Semantic Web Services Composition via Planning as Model Checking

Hong Qing Yu and Stephan Reiff-Marganiec
{hqy1,srm13}@le.ac.uk

Department of Computer Science, University of Leicester, UK

Abstract. The ability to automatically compose services is one essential aspect of service oriented architecture. It can reduce time and cost in development and maintenance of complex services and software systems. We are developing a technique to realize this aim by combining the "planning as model checking" approach with Semantic Web Service concepts. We have modified a current planning as model checking algorithm by using a bounded on-the-fly depth-first search algorithm. The result allows for service execution plans to be generated on the fly. One of the challenges is to model a web service as a state transition system. The approach will be suitable in the context of ontologies, but for now we are simply using dictionaries for mapping operations and parameters. The planning as model checking approach forms part of a larger framework to automatically compose services, which addresses several drawbacks of current composition approaches.

1 Introduction

Since service oriented architecture (SoA) was introduced in the 1990s, it became an important solution to realize distributed application via standard web protocols. An implementation of SoA is Web Services (WS). WS are widely adopted; they are based on XML standards that include WSDL, SOAP and UDDI. However, current WS technologies still have some problems, notably a lack of understanding of the purpose of service (i.e. its semantics) by machines, difficulties in describing workflow structures and difficulties in composition of Web Services to achieve a task which cannot be handled by a single WS. Under this background, on the one hand, researchers developed some new standards. BPEL is an example allowing developers to tackle some of these problems. On the other hand, WS description were enriched with semantics leading to Semantic Web Service (SWS). SWS are intended to enable the information regarding the purpose of the service to be understood by machines. At the same time different WS ontology languages have been developed, such as OWL-S, WSMO, and IRS. However, automatic composition is still a main issue of SWS research.

To achieve this aspect, many techniques have been proposed recently. Usually they are based on AI planning techniques. In this paper, we are going to illustrate a technique called planning as model checking to deal with this challenge.

The rest of this paper is organized as follows. In section 2, we give a real world composition scenario which will be used as case study throughout the paper. The third section will list 9 requirements for designing a high quality composition framework. The forth section introduces our process of modelling semantic web services in a way suitable for planning as model checking. In the fifth section, we explain the planning as model checking technique and illustrate the adapted planning algorithm. In the sixth section, we provide an overview of the composition framework. Then we discuss some related research work. Finally, we present a conclusion and our future work plan.

2 Case study

Our real world scenario consists of 6 services representing parts of an online TV shopping case study. Figure 1 shows the 6 services used in the case study: computer location service (WS1), TV information (WS2), TV shops (WS3), delivery (WS4), insurance (WS5) and TV licence (WS6). WS1 is the service which can locate the city of an IP address. WS2 helps people to enquire about all kinds of information regarding TV sets. WS3 is a TV shop chain with local branches that handle requests based on customer locations. WS4 is a delivery service. WS5 shows an online insurance company. WS6 is the TV licence service.

Figure 1 tells that a user (my mum) wants to buy a TV from the Internet. Her goal is to get the TV delivered and obtain insurance for it. In addition some information is known: Mum knows the address of her home and the computer knows its IP address. Figure 2 shows a smart portal application to find a solution for the goal of obtaining the TV and insurance.

Using current technologies, to achieve this goal is a big challenge: the services to be used need already be composed or a BPEL specification for the composition is required. We will use this example in the rest of this paper to illustrate our technique for tackling the automatic SWS composition.

3 Web Services Composition Quality

Web service composition approaches need to satisfy certain requirements in order to provide guarantees for the quality of the compositions that they produce. We consider the following 9 aspects paramount. Majithia et al. [MWG03] have identified some requirements that partly overlap with ours; we will comment on the differences later.

Automation: Semantic Web Services composition needs to be fully automatic. The complete composition framework should not require manual activities, apart from specifying the goal and providing some initial data.

Correctness: Users can (and will) not tolerate unsuitable composite solution, hence guarantees need to be provided that the composite service does indeed achieve the user's goal and works within the user's constraints.

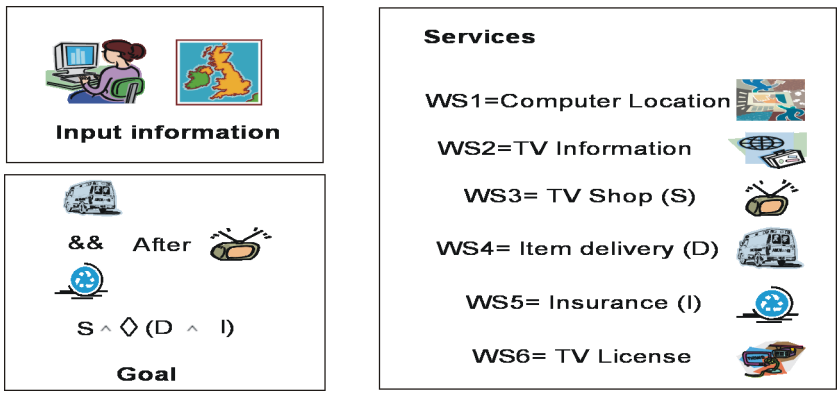

Fig. 1. TV shopping case study : the parts

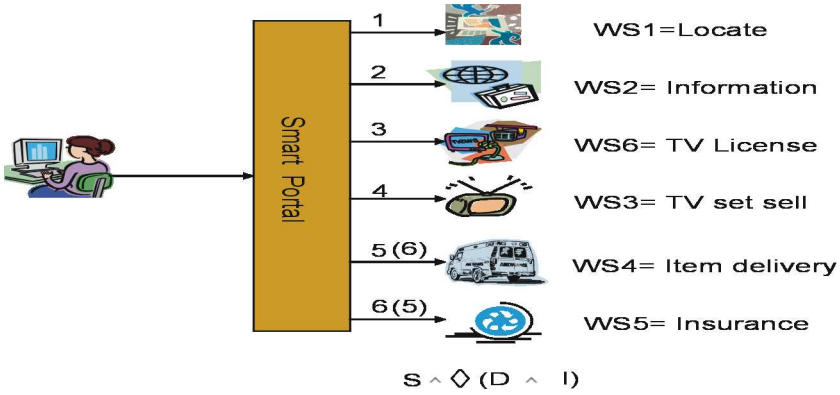

Fig. 2. TV shopping case study : architecture

Fault handling: The composition process should produce fault tolerant solutions, as there are no guarantees that every single registered service is always available. If faults appear at any point during execution, there should be ways to use other equivalent services to fix the problem or at least ensure that no cost has occurred.

Reusability: Reuse is one of the main motivations behind SoA, so composite services should be made available for reuse. This requires producing an abstract composite service description to be provided.

Composition mechanism: This mechanism needs to be able to find an as simple as possible solution in an efficient manner.

Scalability: The composition approach should be scalable to a large number of services.

Low level of skill required for users: There should be no assumptions about the technical knowledge of users; it should be assumed that this is targeted at lay end users.

User-specified optimization: The framework should allow users to specify optimization criteria. For example, a user wants to find the cheapest solution or one that only uses services in their own country.

System provided optimization: The framework should have abilities to select services based on reputation, or trust if such information is available.

As said, these are the basic requirements that we believe a complete automatic services composition framework in semantic environment needs to satisfy.

[MWG03] specifies similar requirements. However, we like to point out some key differences. Firstly, [MWG03] requires workflow granularity supporting mechanisms to allow users to generate workflows of varying levels of granularity. We believe this is not a dynamic way of composing web services. In our opinion, the workflow should be decided at run time by the planning algorithm. Secondly, we suggest correctness and high level of automation as the two most important points, no such qualification has been provided. Finally, we believe reusability to be essential, as we suggested that this is one of the core ideas from SoA; [MWG03] does not mention this aspect.

4 Modelling process

Planning as Model Checking technology is based on transition system model checking, which models systems behaviours as a set of states. WS on the other hand specify their behaviours by exchanging messages and usually the state of a service is unknown. In order to use model checking techniques we have to convert from a message based paradigm to a state oriented one. This can be achieved by letting every single operation in the Web Service imply a state which essentially encapsulates the change after executing the operation. However, for composition purposes Web Services are more than just states: we also need to consider the message flow, business role, service domain and quality requirements. Figure 3 is an overview model of a operation.

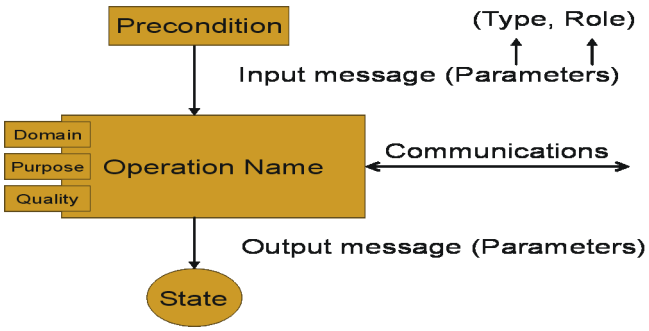

Fig. 3. A WS Operation

In the following we will lay down basic definitions that finally allow us to define the concepts of a "service", "requirement" and "plan"; the former two being input to the planning as model checking algorithms, the latter being the result of the planning process.

We use precondition and postcondition with their usual meaning. Precondition is the condition that has to be satisfied before the invocation of the operation. The postcondition is the final condition that should be achieved after the operation completes.

Model checking is based on ensuring that conditions hold for states; in the planning context we want to ensure that certain conditions are achieved by a service.

Definition 1 (State). *A state σ is a set of post conditions.*

Example 1. Locate web service's state is 'located', TV set information web service's state is 'informed', and delivery service's state is 'delivered': $\sigma_{location} = \{located\}$, $\sigma_{information} = \{informed\}$ and $\sigma_{delivery} = \{delivered\}$.

Note: *start* is a special state representing the empty set of preconditions.

Services interact by message exchange, the information carried by the message is either the output of one service or the input for another. Information carried is identified by parameter names, but also by the type and the domain role that the parameters play. Hence we defined a message as follows:

Definition 2 (Message). *A message is a tuple (P, Ts, Rs) with P being a set of parameter names, Ts being a set of functions that map types T to parameters: $Ts = \{t|\forall p \in P : t(p) \rightarrow T\}$ and Rs being a set of functions that map domain roles R to parameters: $Rs = \{r|\forall p \in P : t(p) \rightarrow R\}$*

Example 2. The delivery service's input message has one parameter which assigns string as type, and IP address as bussiness role . The output message also has one parameter and its role is the location name and again the type is string: $(address, string, IPAddress)$ and $(address, string, LocationName)$

An operation is the smallest observable unit of work that can occur in a service. As we will see, operations form the cornerstone of plans.

Definition 3 (Operation). *An operation is defined by a tuple (Pre, M_{in}, M_{out}, St, Com, P, D and Q) where Pre is a (possibly empty) set of preconditions, M_{in} and M_{out} are the input and output message respectively. St is the state reached after the operation has completed. Com describes the kinds of communication that will appear during execution of the operation, P describes the business function offered by the operation, which should be seen in the context of the Domain D. Finally, Q gives the operation's qualitative properties (non-functional attributes).*

Services consist of operations and some internal logic which describes their interaction protocols. However, for the purpose of this paper we work with a simple definition of service, that ignores the interaction protocol.

Definition 4 (Service). *A service S is a set of operations.*

Having defined the concept of service, we can now place our attention on defining the concept of requirement. The idea behind requirement is that it models the information that the end-user provides.

Definition 5 (Initial State). *The initial state is a tuple (D, P) with D being a set of user data elements and P a set of pre conditions.*

A Goal expresses what the user desires to achieve. In addition to simple goals, we allow for conjunction and eventually to be used as combination operators in a goal.

Definition 6 (Goal). *A goal G is a combination of post conditions.*

Definition 7 (Requirement). *A Requirement R is a tuple (G, IS, D, Q) with G being a goal and IS being the intital state. D and Q are the domain information and quality requirements.*

Example 3. In our case, the goal is to obtain a TV set and get insurance for it, the IS is start (i.e. no precondition), the domain is e-shopping, and Q could be {English language, high security}, hence the requirement is:
$(\{delivered, insured\}, \{start\}, e-shopping, \{Englishlanguage, highsecurity\})$.

Finally we define the concepts of plan and transition as a step in a plan. Recall that the plan is the desired results of the planning step in the framework.

Definition 8 (Transition). *A transition is a function: $St \times O_s \rightarrow St$ with St being a set of states and O_s an operation from a specific service: s.*

Definition 9 (Plan). *A plan is a labelled transition system, the labels are the operations, nodes are states. Also, transitions can have a weights associated and the transition system can be non-deterministic.*

We also use quality, domain, purpose and communication to describe other aspects of operations; here we only give informal definitions:

Quality describes the operation's quality and it may include amongst others, charge, security, privacy, time, rate, language and locations. In general, it is seen as the mechanism to capture non-functional requirements.
Domain is the operation's area of interest defined within a given taxonomy.
Purpose describes the operation's aims, again within a given taxonomy.
Communication describes a message exchange protocol with service's client and other services.

We have used examples from our case study services to illustrate the definitions. For completeness we now provide models for all 6 services in our case study using the graphical notation (Fig. 4).

Some of the definitions above might at a first glance look similar to those in [MBE03]; they are different and we will discuss this in more detail in section 7.3.

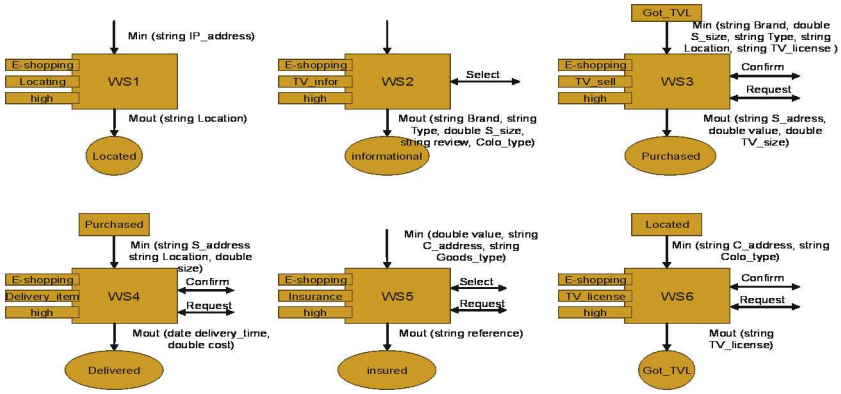

Fig. 4. Modelling 6 services in our case study

5 Planning as Model Checking

In general, model checking allows to validate a formal model of a system against a logical specification. The model checking problem is described as given a Kripke structure $M = (S, R, L)$ that represents a finite-state concurrent system and a temporal logic formula φ expressing some desired specification, find the set of all s in S that satisfy $\varphi : \{s \in S \mid M, s \models \varphi\}$. Traditionally, this technique is used to verify the correctness of hardware circuits and network protocols, for further details we refer the reader to literature on model checking, e.g. [CGP99,HOL99].

More recently, this idea has been used to deal with planning problems [FG99]. The key idea underlying this work is that planning problems should be solved

model-theoretically and planning domains are formalized as semantic models. Properties of planning domains are formalized as temporal formulas. Planning is done by verifying whether temporal formulas are true in a semantic model. Meanwhile, [CRT98,DTV99] introduced three planning as model checking algorithms addressing different level: 'the strong cyclic planning', 'strong planning' and 'weak planning'. The attribute refers to the solution: for a weak solution there is at least one path in the plan reaching the goal, for strong solutions all paths reach the goal; the strong cyclic plans allow all fair executions to reach a solution, which allows for stepping back to (a) previous state(s).

However, the strong cyclic planning solution is not suitable in the web services domain, due to allowing stepping back: in services we normally have created some effect when a step is executed, some of which are world changing. In these cases, we can only cancel the action which is usually achieved by a special cancellation action that the service might offer. We decided to use the strong planning algorithm as the basis for this work, as it seemed best to use the strongest solution possible in the context. We modified the algorithm by changing the plan direction from goal state to initial state and adding the semantics of operation into the algorithm.

We discuss the modified algorithm in the following. P1 is the set of all operations we encountered in the modelling services (we will show more detail in section 6, when discussing the framework), P2 and K are empty at the start of the algorithm. They will be used to store the relevant operations (i.e. those are in the interesting domain) and all plans respectively.

```
Set P1, P2, K
Stack D = {}
int Depth=0,  Bound = MAXDEPTH
Operation Oi=initial Operation
Goal Sgoal = Specification.goal
List condition = initial condition knowledge list
List parameter = initial parameter list
```

The start procedure fills the set of operations so that the planning algorithm can search only those that are meaningful in the domain of the problem. It then starts the model checking algorithm.

```
PROCEDURE start () {
   D.push(Oi);
   FOR EACH Operation Op in P1 space{
       IF Op.domain includes Spcification.domain THEN
             P2.add(Op)
       END IF
   }
   search ()
}

PROCEDURE search () {
```

```
   IF (Depth >= bound) THEN
      return
   END IF
   //We limit our searching depth to one that is requested
   by the client or meaningful for the domain.
   Depth++
   Operation O=D.top()
   IF(D.S satisfy Sgoal) THEN
      K=K+D
      //If we found stack D'S is equal to the goal, we
      //will save this stack as a plan and continue to
      //search another plans.
      D.pop()
      Depth--
      continue
   END IF
```

Every operation that passes the transition test will be collected as a starting point for the next search. Then, this operation's states and output messages will be added or removed or take the place of the original in the condition and parameter lists (it is removed if the operation "consumes" the state, added if it is introduced and updated if it is changed but not consumed).

```
   Operation M[]
   M=TransitionTest(O)
   int i=0
   WHILE(M[i]!=EOF) DO
       IF (D.includes(M[i])) THEN
           continue
       END IF
       D.push(M[i])
       condition.update(M[i].S)
       Parameter.update(M[i].Mout)
       search()
   OD
   D.pop()
   Depth--
}
```

The transition tests essentially establishes whether the operation can be executed in the current state. We expect this test to become more complicated once we add communication protocols to the services.

```
FUNCTION transitionTest (Operation O){
   int i=0
   int j=0
```

```
    Operation[] M
    FOR EACH Operation Op in P2 {
        Boolean f=true
        IF (NOT condition.includes(Op.Pre) THEN
            f=false
        END IF
        IF (NOT Parameter.satisfy(Op.Message)) THEN
            f=false
        END IF
        IF (f==true) THEN
            M[j]=P2[i]
            j++
        END IF
        i++
    }
    return M
}
```

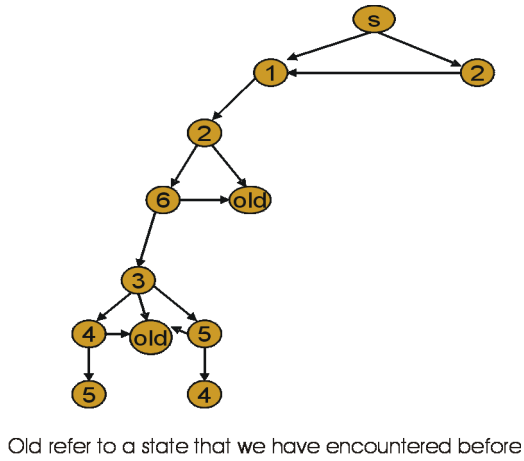

Fig. 5. Results of running our Planning as Model Checking algorithm

In our case study, we obtain the result shown in Fig. 5. There are two plans that can achieve our defined goal, namely: S → 1 → 2 → 3 → 4 → 5 and S → 1 → 2 → 3 → 5 → 4.

6 Overview of composition framework

6.1 SWSC framework

The algorithm forms a small part of the overall framework for WS composition. While this paper concentrates on the modelling and the algorithm, we want to use this section to place these in their context. This framework has four phases as shown in Fig. 6.

Phase 1 specifies the planning goal (the domain of interest, the goal and the bound for the search depth) and describes initial knowledge about the client (initial input message and initial states) as input for the next phase. The condition list and parameter list are initialized.

Phase 2 is the model extraction step. We automatically select web services to build our plan search space from our repository. The repository stores web services' domain, ontology type and web addresses. We can then capture the web services information through their published ontology and map the concepts to a common ontology (that is, our framework is not dependant on a particular ontology). The details about a web service are extracted automatically and cast into our model. The well modelled web services are prepared for phase 3.

Phase 3 now can run our core algorithm to search for plans. At this step, we might get a set of suitable plans.

Phase 4 is the physical composition step. Firstly, we allow clients to choose the best plan (future work will use non-functional properties to identify this automatically). We then generate an executable plan (e.g.in BPEL), but we will monitor the execution and in case of failure will automatically revise the executable plan based on alternatives available from the planning in step 3. Note that we might encounter situation where no suitable plan can be found: this is reported to the users.

6.2 Evaluation

With respect to the contents mentioned in section 3, this framework achieves the following requirements.

1. High degree of automation: the user only needs to provide an initial specification, the rest of the work is performed automatically.
2. High degree of correctness: the possible plans all stem from a process that guarantees correctness by construction. Further, execution is monitored and hence allows for correcting solutions when services become unavailable by drawing on alternative plans.
3. Reusability is high: after obtaining the plans, these are modelled as composite services and added to the repositories. This enables plans to be reused in the future, either as they stand or as part of further compositions.
4. Low level of skill required for users: our requirement specification language is very simple to use, it only requires basic knowledge of composition goal, initial parameters, numbers of services and quality.
5. User-specified optimization criteria: This framework allows users to specify priorities and express their desired criteria through non-functional attributes.

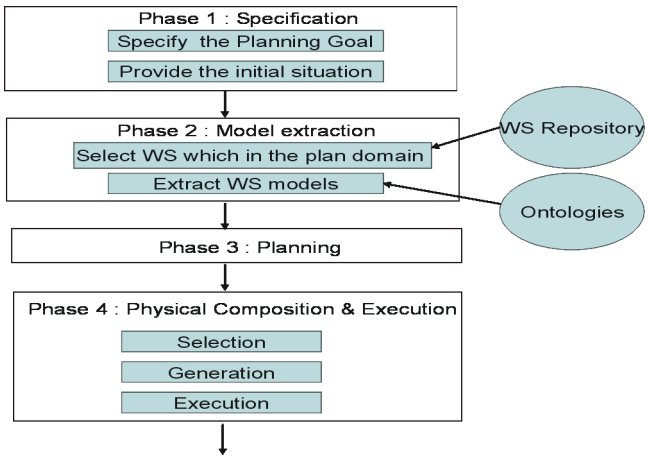

Fig. 6. Our overview framework

7 Related work

In 2003, a preliminary composition framework was introduced by Sycata [SPAS03], which built on AI planning technology. However, it is a basic result stemming from early research on combining AI planning with Semantic Web Services to solve automatic composition problems. It includes two components: The DAML-S Matchmaker that is in charge of registering services and selecting the proper services and the DAML-S Virtual Machine to find the workflow solution. Since the composition mechanism is based on classical AI planning theories, the planning is focussed on determining web services states. There is no mechanism for handling faults. Also, operations step in the background, despite being the really observable entity in service orieneted computing.

Since, some related work has been done by different groups using different markup languages, logical languages, planning systems and algorithms. However, the basic idea is to introduce AI planning to address the problem of automatic web services composition. There are three typical directions and these research works as we will show next.

7.1 Backward STRIPS-style planner based on DAML-S

This framework was developed by Sheshagiri in 2003 [SF03]. It uses STRIPS-style services to compose a plan, which gives the goal and a set of basic services. The backward-chaining algorithm is used. The process starts from converting the DAML-S ServiceModel descriptions of services into Verb-Subject-Object (VSO) triples. Then, they write a set of rules and queries that transform services encoded as VSO triples into a set of facts that form the planning operator. The planner applies two steps repeatedly until none of the services satisfy any of the

goals. The first step is to find services that satisfy the existing goal and add them to the plan. The second step is to store all the operators' inputs and preconditions into step 1's set of goals. If all outstanding goals are external inputs, then composition is successful. As a result, a simple graphical workflow can be generated.

The weakness is that it assumes there always exists a service which can satisfy a given goal. Moreover, all the output artifacts produced during execution are desired by developers, hence it addresses low level aspects. This framework also suffers when dealing with a large number of web service for composition, because there is no method to adjust to different domains.

7.2 Integrating SHOP2 HTN planning system with OWL-S

Another framework [SP04] integrates OWL-S with the SHOP2 HTN planning system. The core process is to encode the OWL-S Process Models as SHOP2 domain. After encoding, the SHOP2 HTN planning system will perform the planning tasks. Comparing SHOP2 HTN planner to our requirements, on the one hand, we can see that it is better than the first two planning systems. First, SHOP2 includes the problem the domain in the planning problem definition. Secondly, it provides some rules of automatically translating OWL-S to SHOP2 domain. Finally, the HTN planning provides natural places for human intervention at plan time. On the other hand, it still has some shortcomings. Firstly, the algorithm does not allow for complicated preconditions, which naturally might arise in web services: one service might depend on the output of several others. Secondly, it is assumed that all services will be available at anytime. Finally, user-specified optimization criteria are not considered.

7.3 The WebDG framework

[MBE03] presents another approach for composing web services technology based on the semantic web. The WebDG web services composition framework proposes an approach for automatic Services Composition based on nodes ontology.

The WebDG defines composability rules such as Mode Composability, Message Composability, Operation semantic Composability, Qualitative Composability and Composition soundness. There are four phases in this framework. Firstly, CSSL (Composite Service Specification Language) describes the specification of a composite service scope. The CSSL is a WSDL-like language defined by WebDG framework. Secondly, the Matchmaker is finding the composition plan based on Composability Rules. Thirdly, the Selection phase decides on a users desired plan dependent on QoC (Quality of Composition) parameters which are defined in the clients' CSSL. Finally, an optional detailed description of a composite service will be generated. The three important features of this generation are that it can be customized, extended and reused. In addition, it gives every single message, operation and Web Service a purpose and a category definition. Thus, the scalability is dramatically increased.

However, the obvious weakness is that the user should have rich knowledge about XML, specification and low-level programming. Furthermore, the degree of fault-tolerance is still low because there is no chance to replace unavailable services.

Also, their definitions rely on their nodes ontology language, that is there work is bound to a specific ontology language – our approach does not rely on a specific ontology language.

More crucially, WebDG use CSSL language as specification language; hence they actually express the workflow in a BPEL like language. The matchmaker is then used to find services whose semantic definitions satisfy this CSSL specification. This is fundamentally different to our work, which tries not to find services for a specific task, but rather to establish the workflow automatically.

8 Future work and conclusion

In this paper, we use a real world example to propose an overview basic framework structure and we focused on modelling services and presenting our core planning as model checking algorithm. It uses a bounded On-The-Fly Depth-First Search technique which has advantages including being independent of semantic ontology languages, simple goal specification, generation of executable plans, high scalability and reusability. However, in our first version of the algorithm implementation stage, we only deal with simple goal specifications.

In the future work, on the one hand we will continue to complete and enhance our framework. On the other hand we will make our goal specification language more flexible to specify client's side goal request.

References

[CGP99] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. Model checking. *The MIT Press, Cambridge, Massachusetts, London, England*, 1999.

[CRT98] A. Cimatti, M. Roveri, and P. Traverso. Strong planning in non-deterministic domains via model checking. *In Proc. of AIPS98*, 1998.

[DTV99] Marco Daniele, Paolo Traverso, and Moshe Y. Vardi. Strong cyclic planning revisited. *In Proc. 5th European Conference on Planning (ECP '99)*, 1999.

[FG99] P. Traverso F. Giunchiglia. Planning as model checking. *In Proc. 5th European Conference on Planning (ECP '99)*, 1999.

[HOL99] Gerard J. HOLZMANN. Software model checking. *Engineering Theories of Software Construction*, 1999.

[MBE03] Brahim Medjahed, Athman Bouguettaya, and Ahmed K. Elmagarmid. Composing web services on the semantic web. *Springer-Verlag*, September 2003.

[MWG03] Shalil Majithia, David W. Walker, and W.A. Gray. A framework for automated service composition in service-oriented architectures. *Cardiff School of Computer Science, Cardiff University, UK*, 2003.

[SF03] Sheshangiri and T. Finin. A planner for composing service described in daml-s. *International Conference on Automated Planning and Scheduling*, 2003.

[SP04] Evren Sirin and Bijan Parsia. Planning for semantic web services. *University of Maryland*, 2004.

[SPAS03] Katia P. Sycara, Massimo Paolucci, Anupriya Ankolekar, and Naveen Srinivasan. Automated discovery, interaction and composition of semantic web services. *Web Semantics*, December 2003.