creation, ontology title, ontology comments etc. After namespaces and ontology header definitions original ontology elements are defined. Main ontology elements are:

- OWL classes,
- OWL properties (object properties, data properties),
- OWL individuals (instances).

OWL-S represents ontological description for WS. This description consists of three parts:

- service profile – it describes what a particular web service provides for the consumers,
- service model – describes how the web service works,
- service grounding – provides necessary details about transport protocol and how one can interact with WS.

For the AWSC purpose the most important is service model, which describes WS as processes with their inputs, outputs, preconditions and effects. There are several possibilities how to define preconditions and effects in OWL-S. In our proposal we are using SWRL (Semantic Web Rule Language) conditions' specification as OWL-S preconditions and effects.

PDDL planning task consists from planning domain and planning problem definition (see Fig. 2). These may be further divided into smaller parts. Each from these parts will be presented in the next subsection. PDDL planning domain and problem structures are often defined by EBNF (Extended Backus-Naur Form). Complete definition of PDDL planning domain and problem can be found e.g. in [10].

3.2. PDDL planning domain creation

EBNF form of PDDL planning domain may be described in the following simplified form:

```
<domain> ::= (define (domain <name>)
                    [<require-def>]
                    [<types-def>] :typing
                    [predicates-def>]
                    <action-def>*)
```

For creation of PDDL planning domain we will use OWL domain ontology and OWL-S WS description. Each part of this process is described in particular subsection below, together with proposed transformation algorithms in pseudo-code.

3.2.1. Domain name

Domain name is represented as PDDL name. PDDL name is a string of characters beginning with a letter and next containing only letters, or digits, or hyphens (-), or underscores (_). Domain name can be created by using domain ontology, and may be located in header of this ontology (e.g. as title of ontology). This is the reason why the string obtained from ontology header should contain only permitted characters. In such a case a simple algorithm for PDDL domain name creation (Alg. 1) satisfies our purpose.

PDDL domain name algorithm

```
FUNCTION owl2pddlDomainName(O:OWLOntology):PDDLName
  VAR domainName : PDDLName;
  BEGIN
    domainName := getOntologyTitle(O);
    owl2pddlDomainName := domainName;
  END;
```

Alg. 1 Algorithm for PDDL domain name creation

3.2.2. Requirements

This is an optional element of PDDL domain structure. In case that this element is not included, there is an assumed only one requirement, namely STRIPS [4] (`:strips`). In our case we create PDDL domain from OWL ontologies and therefore we need also other requirements, e.g. the possibility of type definition

(`:typing`). There is an expectation, that requirements we may define in ontology header as additional ontology information. Currently the PDDL domain requirements are created manually in our AWSC system.

3.2.3. Types

PDDL types may be created from OWL classes and from a relationship Class\SuperClass in OWL ontologies. PDDL types are created for each OWL class definition. In case when a particular OWL class has defined a superior class (SuperClass), also the corresponding PDDL type has superior type. If superior class does not exist, superior type for corresponding PDDL type is an object.

PDDL types are represented by a special list called typed list. This list can assign types to its elements. But this possibility is available only in case if there is requirements flag for typing (see antecedent section). If this flags is not presented, the typed list is a normal list of elements.

Our algorithm for PDDL type created based on OWL domain ontology can be seen on Alg. 2.

PDDL domain types algorithm

```
FUNCTION owl2pddlTypes(O : Owl Ontology):PDDL Typed List
  VAR
    TL : PDDL Typed List;
    CS : Set Of Ordered Pair - OWL Class\SuperClass;
    C  : Set Of String - OWL Class;
    S  : Set All Different Types - PDDL Types;
  BEGIN
    C := getAllClassesFromOntology(O);
    FOR c TO C DO BEGIN
      VAR
        pom : Pair Represent Class\SuperClass
      pom.Class := c;
      pom.SuperClass := getSuperClassForOWLClass(O,c);
      IF (SuperClass is NULL) THEN BEGIN
        SuperClass:="Object";
      END;
      IF (NOT(S Don't contain Pom.SuperClass)) THEN BEGIN
        add pom.SuperClass into S;
      END;
      add pom Into CS;
    END;
    FOR s TO S DO BEGIN
      VAR
        type : String;
        X : Set All Classes With SuperClass=s;
      type := s;
      FOR cs TO CS DO BEGIN
        IF (cs.SuperClass = s) THEN BEGIN
          add cs.Class into X;
        END;
      END;
      add X with type into TL;
    END;
    owl2pddlTypes : TL;
  END;
```

Alg. 2 Algorithm for identification of PDDL domain types

3.2.4. Predicates

A predicate may represent property or relationship between entities. Predicates are in PDDL domain structures represented as atomic formulas skeletons. Each skeleton represents one predicate and consists from predicate name and from a set of parameters. Predicates may be considered as patterns, by which it is possible to create facts. These facts are created by substitution of predicate variables by concrete objects.

Predicates will be created from OWL domain ontology. For creation of PDDL predicates we use object properties, data properties and OWL classes' definitions.

Our algorithm for definition of PDDL predicates based on OWL domain ontology is presented on Alg. 3.

3.2.5. Actions

Each PDDL action consists of an action name (called also action functor), action parameters and action body. Action body further contains preconditions and effects for respective PDDL action.

PDDL domain predicates algorithm

```
FUNCTION owl2pddlPredicates(O : OWL Ontology)
                                              :PDDL Predicates
  VAR
    V : Set OWL Object and Data Properties;
    P : Set of OWL Classes from Properties Domains;
    C : Set of All OWL Classes;
    Preds  : PDDL Predicates;
  BEGIN
    V:=getAllOWLOntologyProperties(O);
  FOR v TO V DO BEGIN
    VAR
      d : Domain of Property v;
    IF (Set P don't contain d) THEN BEGIN
      add d into P;
    END;
  END;
  FOR p TO P DO BEGIN
    VAR
      Pred : PDDL Predicate;
    Pred.PredicateName := p;
    FOR v TO V DO BEGIN
      IF (The domain of v is p) DO BEGIN
        ParameterName := v;
        ParameterType := range of v;
        add ParameterName with ParameterType into
          Pred.Parameters;
      END;
    END;
    ordering all parameters from pred.Parameters by types;
    add Pred into Preds;
  END;
  FOR c TO (C exclude P) DO BEGIN
    VAR
      Pred : PDDL Predicate;
      Pred.PredicateName := c;
      ParameterName := c;
      ParameterType := c;
      add ParameterName with ParameterType into
          Pred.Parameters;
      add Pred into Preds;
  END;
  owl2pddlTypes : Preds;
END;
```

Alg. 3 Algorithm for creation of PDDL domain predicates

For creation of PDDL actions we use OWL-S web services descriptions. Each OWL-S description consists of three main parts: profile, process and grouding. In terms of AWSC the process model is important. Process model represents operations as particular processes. An OWL-S process is described by its IOPE (inputs, outputs, preconditions and effects). From this IOPE we create PDDL action parameters and action body. For PDDL action name we use the name of related OWL-S process.

There exist three types of processes in OWL-S description:

- *Atomic process* is simplest OWL-S process. From the user's point of view this process represents one step and there is a direct web service invocation for its execution. This WS is described by OWL-S.
- *Simple process* is not directly executable. Likewise as atomic processes are simple processes considered as one step processes too. Simple

process serves for abstract description of atomic and composite processes.

- *Composite process* is a process composed from subprocesses. A decomposition divides a composite process into simpler parts, which are performed by given constructors (sequence, split, split-join, choice, if-then-else, iterate, repeat until and repeat while). A composite process execution inheres in execution its sub-processes.

For every type of OWL-S process we must provide particular way for transformation of this process into PDDL action(s). Our algorithm for PDDL actions creation is shown on Alg. 4, Alg. 5, Alg. 6 and [12].

PDDL domain actions algorithm

```
FUNCTION owls2pddlActions(sO : OWL-S Ontologies)
                                          : PDDL actions
  VAR
    Processes : Set of OWL-S Processes;
    Actions: Set of PDDL actions;
  BEGIN
    FOR o TO sO DO BEGIN
      VAR
        process : OWL-S Process;
      process := getProcessFromOWLSOntology(o);
      add process into Processes;
    END;
    FOR p TO Processes DO BEGIN
      VAR
        a : PDDL action;
      IF (p is OWL-S atomic process) THEN BEGIN
        a := owls2pddlActionAtomicProcess(p);
      END;
      IF (p is OWL-S composite process) THEN BEGIN
        a := owls2pddlActionCompositeProcess(p);
      END;
      IF (p is simple process) THEN BEGIN
        a := owls2pddlActionSimpleProcess(p);
      END;
      add a into Actions;
    END;
    owls2pddlActions : Actions;
  END;
```

Alg. 4 Algorithm for PDDL domain actions definition

Because of complexity of algorithms for definition of simple and composite processes and the article size limit we describe here only the way of creation of PDDL actions from OWL-S atomic processes. The others are described in [12].

PDDL domain actions algorithm – atomic process

```
FUNCTION owls2pddlActionAtomicProcess(P :
                        OWL-S Atomic Process) : PDDL Action
  VAR
    a : PDDL Action;
  BEGIN
    IF (P is atomic process with only outputs) DO BEGIN
      a := owls2pddlActionAtomicProcessOutputs(P);
    END;
    IF (P is atomic process with only effect) DO BEGIN
      a := owls2pddlActionAtomicProcessEffects(P);
    END;
    owls2pddlActionAtomicProcess := a;
  END;
```

Alg. 5 Algorithm for PDDL actions derived from OWL-S atomic processes

In case of atomic process translation we assume that each atomic process contains only outputs or effects, not both [7], [11]. Therefore the algorithm presented on Alg. 5 includes calling of one from two available functions with regard to type of atomic process.

The function for creation of PDDL actions from atomic processes, which contains only effects, can be seen on Alg. 6. This is the simple PDDL action creation from OWL-S process. For other PDDL actions creation algorithms please refer to [12].

For PDDL action name we use the name of particular OWL-S process. Inputs of this process are translated into PDDL parameters. Each of these inputs may have type and therefore they are in PDDL actions represented by typed list (see subsection 3.2.3).

Each PDDL action has action body, which presents preconditions and effects. We are able to create these parts by using OWL-S process preconditions and effects.

PDDL domain actions algorithm – atomic process

```
FUNCTION owls2pddlAction_AtomicProcessEffects(P :
                        OWL-S Atomic Process): PDDL  Action
  VAR
    a : PDDL Action;
    Preds : PDDL Domain Predicates;
    Prec : Preconditions;
  BEGIN
    a.ActionName := getNameOfProcess(P);
    FOR i TO All P Inputs DO BEGIN
      add i with corresponding type into a.Parameters;
    END;
    ordering all parameters from a.Parameters by types;

    add to a.ActionBody.Preconditions :=
        swrl2pddlPreconditions(P, Preds);
    add to a.ActionBody.Effects :=
        swrl2pddlEffects(P, Preds);

    owls2pddlAction_AtomicProcessEffects := a;
  END;
```

Alg. 6 Algorithm for PDDL actions derived from OWL-S processes with effects only

The condition in OWL-S may be defined in several ways (e.g. by using SWRL – Semantic Web Rule Language, KIF - Knowledge Interchange Format etc.). In presented work we use SWRL conditions for definition of preconditions and effects in OWL-S process model.

Our algorithms of creation of PDDL preconditions and effects from SWRL conditions are presented on Alg. 7 and Alg. 8. These two functions are utilized by function presented above on Alg. 6.

PDDL domain actions algorithm – SWRL Precondition

```
FUNCTION swrl2pddlPreconditions(P : OWL-S Atomic Process,
            Pr : PDDL Domain Predicates) : PDDL GD
  VAR:
    Preconditions : PDDL GD (Goal Description);
  BEGIN
    FOR p TO All Process P Preconditions DO BEGIN
      add to Preconditions := swrl2atomicFormula(p, Pr);
    END;
    swrl2pddlPreconditions : Preconditions;
  END;
```

Alg. 7 Algorithm for creation of PDDL preconditions

PDDL domain actions algorithm – SWRL Effect

```
FUNCTION swrl2pddlEffects(P : OWL-S Atomic Process,
            Pr : PDDL Domain Predicates) : PDDL Effect
  VAR:
    Effects : PDDL Effect;
  BEGIN
    FOR e TO All Process P Effects DO BEGIN
      add to Effects := swrl2atomicFormula(e, Pr);
    END;
    swrl2pddlEffects : Effects;
  END;
```

Alg. 8 Algorithm for creation of PDDL effects

Both Alg. 7 and Alg. 8 utilize function presented on Alg. 9. This function is the basic function for transformation of a SWRL condition into a PDDL action precondition and/or effect. For this we need to create PDDL atomic formulas. As it has been mentioned above in section about predicates (section 3.2.4) for creation of atomic formulas we use atomic formulas' skeletons from PDDL predicates. Therefore PDDL predicates must be well-known functions.

PDDL domain actions algorithm – SWRL Condition

```
FUNCTION swrl2atomicFormula (S :
    SWRL Condition, P : PDDL Predicates):PDDL Atomic Formula
  VAR:
    CA : Set of Atoms Represented Classes;
    PA : Set of Atoms Represented Properties;
    BA : Set of BuildIn Atoms (Presently only not);
    AF : PDDL Atomic Formula;
  BEGIN
    FOR a TO All Atoms from S DO BEGIN
      IF (a is atom representing Class) DO BEGIN
        add a into CA;
      END;
      IF (a is atom representing Properties) DO BEGIN
        add a into PA;
      END;
      IF (a is buildin atom for not) DO BEGIN
        add a into BA;
      END;
    END;
    IF (PA is empty) DO BEGIN
      mapping OWL Class from CA into predicate from P;
      check the existence OWL Class in BA (if is it
      negative or not)
      create atomic formula AF;
    ELSE BEGIN
      create from atoms from PA a CA atomic formula and
      mapping this formula to predicate from P;
      check the OWL Class representing domain property for
      its existence in BA;
      add all a
      create atomic formula AF;
    END;
    swrl2atomicFormula : AF;
  END;
```

Alg. 9 Algorithm for translation of SWRL conditions

3.3. PDDL planning problem creation

The simplified planning problem structure represented by EBFN form is as follows:

```
<problem> ::= (define (problem <name>)
                      (:domain <name>)
                      [<require-def>]
                      [<object declaration>]
                      [<init>]
                      <goal>+)
```

For creation of PDDL planning problem we use initial and goal OWL ontologies. Each part of this process is described in particular subsection with proposed algorithms in pseudocode.

3.3.1. Problem and problem-domain name

Problem name is the name of the PDDL planning problem. It is a string for which holds the same conditions as for PDDL domain name (see section 3.2.1). This name may be created e.g. from the initial ontology, where it may be located in the ontology header and we use the same approach as it was presented in Alg. 1.

Problem-domain name refers to the corresponding PDDL domain. Creation of this name from OWL ontology is not so straightforward. When this name is defined in ontology header as additional ontology information, we can easily extract it. Otherwise we need to define it manually.

3.3.2. Requirements

Requirements flag may be defined as requirements, which are desired by PDDL planning problem. For these PDDL problem requirements hold the same rules as for PDDL domain requirements (see section 3.2.2).

3.3.3. Object declaration

PDDL objects represent objects, which may be substituted on predicates' parameters and thus create facts in PDDL planning problem. These objects will be retrieved from initial OWL ontology, namely from OWL individuals. The algorithm for PDDL objects creation is presented on Alg. 10. In this algorithm we need only OWL individuals, which are located in range of some OWL properties. OWL individuals located only in domain of some properties are for PDDL object creation uninteresting.

PDDL problem objects algorithm

```
FUNCTION owl2pddlObjects(O : OWL Ontology) : PDDL Objects
  VAR
    OP : Set of Concrete Object Properties from O;
    DP : Set of Concrete Data Properties from O;
    Objects : Set of PDDL Objects;
  BEGIN
    FOR p TO OP DO BEGIN
      VAR
        i : OWL Class Individual;
      i := range value of property p;
      IF (Objects don't contain i) THEN BEGIN
        add i into Objects together with PDDL type
          represented by corresponding OWL Class for i;
      END;
    END;
    FOR v TO DP DO BEGIN
      VAR
        i : XSD Datatype Value;
      i := range value of property v;
      IF (Objects don't contain i) THEN BEGIN
        add i into Objects together with PDDL type
          represented by corresponding Datatype for I;
      END;
    END;
    ordering Objects by types;
    owl2pddlObjetcs : MO;
  END;
```

Alg. 10 Algorithm for creation of PDDL problem objects

3.3.4. Initial state

Initial state of PDDL problem is represented as a set of PDDL literals. Each literal is either positive or negative atomic formula. Atomic formula consists of predicate name and from a list of objects. Therefore we need for creation of an atomic formula a PDDL predicate, which is represented by atomic formula skeleton. Predicate parameters are replaced by concrete objects. By mapping concrete object into PDDL predicate we create a fact, which may represent the initial state.

For PDDL planning problem initial state we will use initial OWL ontology, and this process is showed on Alg. 11. At the beginning of this process we need to know all relevant OWL properties (both object and data). For every OWL individual we need to obtain all OWL properties, where this individual occurs in the domain. Afterwards we map this information to concrete predicate

from PDDL predicates. If the OWL individual is not located in any OWL property, we may map this OWL individual directly on PDDL predicate.

3.3.5. Goal state

Likewise initial state also the goal state of PDDL problem is represented as a set of PDDL literals. PDDL goal is in the PDDL problem structure represented as PDDL GD (Goal Description). Therefore there may exist quantifiers in goal state descriptions (and, or, not, imply, exists, forall).

PDDL problem init state algorithm

```
FUNCTION owl2pddlInitState(O : OWL Ontology)
            :PDDL Init State
  VAR
    OP : Set of Concrete Object Properties from O;
    DP : Set of Concrete Data Properties from O;
    IS : Set of Literals – Init State;
    I : Set of Individuals;
    I1 : Helping Set of Individuals;
    PR : Set of Domain Predicates;
  BEGIN
    PR := owl2pddlPredicates(O);
    FOR v TO OP DO BEGIN
      VAR
        i : OWL Class Individual;
      i := value of property v domain;
      IF (I don't contain i) THEN BEGIN
        add i into I;
      END;
    END;
    FOR v TO DP DO BEGIN
      VAR
        i : OWL Class Individual;
      i := value of property v domain;
      IF (I don't contain i) THEN BEGIN
        add i into I;
      END;
    END;
    FOR i TO I DO BEGIN
      VAR
        a : PDDL Atomic Formula;
        l : PDDL Literal;
      a.Predicate := PDDL type of i;
      a.Parameters := {all properties where i is situated in
        their domains};
      mapping a to concrete PDDL predicate from PR;
      l := a;
      add l into IS;
    END;
    I1 is a set of Individuals, which are not located in any
      property domain;
    FOR i TO I1 DO BEGIN
      mapping i to concrete PDDL predicate from PR;
      l := a;
      add l into IS;
    END;
    owl2pddlInitState : IS;
  END;
```

Alg. 11 Algorithm for creation of PDDL problem initial state

In current version of our AWSC system we consider only two of these quantifiers, namely 'and' and 'not'. They may be several methods for PDDL problem goal creation.

One of usable method is presented in our algorithm on Alg. 12. In this method we consider existence of initial and goal ontologies. Both from these two ontologies contain information for creation of PDDL literals. The literals obtained from initial ontology were used for creation of initial state (see previous subsection 3.3.4). Now we discover goal OWL ontology and find the new literals regarding the initial OWL ontology. From this literals we create the PDDL goal state.

PDDL problem goal state algorithm

```
FUNCTION owl2pddlGoalState(OInit , OGoal : OWL Ontology)
            :PDDL Goal State
  VAR
    IS : Set of Literals – Initial State;
    GS : Set of Literals – Goal State;
    G : PDDL Goal State;
    X : Helping Set of Literals;
  BEGIN
    IS := owl2pddlInitState(OInit);
    GS := owl2pddlGoalState(OGoal);
    X := Literals from GS, which are not situated in IS;

    G := Conjunction of Literals from X;
    owl2pddlGoalState := G;
  END;
```

Alg. 12 PDDL goal state

3.4. PDDL planning task creation

Alg. 13 presents the whole process for creation of a PDDL planning problem. In this program we are using algorithms (functions), which were described in previous sections.

PDDL planning task program

```
PROGRAM PDDLPlanningTaskCreation
  VAR
    O : Domain OWL Ontology;
    initO : Initial State Ontology;
    goalO : Goal State Ontology;
    owlS [Owls1, Owls2, ... , Owlsn] : Set of available
      OWL-S Web Services Descriptions;
    domain : PDDL Planning Domain;
    problem : PDDL Planning Problem;
  BEGIN
    domain.Name := owl2pddlDomainName(O);
    domain.Requirements := {:strips, :typing};
    domain.Types := owl2pddlTypes(O);
    domain.Predicates := owl2pddlPredicates(O);
    FOR p TO owlS DO BEGIN
      VAR
        A : PDDL action;
      A := owls2pddlActions(p);
      add A into domain.Actions;
    END;

    problem.Name := owl2pddlProblemName(initO);
    problem.DomainName := owl2pddlDomainName(O)
    problem.Requirements := {:strips, :typing};;
    problem.Objects := owl2pddlObjects(initO);
    problem.Init := owl2pddlInitState(initO);
    problem.Goal := owl2pddlGoalState(initO, goalO);
  END;
```

Alg. 13 Algorithm for PDDL planning task creation

3.4.1. Examples

```
(define
  (domain travel)
  (:requirements  :strips :typing)
  (:types road thing place at - object
          person vehicle - thing)
  (:predicates
    (at   ?has_place - place  ?has_thing - thing)
    (road   ?has_place_to ?has_place_from - place)
    (person   ?person - person)
    (vehicle   ?vehicle - vehicle)
    (thing   ?thing - thing)
    (place   ?place - place)
  )
  (:action Driving
    :parameters
      (?Thing - string
       ?FromPlace ?ToPlace - place)
    :precondition
      (and (road ?FromPlace ?ToPlace)
        (at ?FromPlace ?Thing))
    :effect
      (and (at ?ToPlace ?Thing)
        (not (at ?FromPlace ?Thing)))
  )
)
```

Fig. 3 PDDL domain example

On Fig. 3 can be seen a simple example of a PDDL planning domain. This example has been derived by transformation of the semantic description for primary composition problem definition in OWL and OWL-S. This problem was next transformed into PDDL planning problem (Fig. 4) using proposed algorithm (see Alg. 13).

```
(define
  (problem pb1)
  (:domain travel)
  (:requirements  :strips :typing)
  (:objects   a b c d - place  mazda - vehicle
              john - person)
  (:init
    (place a)
    (place b)
    (place c)
    (place d)
    (at b mazda)
    (at a palo)
    (road a b)
    (road b a)
    (road b c)
    (road c b)
    (vehicle mazda)
    (road c d)
    (road d c)
    (person palo)
  )
  (:goal
    (and (at d mazda)
      (at d palo))
  )
)
```

Fig. 4 PDDL problem example

4. RELATED WORK

Most of the systems and works, which deal with (semi)automatic web service composition, make use of existing automated planning systems. It is due to relatively straightforward and suitable mapping between planning problem and web service composition problem. In our work we introduced a possibility of mapping semantically represented web service composition problem into a planning problem represented in the PDDL language.

Table 1 Comparison of existing systems (AC = automated composition, SAC = semi AC)

System	WS standards	Internal problem represent.	System type	Dynamic planning
WS prolog	OWL-S, WSDL	logical program	SAC	no
SHOP2 system	OWL-S, WSDL	PDDL 2.1	AC	no
OWLSX plan	OWL, OWL-S, WSDL	PDDL 2.1	AC	yes
WSPlan	WSDL	PDDL 1.2	AC with manual WS selection	no
SEMCO -WS	OWL, OWL-S, WSDL	GWorkflowDL	SAC with workflow	no

Before we designed our AWSC system, we analysed a couple of already existing systems and proposals. Summary of the analysed systems can be seen in Tab. 1.

WS Prolog [14] is a system for web service composition, which uses logical programming elements. As already suggests the name of this system, it is based on Prolog programming language.

SHOP2 is domain independent HTN (Hierarchical Task Network) planning system, which won one of the four main prices on international planning competition in 2002[3]. HTN is a planning method, which is focused on task decomposition in order to create a plan. Tasks decomposition is performed until all tasks aren't decomposed into primitive tasks. On given planning system authors in created system for AWSC [7]. In this system transformation of OWL-S WS descriptions into HTN planning domain is performed. In our work we are inspired by some of the algorithms from this system, but we use different target language.

OWLSXplan [9] is a tool developed for automatic web service composition by using artificial planning method. This system realises conversion of WS described by OWL-S version 1.1 into equivalent planning problem described by PDDL 2.1 language. After conversion into planning domain the planning itself is performed by XPlan planner. XPlan planner is based on Fast Forward planner improved by HTN planning. One of our innovations with respect to this system is the use of SWRL for conditions specification.

Authors in [15] dealt with web service composition only with using WSDL descriptions. They proposed a system, which goal is to provide WSDL described web service composition through artificial intelligence planner. The problem of composition starts with user part, in which users must select initial WS. Therefore this system is semiautomatic.

SEMCO-WS project [16] dealt with processing web and grid services composition. A service composition is performed through semantic WS and data annotations and afterwards executable workflows are created. Petri Nets are used for representation of workflows and the mechanism for web service composition is completely different than in our approach.

5. CONCLUSIONS

In this article we presented our proposal for a AWSC system. The main focus of our activity was to design and implement suitable transformation process from external composition problem specification into an internal specification. For external specification we used knowledge approaches (OWL and OWL-S ontology) and for internal specification the PDDL planning problem specification was used. We proposed several original algorithms, presented partially in this article, which were implemented and experimentally verified.

ACKNOWLEDGMENTS

This work was supported by the Slovak Research and Development Agency under the contract No. VMSP-P-

[3] IPC – International Planning Competition, http://planning.cis.strath.ac.uk/competition/

0048-09 (30%); the Slovak Grant Agency of Ministry of Education and Academy of Science of the Slovak Republic under grants No. 1/0042/10 (30%).

This work is also the result of the project implementation Development of the Center of Information and Communication Technologies for Knowledge Systems (project number: 26220120030) supported by the Research & Development Operational Program funded by the ERDF (40%).

REFERENCES

[1] BECHHOFER, S. et al.: Owl Web Ontology Language Reference, W3C Proposed Recommendation. http://www.w3.org/TR/owl-ref/, 2004.

[2] MARTIN, D. et al.: OWL-S: Semantic Markup for Web Services, W3C Member Submission, http://www.w3.org/Submission/OWL-S/, 2004.

[3] RAO, J. – SU, X.: A Survey of Automated Web Service Composition Methods. In Proceedings of the First International Workshop on Semantic Web Services and Web Process Composition, SWSWPC 2004, San Diego, California, USA, 2004.

[4] NILSSON, N. J. – FIKES, R. E.: STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving, Artificial Intelligence, 2(3):189-208, 1971.

[5] HOFFMAN, J.: FF: The Fast Forward Planning System, AI Magazine, Vol. 22 #3, pp. 57–62, 2001

[6] REITER, R.: On knowledge-based programming with sensing in the situation calculus. ACM Trans. Comput. Logic 2, 4 (Oct. 2001), 433-457, 2001.

[7] WU, D. – SIRIN, E. – HENDLER, J. – NAU, D. – PARSIA, B.: Automatic Web services composition using SHOP2, In Workshop on Planning for Web Services, 2003.

[8] PEER, J.: Web Service Composition as AI Planning - A Survey, Dissertation, University of St. Gallen, Switzerland, 2005.

[9] KLUSCH, M. – GERBER, A. – SCHMIDT, M.: Semantic Web Service Composition Planning with OWLS-Xplan. 1st Intl. AAAI Fall Symposium on Agents and the Semantic Web, Arlington VA, USA, 2005.

[10] GHALLAB, M. et al.: PDDL – The Planning Domain Definition Language, Version 1.2. Yale Center for Computational Vision and Control, Tech Report CVC TR-98-003/DCS TR-1165, October, 1998.

[11] YANG, B. – QIN, Z.: Composing Semantic Web Services with PDDL. Information Technology Journal 9 (1), ISSN 1812-5638, 48-54, 2010.

[12] ĎURČÍK, Z.: Webové služby - kompozícia umelo inteligentnými metódami. In: Znalosti 2010, Jindřichuv Hradec, 2010, ISBN 978-80-245-1636-3, pp. 203–206.

[13] ĎURČÍK, Z.: Translation Semantic Web Services Descriptions into Planning Problem. In: SCYR 2010: 10th Scientific Conference of Young Researchers of FEI Technical University of Košice: proceedings from conference: May 19th, 2010, Košice, Slovakia. - Košice: FEI TU, 2010, ISBN 978-80-553-0423-6, pp. 189–191.

[14] SIRIN, E. – HENDLER, J. – BIJAN, P.: Semi-automatic composition of web services using semantic descriptions in Web services: Modeling, Architecture and Infrastructure workshop in ICEIS 2003, Angers, France, April 2003.

[15] PEER, J.: A PDDL Based Tool for Automatic Web Service Composition. Lecture Notes in Computer Science, Springer Berlin / Heidelberg. ISBN 978-3-540-22961-2, pp. 149–163, September 2004.

[16] HABALA, O. – PARALIČ, M. – ROZINAJOVÁ, V. – BARTALOS, P.: Semantically-Aided Data-Aware Service Workflow Composition. SOFSEM '09: Proceedings of the 35th Conference on Current Trends in Theory and Practice of Computer Science, ISBN 978-3-540-95890-1, pp. 317–328, 2009.

Received November 25, 2010, accepted April 10, 2011

BIOGRAPHIES

Zoltán Ďurčík graduated (MSc) with distinction at the department of Cybernetics and Artificial Intelligence of the Faculty of Electrical Engineering and Informatics at Technical University in Košice. He finished his PhD. study at the same department in 2010 and this article describes main aspects of his dissertation thesis. His scientific research was focused on semantic web and automated web service composition. Currently he is working in IT department of an electricity providing company in Slovakia.

Ján Paralič received his Ph.D. degree in 1998 at the Technical University in Košice. Since 2004, he is associate professor at the Department of Cybernetics and Informatics, Technical University in Košice and since 2005 also head of the Centre for Information Technologies at the same university. He (co-)authored four books; (co-)edited 12 proceedings from various international workshops and conferences and published more than 90 scientific papers. His research interests currently are in the areas of knowledge discovery, text mining, semantic technologies, and knowledge management.