Int J Parallel Prog (2015) 43:339–358
DOI 10.1007/s10766-013-0279-0

Automatic Composition of Heterogeneous Models Based on Semantic Web Services

Hui Huang · Ligang He · Xueguang Chen · Minghui Yu · Zhiwu Wang

Received: 26 March 2013 / Accepted: 4 October 2013 / Published online: 13 October 2013

Abstract As an important function of a distributed decision support system, model composition aims to aggregate model functions to solve complex decision problems. Most existing methods on model composition only apply to the models which have the same type of input and output data so that they can be linked together directly. Those methods are inadequate for the heterogeneous models, since a heterogeneous model may have different types of input and output data that are represented in either qualitative or quantitative manner. This paper aims to address the problem of heterogeneous model composition by employing the techniques based on semantic web services and artificial intelligence planning. In this paper, the heterogeneous model composition problem is converted to the problem of planning in nondeterministic domains under partial observability. An automatic composition method is presented to generate the composite model based on the *planning as model checking* technique. The experiment results are also presented in this paper to show the feasibility and capability of our approach in dealing with the complex problems involving heterogeneous models.

Keywords Decision support systems · Model management · Distributed model composition · Planning as model checking · Semantic web service

H. Huang (✉)· X. Chen · M. Yu
Department of Control Science and Engineering, Huazhong University of Science and Technology, Wuhan 430074, China
e-mail: shocking.easy@163.com

L. He
Department of Computer Science, University of Warwick, Coventry CV4 7AL, UK

Z. Wang
Department of Computer Science and Engineering, Henan Institute of Engineering, Zhengzhou 451191, China

1 Introduction

A decision support system (DSS), which assists users in decision making, normally includes multiple decision models. The models encapsulate the knowledge about analyzing and solving the complex problems, and therefore should be managed, shared and re-used efficiently. Currently, the models are typically distributed in different geographical locations with different supporting platforms interconnected through the communication networks. Decision-making now demands remote access to these models/resources without any obstacles. Distributed model management systems (DMMS) can be used as a solution to this problem. DMMS supports the model management (MM) functionalities including representation, manipulation, integration, composition and execution of models on the web (these models may be developed by the service providers from different organizations). Therefore, the decision models can be fully utilized remotely through DMMS.

One of the most important features of MM is model composition, which refers to composing a sequence of models (i.e., create a composite model) in response to user queries [1]. Composite models provide the aggregate functionalities which cannot be offered by each individual model. However, only a few of research works attempt to address model composition in distributed environments [2].

Web services are the loosely coupled, reusable software components that semantically encapsulate discrete functionality. They are distributed and programmatically accessible through the standard Internet protocols. The features of the web service technology, such as its support for the distributed application development and delivery, reliance on syntactic standards for inter-service communication, support for dynamic service discovery and the potential to support dynamic composability, make it an attractive alternative technology for developing a distributed model management framework [3].

Over the past decade, researchers have developed many approaches on model/service composition such as those presented in [4–9]. Most of the existing methods on web service composition are aimed at homogeneous models/services, that is, they assume that the inputs and outputs of each models/services being composed have the same data type (e.g., numeric) so that the individual models can be linked together directly. However, from the DSS perspective, the models can be either quantitative or qualitative (termed heterogeneous models in this paper), and they populate in the model base simultaneously. Heterogeneous models cannot be directly composed using the existing methods because the inputs and outputs have different data types. It still remains unresolved to compose heterogeneous models, especially in the distributed settings. In this paper, we observed that the heterogeneous models may have the ties among the heterogeneous models, for example, the ties may exist through the "precondition" segment, i.e., the output of one model determines the satisfaction of the precondition of another model. This insight will be utilized to compose the heterogeneous models.

A motivating example of earthquake prediction is presented below to illustrate the challenges involved in heterogeneous model compositions.

Earthquake is a typical type of natural hazards. It usually occurs without an explicit warning and may cause serious injuries or loss of human lives and properties. To effec-

tively mitigate the damage from an earthquake, seismologists have spent many years on predicting seismic events, although this problem remains a subject of numerous controversial discussions and debates. Earthquake prediction, which aims to specify three elements, namely, the occurrence time, the epicenter, and the seismic magnitude, is one of the most important unsolved problems in the field of seismology.

Researchers have tried to predict an impending earthquake through different phenomenon, such as seismicity patterns, electromagnetic fields, weather conditions and unusual clouds, radon or hydrogen gas content of soil or ground water, water level in wells, and animal behaviors, etc [10,11]. Quantitative and qualitative models have already been constructed to represent these connections.

An idea of performing rapid earthquake predictions in a specific region is to compose and remotely access multiple prediction models and other auxiliary models through web services, based on the observational phenomena and data we have gathered. For example, Geng [12] proposed a empirical formula to calculate the magnitude of an impending earthquake, taking as input the duration of the drought and the area of the anomalous field of interest. The empirical formula is listed below:

$$\mathrm{Ms} = 1.5 \log \mathrm{T} + 3.5 \log \mathrm{S} + 0.5 \tag{1}$$

Where Ms is the magnitude, T is the duration of the drought and S is the area of the studied field. This empirical formula is a quantitative model. Formula (1) is called the Magnitude-Compute model in this paper.

Moreover, according to [12], a precondition must be satisfied before the Magnitude-Compute model is invoked, i.e., the drought degree of the anomalous field must be serious. This means that in order to apply the model to calculate the magnitude, the drought level must be determined first using a method on meteorology. According to [12], the drought degrees can be determined using another model, which is called the Drought-Degree model in this paper. The model consists of three rules which are composed using the if-then structure. The model is described in detail in Table 1. Different from Formula (1), this is a qualitative model, which takes a numeric number as input (i.e., the value of PAAP) and outputs a text (i.e., "mild", "moderate" and "serious").

Therefore, predicting the earthquake magnitude can be converted to composing the qualitative Drought-Degree model and the quantitative Magnitude-Compute model in sequence. The composite model will require three input date: PAAP for the Drought-Degree model, and duration and area for the Magnitude-Compute model. However, the composition cannot be carried out by linking these two methods directly, since the data type of the output of the Drought-Degree model is string, while the data type

Table 1 The Drought-Degree model

If the percentage of average annual precipitation (PAAP) is larger than 89 %, then the drought degree is mild
If the PAAP is larger than 79 % and not more than 89 %, then the drought degree is moderate
If the PAAP is not more than 79 %, then the drought degree is serious

of the input of the Magnitude-Compute model is numeric. To date, composing such heterogeneous models remains an unsolved problem.

Therefore, this paper aims to address the composition problem for heterogeneous models. This paper proposes a method to represent and enable the composition of heterogeneous models in response to complex user requests. Moreover, the service-oriented techniques, which are based on semantic web and artificial intelligence planning, are employed to perform actual model composition. A set of experiments have been conducted in this paper to verify the effectiveness of the proposed method in terms of the composition speed.

The rest of this paper is organized as follows: Sect. 2 reviews the existing research work in the areas of distributed model composition and the AI planning, which are the main techniques used in this paper. Section 3 proposes the representation of qualitative and quantitative models, respectively. Section 4 presents the composition of model services. Experiments are conducted in Sect. 5 to evaluate the performance of the proposed composition method in terms of the composition speed. Finally, this paper is concluded and the future work is discussed in Sect. 6.

2 Background and Related Work

This section discusses the related work in the following four areas: distributed model composition, semantic Web and AI planning Business rule engine.

2.1 Distributed Model Composition

Model composition is an important component of model management, and it is invoked when the user requests cannot be fulfilled by a single model from the model base. An example of model composition is to link together a demand forecasting model and a production scheduling model, i.e., the demand forecast generated by the former model is used as a parameter in the latter model [1].

Due to the inherent complexity of model composition, there are only a few research works attempting to address the problem, especially in distributed settings. Chari [4] addressed the problem of model composition when data sources (models and data) are distributed among multiple sites. Their work focuses on managing the search space of model composition by using logical constraints called filter spaces. Madhusudan and Uttamsingh [5] presented a novel declarative approach to facilitating dynamic and scalable web service composition [called Integrated Service Planning and Execution (ISP&E)] based on AI planning techniques. Madhusuda [3] then proposed a framework for model management based on the previously proposed ISP&E to support various activities in the life cycle of model management. Deokar and El-Gayar [2] presented an architecture based on semantic web services for model management, and solved the problem of model composition using AI planning and semantic web technologies.

However, the main drawback of the work in the literature is the implicit assumption that the elementary models/services in the distributed model/service library are homogeneous, meaning that the output of the preceding model/service can be directly accepted by the subsequent model/service. However, a DMMS often contains both

quantitative and qualitative models. The aforementioned methods cannot be directly employed to compose such heterogeneous models.

2.2 Semantic Web Standards

The vision of Semantic Web services is to describe Web services' properties, capabilities, interfaces, and effects in an unambiguous, computer-interpretable language [13,14] and promote the realization of web service discovery and composition.

Ontologies are the content theories about the sorts of objects, properties of objects, and the possible relations between objects in a specified domain of knowledge. They provide potential terms for describing our knowledge about the domain [15]. The Web Ontology Language (OWL) is a semantic markup language for publishing and sharing ontologies on the World Wide Web [16]. Protégé[1] is one of the most popular ontology modeling tools.

OWL-S is an ontology built on top of OWL by the DARPA DAML program. It replaces the former DAML-S ontology. OWL-S is an ontology, within the OWL-based framework of the Semantic Web, for describing Semantic Web Services. It will enable users and software agents to automatically discover, invoke, compose, and monitor Web resources that offer services, subject to specified constraints [17].

2.3 Planning as Model Checking

Planning as model checking offers a formal method of planning under uncertainty, which can manage nondeterminism and partial observability. It was first introduced in [18,19]. The nondeterminism may result from the incomplete information about the initial condition or the uncertain effects of actions. For example, in the example of earthquake prediction discussed in Sect. 1, the internal logic of prediction model may not always be visible, or may be changed occasionally. Therefore the output of the model is non-deterministic (which may take one of the three possible results), even if we know the input of the drought-Degree model. Partial observability concerns the limits on information available at run-time. That is, some applications may involve variables that are only observable in certain states, or only after some sensing actions have been executed.

At an early stage of the studies on Planning as model checking, researchers formally characterize different planning problems into three types: weak planning, strong planning, and strong cyclic planning. Strong plans are guaranteed to achieve the goal for all possible executions. Weak plans have a chance of success, in that some of its executions achieve the goal. Strong cyclic plans are guaranteed to reach the goal under the supposition that executions will eventually exit any loops in the system. Strong cyclic plans reach the goal with an iterative trial-and-error strategy: executions always have a possibility of being terminated. When they were, it means that they have achieved the goal [20]. Another type of planning problems is conformant plan-

[1] The Protégé Ontology Editor and Knowledge Acquisition System: http://protege.stanford.edu/.

ning (or called planning under null observability), which is the problem of finding a sequence of actions that is guaranteed to achieve the goal for any possible initial state and nondeterministic behavior of the planning domain [21], that is, in the case of null observability where no observations are available at all at run-time.

A system called the model based planner (MBP) [22] is developed for planning in non-deterministic domains. It can generate plans automatically to solve various planning problems, like conformant planning, planning under partial observability, and planning for temporally extended goals.

Indeed, researchers have conducted considerable studies on the characteristics of Web service composition. They found that this problem can be adequately solved by the Planning as model checking technology [23–25]. However, the web service compositions in these works still need much manual intervention.

3 Representing Heterogeneous Models as Semantic Web Services

In terms of the analysis mode, models for decision making can be generally divided into quantitative and qualitative models. As mentioned in Sect. 1, a model of loosely coupled components delivering specific functionality can be conceptualized as a service [2]. However, different types of models have different representations. Although a number of model representation approaches have been proposed, in this paper we view a "model" as a computer-executable procedure that may require data inputs and produce outputs. This section discusses the syntax and semantic representation of the quantitative and qualitative models.

Generally speaking, models may be represented at different levels of abstraction: modeling paradigm, model schema and model instance [2], as shown in Fig. 1. Level 1 is the highest level of abstraction that provides the concepts and relationship that can be used to represent both the model schema and the model instance, which can be seen as the ontology of the modeling domain, as discussed in section 0. Level 2 is a class of model schema, whose parameters are known but not yet instantiated (such as a computer program). This level is represented as a model service that receives user inputs and is instantiated as an executable model (such as the running of a computer program after giving a set of input) to solve a particular decision problem in an organization.

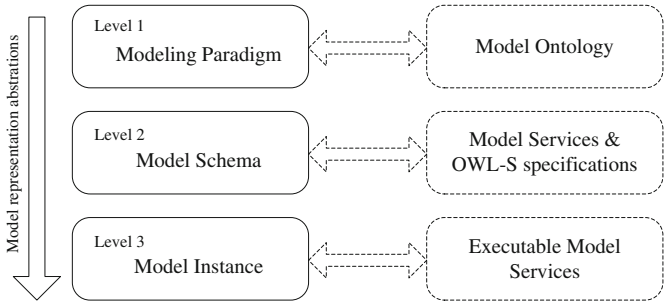

Fig. 1 Model representation abstractions

```
public class Magnitude_Compute
{
    public double
    magnitude_compute(double duration, double area)
{
    return 1.5*Math.log10(duration)+3.5*Math.log10(area)+0.5;
}}
```

Fig. 2 The Java code of the Magnitude-Compute model

The executable model is the model instance in Level 3. Besides, the semantic markup of every model service must be encoded to declare the constraints ("preconditions" and "effects") under which the service are invoked. Level 2 will be further discussed in section 0 and 0.

3.1 Representation of Quantitative Models

Most DSS models are mathematical models which have a target output, a set of inputs, and operations for converting inputs to outputs [26].

Generally, the quantitative models such as mathematical models can be directly encoded by the advanced programming languages, such as C# and Java, and then exposed as web services (called model services) using the third-party tools. The advanced programming languages themselves embed abundant mathematical functions, which can represent sophisticated mathematical formulas.

Recall the formula of calculating M_s in Sect. 1 in the example of earthquake prediction. It can be encoded in JAVA as in Fig. 2.

The code listed in Fig. 2 implements the program function corresponding to the Magnitude-Compute model in Formula (1). In the code, the function "Magnitude_Compute" has specified the data types of its inputs and outputs (model schema), but the inputs and output can take different data values when the function is running (model instance).

With the assistance of Eclipse[2] Integrated Development Environment (IDE), Apache Tomcat[3], and the Apache Axis2,[4] we can easily encode the web service description of the Magnitude_Compute model, which is shown in Fig. 3.

Figure 3 presents the WSDL specification for the Magnitude-Compute model service, omitting technical details irrelevant to our discussion. The input and output parameters ("drought_duration", "drought_area" and "magnitude") of the WSDL are defined in the ontology shown in Fig. 5, which will be discussed in section 0.

In addition to describing a model service using WSDL, we also need to describe the way in which a client may interact with the model service, especially the preconditions under which the model service can be invoked and also the effect after running the service. These are typically represented as a process model through semantic representations, usually in the form of OWL-S specifications. This capability of the OWL-S

[2] Eclipse platform: http://www.eclipse.org/.

[3] Apache Tomcat: http://tomcat.apache.org/.

[4] Apache Axis2: http://axis.apache.org/axis2/java/core/.

```
.....
<wsdl:types>
.....
<xs:element name="magnitude_compute">
<xs:complexType>
<xs:sequence>
<xs:element minOccurs="0" name="drought_duration" type="xs:double"/>
<xs:element minOccurs="0" name="drought_area" type="xs:double"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="magnitude_computeResponse">
<xs:complexType>
<xs:sequence>
<xs:element minOccurs="0" name="magnitude" type="xs:double"/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>
</wsdl:types>
.....
```

Fig. 3 The WSDL document for the Magnitude-Compute model service

```
.....
<process:hasPrecondition>
<process:Condition>
<drs:Atomic_Formula>
<rdf:subject rdf:resource="http://www.hust.edu.cn/
             earthquake.owl#drought_degree"/>
<rdf:predicate rdf:resource="http://www.hust.edu.cn/
              earthquake.owl#is_serious_drought"/>
</drs: Atomic_Formula>
</process: Condition>
</process: hasPrecondition>
.....
```

Fig. 4 The process model for the Magnitude-Compute model service

specification is extremely useful for heterogeneous model compositions, as we will discuss in the Sect. 4.

The process model of the exemplar Magnitude-Compute model service is shown in Fig. 4, where the "hasPrecondition" segment represents the precondition that must be satisfied first before the Magnitude-Compute model service is invoked, and there is no effect after running the service. Again, the code in Fig. 4 omits technical details (such as the "message" and "binding" definition) irrelevant to our discussion.

3.1.1 Representation of Qualitative Models

Qualitative models are typically represented using a descriptive language, usually in the form of if-then, like the Drought-Degree model in Table 1.

In this paper, we use business rules to represent the qualitative models. The reason for this is because of their support for if-then statements, although they are not designed for representing qualitative models.

Table 2 The decision table for Drought-Degree model

	The avg precipitation percentage		The drought degree
	Min	Max	
1	0	0.79	Serious
2	0.79	0.89	Moderate
3	0.89	1	Mild

Several rule engines has been developed by academic and business communities. To the best of our knowledge, the WebSphere ILOG JRules[5] is the only rule engine to date that provides support for Service-Oriented Architectures. Rules can be expressed using business rules, decision tables, decision trees and technical rules in JRules. In this paper, we use JRules to represent the Drought-Degree model in the form of decision table (shown in Table 2), which provides a concise view of a set of business rules in the form of a spreadsheet. Decision tables are composed of rows and columns. Each row corresponds to a single rule, while the columns define the conditions and actions of the rules.

The condition column is on the left, while the action column is on the right. The actions of a given rule are performed when its conditions are met. For example, the rule corresponding to the second row in Table 2 reads as follows:

If the avg precipitation percentage (i.e., PAAP) is between 0.79 and 0.89, then the drought degree is set to be moderate.

Three rules in the decision table in Table 2 are the assertions about the value of "drought degree" depending on the value of PAAP. In other words, the actual value of "drought degree" is unknown until the PAAP is determined explicitly.

Taking advantage of the Hosted Transparent Decision Service (HTDS) function in JRules, the decision table for the Drought-Degree model can be conveniently exposed as a decision service without detailed programming. HTDS is technically a Web service with management capabilities that use JMX MBeans. It drives rule execution and enables users to access Rule Execution Server through a Web service using HTTP and XML data formats (SOAP). It is referred to as a transparent decision service because the users do not need to know how the service is implemented.

The WSDL document and process model of OWL-S specification for the Drought-Degree model service are similar to that of the Magnitude-Compute model service as shown in Fig. 4, apart from the "hasPrecondition" segment. Therefore, its code segment is omitted in this paper.

3.2 Semantic Representation of Elementary Model Services

The benefit of building the model ontology is to avoid the ambiguity between concepts, e.g., the same words that have different meanings, or different words that have the same meanings.

[5] WebSphere ILOG JRules: http://www-01.ibm.com/software/integration/business-rule-management/jrules/.

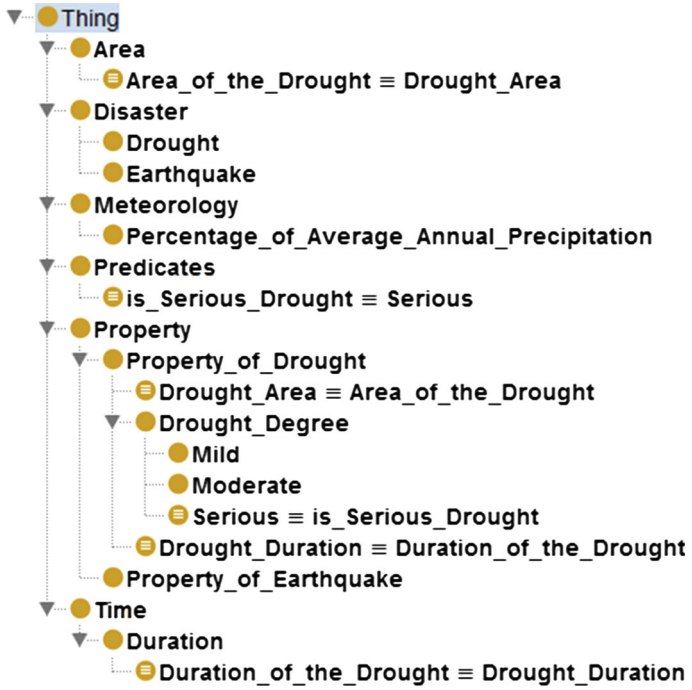

Fig. 5 The model ontology for the two model services

As an example, a model ontology using OWL is built to capture the concepts and relationships of the terminologies used in the WSDL and OWL-S specification of the two model services in the section 0 and 0.

Figure 5 presents the model ontology of our example scenario. It should be noted that the "Predicates" class in the model ontology involves the "precondition" fragment in the OWL-S specification of the Magnitude-Compute model service. Another thing that needs to be noted is the subclasses of the "Drought-Degree" class: Serious, Moderate, and Mild, which can be seen as the range of the output value of the Drought-Degree model service.

4 Composition of Model Services

In this paper, the heterogeneous model composition is modeled as the problem of planning under partial observability. This is because at the execution time of the composed model services, (a) the satisfaction of the preconditions in some model services cannot be fully observed, and (b) some model services may have several possible outcomes. The uncertainties above can be uniquely determined by performing some "sense" operations, e.g., capturing and analyzing the return value of some model services. The problem of planning under partial observability has been shown to be difficult, both theoretically and experimentally. Compared to planning under full observability,

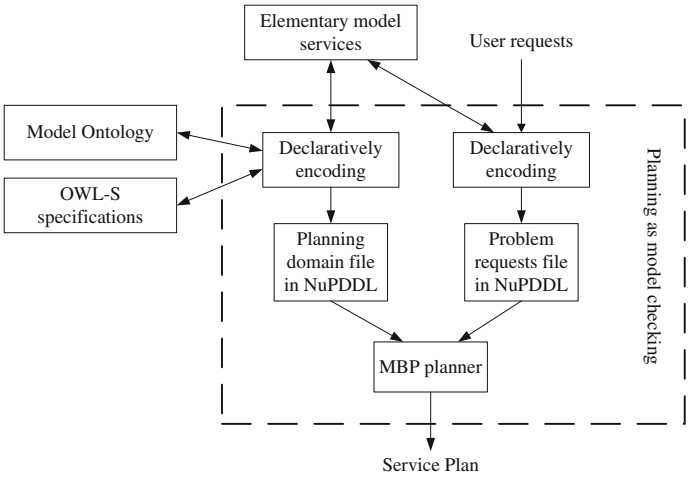

Fig. 6 The process of generating the service plan

planning under partial observability must deal with uncertainty of the states in which the actions will be executed. This makes the search space no longer the set of states of the domain, but its powerset. Compared to the case of null observability, which aims to find a sequence of satisfactory actions for any possible initial state and nondeterministic behavior of the planning domain, planning under partial observability are no longer sequential (i.e., actions in a plan are executed in sequence), but conditional (e.g., using the "switch-case" statement), in order to represent a conditional course depending on the observations performed at execution time [27]. An AI planning technique, called planning as model checking, is employed in this paper to solve the problem of planning under partial observability, because the model checking technique proves to have prominent performance on partial observability and nondeterministic planning problems [27].

A composite model service can be represented as a service plan, which is a sequence of interleaved model services in a particular decision-making situation. It typically involves three key steps to utilize the AI planning techniques such as planning as model checking to generate a service plan: (1) declaratively encoding elementary model services, (2) modeling problem requests, and (3) generating the service plan with the aid of MBP. Figure 6 shows the process of generating the service plan.

As shown in the dashed box in Fig. 6, the elementary model services (together with the Model Ontology and the OWL-S specification associated with these elementary model services) are first encoded declaratively (corresponding to Step 1 discussed above). The user requests are also declaratively encoded (Step 2). The generated files in these two steps are composed in Nupddl (which will be discussed in Sects. 4.1, 4.2). The MBP planner will then take these two Nupddl files as input to generate the service plan (Step 3). The rest of this section describes all these three steps in detail and illustrates how to generate a service plan for heterogeneous models.

Table 3 Mapping between the main elements in WSDL/OWL-S and Nupddl

Elements in WSDL and OWL-S	Elements in Nupddl
wsdl: input	predicates
wsdl: output	predicates
process: hasPrecondition	(:types constraint)
wsdl: service	(:action XX)

Table 4 Mapping between input and output parameters in WSDL and the predicates in Nupddl

Input and output parameters in WSDL	Predicates in Nupddl
drought_duration	(agentKnowTheValueOf_drought_duration)
drought_area	(agentKnowTheValueOf_drought_area)
drought_degree	(agentKnowTheValueOf_drought_degree)
Magnitude	(agentKnowTheValueOf_magnitude)
avg_precipitation_percentage	(agentKnowTheValueOf_avg_precipitation_percentage)
epicenter_area	(agentKnowTheValueOf_epicenter_area)

4.1 Declaratively Encoding of Elementary Model Services

MBP supports an extension of PDDL2.1 [28] named NuPDDL[6] that allows to model uncertainty in the initial situation, nondeterministic action effects, and partial observability in the domain. The declarative language helps describe elementary model services using the methods and operations that fit into MBP. Before giving the comprehensive declarative descriptions of the two models, we first discuss the mappings among the elements in WDSL, OWL-S and NuPDDL descriptions.

In WSDL and OWL-S documents, the main elements we will use are the inputs, outputs and preconditions. As shown in the last row in Table 3, the fundamental idea is to map model services to actions in the planning domain, which is necessary when utilizing the planning as model checking technique in service composition. The inputs and outputs are mapped to the "predicates", while preconditions are mapped to the "types". The "predicates" are the assertions about a domain and can be judged by true or false, and the "types" are the classes of the parameters that appear in actions. In our example, input and output parameters in WSDL are mapped to different "predicates" as shown in Table 4.

In our example, the precondition of invoking the Magnitude-Compute model service is that "the drought_degree" must be "serious", in other words, the precondition is the constraint on the "value" of "the drought_degree". So the "constraint" segment in the 4th row (i.e., process:hasPrecondition) in Table 3 can be instantiated as a constraint on "value". The value of "the drought_degree" can be mild, moderate or serious, and the "functions" element in NuPDDL can be used to determine the final value of a variable. In this paper, therefore, the preconditions in the OWL-S specifications of the

[6] NuPDDL: http://mbp.fbk.eu/NuPDDL.html.

Table 5 Mapping of the preconditions

Preconditions in OWL-S	Functions in Nupddl
drought_degree is_serious_drought	(= (valueof_drought_degree) serious)

Table 6 Mapping the auxiliary elements

Auxiliary elements	Element in Nupddl
error handling	(:types faultType)
precondition_unsatisfied	(:action precondition_unsatisfied)

Magnitude-Compute model service are mapped to the functions in Nupddl, as shown in Table 5.

Due to the characteristics of service composition and planning as model checking, the heterogeneous model composition can be seen as a planning domain with nondeterministic and partial observability. This is because the following reasons:

(a) In realistic cases, the state of the world cannot be completely observed during executions, that is, the availability of elementary model services cannot be determined *a priori*.
(b) The actions may have several possible outcomes. For example, the output of the Drought-Degree model may be mild, moderate or serious.

The MBP provides the capability to solve these problems. However, the MBP also has the following limitation. The planner can only deal with strong planning under partial observability in nondeterministic domains [29], which means that the system requires finding a conditional plan that will result in a successful state, regardless of multiple initial states, the nondeterministic effects of actions, and partial observability. However, the system cannot determine in advance whether the "preconditions" of some "actions" can be satisfied, and there always exists the possibility that some "preconditions" are not met. Therefore, the MBP planner cannot guarantee to achieve the goal for all possible executions of the plan. The consequence of this is that the MBP will fail to generate the plan, due to the fact that the MBP can only deal with strong planning.

Because of this limitation, additional programming is needed in the case where the precondition of the Magnitude-Compute model is not satisfied, or it goes beyond its capacity and fails to find the service plan. Therefore, several auxiliary elements have to be introduced to guarantee obtaining the plan successfully, and these elements have also to be mapped to the elements in Nupddl. These additional elements and their mapping are shown in Table 6.

Now, we can list the whole declarative encoding of elementary model services in a single domain description file:

In Fig. 7, the "oneof" element in the "drought_degree" action is the embodiment of the uncertainty, indicating that the "drought_degree" can take the value from serious, mild and moderate. The value of "drought_degree" must be determined at runtime. Therefore, an "observation" element has to be added, as shown at the bottom of Fig. 7.

```
(define (domain model_composition)
  (:types value faulttype)
  (:constants
     serious mild moderate - value
     hasFault noFault - faulttype
  )
(:predicates
     (agentKnowTheValueOf_drought_degree)
     (agentKnowTheValueOf_drought_duration)
     (agentKnowTheValueOf_drought_area)
     (agentKnowTheValueOf_magnitude_degree)
     (agentKnowTheValueOf_epicenter_area)
     (agentKnowTheValueOf_ave_precipitation_percentage)
)
(:functions
    (valueof_drought_degree) - value
    (fault) - faulttype
)
(:action magnitude_compute
     :precondition
        (and (agentKnowTheValueOf_drought_degree)
             (= (valueof_drought_degree) serious)
             (agentKnowTheValueOf_drought_duration)
             (agentKnowTheValueOf_drought_area)
             (not (agentKnowTheValueOf_magnitude_degree)))
     :effect (agentKnowTheValueOf_magnitude_degree)
)
(:action epicenter_compute
     :precondition
        (and (agentKnowTheValueOf_magnitude_degree)
             (not (agentKnowTheValueOf_epicenter_area)))
     :effect (agentKnowTheValueOf_epicenter_area)
)
(:action drought_degree
     :precondition
        (and
        (agentKnowTheValueOf_ave_precipitation_percentage)
        (not (agentKnowTheValueOf_drought_degree)))
     :effect
        (and (agentKnowTheValueOf_drought_degree)
             (oneof
                (assign (valueof_drought_degree) serious)
                (assign (valueof_drought_degree) mild)
                (assign (valueof_drought_degree) moderate)
              ))
)
(:action fault_process
     :precondition
        (and (not (= (valueof_drought_degree) serious))
                  (= (fault) noFault))
     :effect (assign (fault) hasFault)
)
(:observation output_of_drought_degree_equal_serious
               - boolean
    (imply (= output_of_drought_degree_equal_serious 1)
        (= (valueof_drought_degree) serious))

(imply (= output_of_drought_degree_equal_serious 0)
     (not (= (valueof_drought_degree) serious)))))
```

Fig. 7 The planning domain

```
(define (problem model_composition_pb)
  (:domain model_composition)
  (:init
      (agentKnowTheValueOf_drought_duration)
      (agentKnowTheValueOf_drought_area)
      (agentKnowTheValueOf_ave_precipitation_percentage)
      (not (agentKnowTheValueOf_epicenter_area))
      (not (agentKnowTheValueOf_magnitude_degree))
      (not (agentKnowTheValueOf_drought_degree))
      (= (fault) noFault))
  (:postronggoal (or (agentKnowTheValueOf_epicenter_area)
                     (= (fault) hasFault))))
```

Fig. 8 Problem request

In order to show the compatibility of our method in composing homogeneous models, we added one more quantitative model service, called "epicenter_compute", in Fig. 7. The "epicenter_compute" service receives the magnitude value and computes the epicenter area, and it is a successor of the "magnitude_compute" model.

4.2 Modeling Problem Requests

The problem requests define the initial states and goals, i.e., what we have already known about every elementary model services and what we want to obtain. The initial states are the assignments to the "predicates" and "functions" declared in the Nupddl domain description file, reflecting both the known and unknown information from user inputs. The goal statements are the type of the problem (in this paper, the problem is strong planning under partial observability, termed as "postronggoal"), and the information we want to know through the agent.

In our example, suppose that a user wants to know the epicenter area of the impending earthquake with the knowledge of PAAP, drought_duration and drought_area. Then the problem request can be formulated as in Fig. 8.

Note in Fig. 8 that the user request is either satisfied (the expression "(agentKnowTheValueOf_epicenter_area)") or not (has no solution, i.e., the expression (=(fault) hasFault)). By doing so, it is guaranteed to reach the strong planning goal.

4.3 Generating the Service Plan

Now we can feed the domain description and problem requests to the MBP planner. The generated plan of our example is shown in Fig. 9. As one can see from the figure, the action "drought_degree" executes first, followed by a "switch" statement to test the actual output of the former: if the "drought_degree" equals "serious" then the "magnitude_compute" and "epicenter_compute" actions are executed in sequence; or reach the "fault_process" stage. This logic flow is consistent with the description discussed in Sect. 1.

It is worth mentioning that the "switch" fragment in Fig. 9 is used to test the output of Drought-Degree model, which is equivalent to testing the precondition of the successor Magnitude-Compute model.

```
(define (plan ___undefined___)
(:problem model_composition_pb)
(:domain model_composition)
(:body
 (sequence
   (action (drought_degree))
   (switch
    (case (output_of_drought_degree_equal_serious)
           (sequence
              (action (magnitude_compute))
              (action (epicenter_compute))
              (done)))
    (case (not (output_of_drought_degree_equal_serious))
           (sequence
              (action (fault_process))
              (done)))))))
```

Fig. 9 The exemplar service plan

5 Experimental Studies

This section presents the experimental results to show the feasibility of the proposed approach. The experiments were conducted to identify the factors affecting the generation time of the service plan. We assume that the semantic representation and declarative encoding of elementary model services have already been provided, and the user request is the same as that in Sect. 4.2. All experiments were run on a 1.0 GHz Pentium machine, equipped with 1 GByte memory, running Linux. All WSDL, OWL and OWL-S files are stored, and a distributed library of model services is simulated in this machine. The experimental data were averaged over five independent runs.

5.1 The Number of Elementary Model Services

This set of experiments tests the performance of the proposed approach over the increasing number of elementary model services. We use the model composition example introduced in Sect. 1, but gradually increase the total number of the available model services. Suppose these model services are developed by multiple organizations and irrelevant to our composite model. Intuitively, more models have to be explored, and more time has to be spent to generate the service plan.

As shown in Fig. 10a, the generation time of the service plan increases as the number of elementary model services increases, which is to be expected. A closer observation shows that when the total number of elementary model services is less than 175, the generation time increases at a relatively slow pace. But when it exceeds 175, the time grows rapidly. This result suggests that when a distributed model library has relatively small scale (e.g., less than 175), the developed method can complete the search and composition of model services efficiently. When it comes to a larger-scale composition problem, much longer time are needed.

Existing research work has proved that the plan existence problem of propositional non-probabilistic planning with partial observability is 2-EXP-complete [29]. This is why the time grows rapidly when the number of services becomes big in the

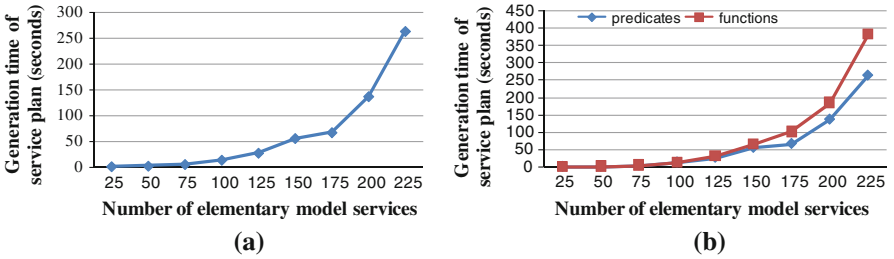

Fig. 10 The generation time of service plans (**a**) under different number of elementary model services and (**b**) with different mapping styles

experiments. Our future studies will investigate efficient heuristic algorithms to find a sub-optimal solution with relatively low time complexity for large scale heterogeneous model compositions.

5.2 Different Ways of Encoding

The mapping between model service descriptions and the Nupddl encoding is not unique. The declarative encoding of the input and output of the model services can be "functions" rather than "predicates" as shown in Sect. 4.1. The "functions" are assigned with "has value" or "no value" to represent the presence of the inputs, outputs and intermediate variables of each elementary model services. In this set of experiments the performance of the two types of mapping are tested as the number of elementary model services increases. The results are shown in Fig. 10b.

We can see from Fig. 10b that the performance of the two types of mapping is very similar when the number of model services is less than 125, and that the performance begins to deviate visibly when the number of model services becomes bigger. It can be seen that the "predicate" mapping style, which is the one used in this paper, is more efficient than the "functions" mapping. The reason for this is because each "function" has two possible states, "has value" or "no value", which can be seen as two "predicates". This is equivalent to increasing the total number of propositions, and therefore the planner needs more time to find the satisfactory solutions.

5.3 The Complexity of Composite Model

Intuitively, more complex the composite model structure is, more time it takes to generate the service plan. In this subsection the service plan is abstracted into a "sequential" pattern of a number of blocks. We then increase the model complexity by increasing the number of blocks, aiming to test its impact on the generation time of the service plan.

The entire service plan generated in Fig. 9 can be abstracted as a "cell" as a whole, as shown in the dashed box in Fig. 11a. The black dots represent actions and observations (the "switch" fragments) in the service plan, while the white dots refer to the internal

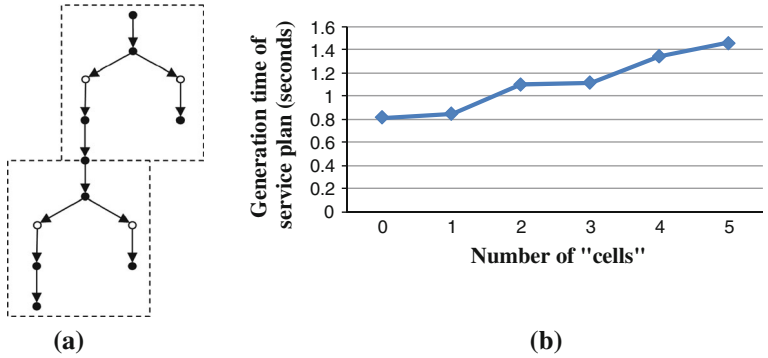

Fig. 11 **a** The construction of complex composite model and **b** the performances

state (the "case" fragments). So model complexity will be increased by linking more cells in sequence.

The experiment results are shown in Fig. 11b. In order to exclude the impact of the number of elementary services discussed in section 0, the total number of model services is fixed at 50. As seen in the figure, the generation time increases approximately in linear with the increase in the number of "cells". It is noteworthy that the increase of the number of "cells" also implies the increase in the total number of "preconditions". Therefore this experiment also shows the impact of the number of "preconditions" on the generation time of the service plan.

6 Conclusions and Future Works

This paper proposes an approach to solving the heterogeneous model composition in DSS field. The heterogeneous model composition is converted to an AI planning problem with nondeterministic and partial observability, and eventually solved by using the planning as model checking technique. We conducted a series of experiments to identify the key factors affecting the time needed to generate the service plan. The experimental results show the feasibility of our method and its capability in dealing with complex requests.

Our current method has the following limitations and therefore future research work has been planned to address them. First, it needs to spend considerable time in handling the large-scale model service composition. In the future, we plan to develop a heuristic approach to composing the models with a low time overhead. Second, the service plan generated in this paper is just a logical structure of the composite model, and the data flow must be added manually to form an executable program (such as BPEL). In order to address this issue, the work has been planned in our future research agenda to investigate the possible approaches to achieving fully automated model compositions. Finally, our current composition method lacks the QoS (quality of service) support. We also plan to incorporate QoS demands into the model composition process.

Acknowledgments This work is jointly supported by National Natural Science Foundations of China (No. 60903174, 61142010), the Fundamental Research Funds for the Central Universities, HUST: 2012QN087, 2012QN088, the Fund of Key Lab for Image Processing and Intelligent Control (20093) and the Leverhulme Trust (Grant No. RPG-101).

References

1. Krishnan, R., Chari, K.: Model management: survey, future research directions and a bibliography. Interact. Trans. OR/MS **3**(1) (2000). http://www.informs.org/Pubs/ITORMS/Archive/Volume-3/No.-1-Krishnan-and-Chari
2. Deokar, A.V., El-Gayar, O.F.: Enabling distributed model management using semantic Web technologies. In: 42nd Hawaii International Conference on System Sciences, 2009 (HICSS '09), 5–8 Jan. 2009, pp. 1–9
3. Madhusudan, T.: A web services framework for distributed model management. Inf, Syst. Frontiers **9**(1), 9–27 (2006). doi:10.1007/s10796-006-9015-2
4. Chari, K.: Model composition in a distributed environment. Decis. Support Syst. **35**(3), 399–413 (2003). doi:10.1016/s0167-9236(02)00116-1
5. Madhusudan, T., Uttamsingh, N.: A declarative approach to composing web services in dynamic environments. Decis. Support Syst. **41**(2), 325–357 (2006). doi:10.1016/j.dss.2004.07.003
6. Karakoc, E., Senkul, P.: Composing semantic Web services under constraints. Expert Syst. Appl. **36**(8), 11021–11029 (2009). doi:10.1016/j.eswa.2009.02.098
7. Sirin, E., Parsia, B., Wu, D., Hendler, J., Nau, D.: HTN planning for Web service composition using SHOP2. Web Semant.: Sci. Serv. Agents World Wide Web **1**(4), 377–396 (2004). doi:10.1016/j.websem.2004.06.005
8. Tang, X., Jiang, C., Zhou, M.: Automatic Web service composition based on Horn clauses and Petri nets. Expert Syst. Appl. **38**(10), 13024–13031 (2011). doi:10.1016/j.eswa.2011.04.102
9. Yeung, W.L.: A formal and visual modeling approach to choreography based web services composition and conformance verification. Expert Syst. Appl. **38**(10), 12772–12785 (2011). doi:10.1016/j.eswa.2011.04.068
10. Panakkat, A., Adeli, H.: Recent efforts in earthquake prediction (1990–2007). Nat. Hazards Rev. **9**(2), 70–80 (2008)
11. Cicerone, R.D., Ebel, J.E., Britton, J.: A systematic compilation of earthquake precursors. Tectonophysics **476**(3–4), 371–396 (2009). doi:10.1016/j.tecto.2009.06.008
12. Geng, Q.: The "seismic drought" Connection in China. Ocean Press, Beijing (1985)
13. McIlraith, S.A., Son, T.C., Honglei, Z.: Semantic Web services. IEEE Intell. Syst. **16**(2), 46–53 (2001). doi:10.1109/5254.920599
14. McIlraith, S.A., Martin, D.L.: Bringing semantics to Web services. IEEE Intell. Syst. **18**(1), 90–93 (2003). doi:10.1109/mis.2003.1179199
15. Chandrasekaran, B., Josephson, J.R., Benjamins, V.R.: What are ontologies, and why do we need them? IEEE Intell. Syst. Their Appl. **14**(1), 20–26 (1999)
16. Bechhofer, S., Harmelen, F.v., Hendler, J., Horrocks, I., McGuinness, D.L., Patel-Schneider, P.F., Stein, L.A.: OWL Web Ontology Language Reference (2004). http://www.w3.org/TR/2004/REC-owl-ref-20040210/
17. Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., Narayanan, S., Paolucci, M., Parsia, B., Payne, T., Sirin, E., Srinivasan, N., Sycara, K.: OWL-S: Semantic Markup for Web Services (2004). http://www.w3.org/Submission/OWL-S/
18. Cimatti, A., Giunchiglia, E., Giunchiglia, F., Traverso, P.: Planning via model checking: a decision procedure for AR. In: Steel, S., Alami, R. (eds.) vol. 1348. Lecture Notes in Computer Science, pp. 130–142. Springer, Berlin (1997)
19. Cimatti, A., Roveri, M., Traverso, P.: Automatic OBDD-based generation of universal plans in non-deterministic domains. Paper Presented at the Proceedings of the Fifteenth National/Tenth Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence, Madison, WI, USA
20. Cimatti, A., Pistore, M., Roveri, M., Traverso, P.: Weak, strong, and strong cyclic planning via symbolic model checking. Artif. Intell. **147**(1–2), 35–84 (2003). doi:10.1016/s0004-3702(02)00374-0
21. Cimatti, A., Roveri, M.: Conformant planning via model checking. In: Biundo, S., Fox, M. (eds.) vol. 1809. Lecture Notes in Computer Science, pp. 21–34. Springer, Berlin (2000)

22. Bertoli, P., Cimatti, A., Pistore, M., Roveri, M., Traverso, P.: MBP: a model based planner. In: IJCAI-2001 Workshop on Planning Under Uncertainty and Incomplete Information, pp. 93–97 (2001)
23. Traverso, P., Pistore, M.: Automated composition of semantic Web services into executable processes. In: McIlraith, S., Plexousakis, D., van Harmelen, F. (eds.) vol. 3298. Lecture Notes in Computer Science, pp. 380–394. Springer, Berlin (2004)
24. Marconi, A., Pistore, M., Poccianti, P.: Automated Web service composition at work: the Amazon/MPS case study. In: IEEE International Conference on Web Services, 2007 (ICWS 2007), 9–13 July 2007, pp. 767–774 (2007)
25. Bertoli, P., Pistore, M., Traverso, P.: Automated composition of Web services via planning in asynchronous domains. Artif. Intell. **174**(3–4), 316–361 (2010). doi:10.1016/j.artint.2009.12.002
26. Liang, T-p: Development of a knowledge-based model management system. Oper. Res. **36**(6), 849–863 (1988). doi:10.1287/opre.36.6.849
27. Bertoli, P., Cimatti, A., Roveri, M., Traverso, P.: Strong planning under partial observability. Artif. Intell. **170**(4–5), 337–384 (2006). doi:10.1016/j.artint.2006.01.004
28. Fox, M., Long, D.: PDDL2.1: an extension to PDDL for expressing temporal planning domains. J. Artif. Int. Res. **20**(1), 61–124 (2003)
29. Rintanen, J.: Complexity of planning with partial observability. Paper Presented at the Proceedings of ICAPS 2004