Composing Semantic Web services with Interaction Protocols

Zixin Wu, John F. Harney, Kunal Verma, John A. Miller, Amit P. Sheth

LSDIS Lab, University of Georgia, Athens, Georgia
{wu, harney, verma, amit, jam} @cs.uga.edu

Abstract. Web service composition has quickly become an important area of research in the services oriented architecture community. One of the challenges in composition is the existence of heterogeneities between independently created and autonomously managed Web service requesters and Web service providers. This paper focuses on the problem of composing Web services in the presence of ordering constraints on their operations imposed by the service providers. We refer to the ordering constraints on an services operations as interaction protocol We present a novel approach to composition involving what we term as pseudo operations to expressively capture the service provider's interaction protocol. Pseudo operations are used to resolve heterogeneities by constructing a plan of services in a more intelligent and efficient manner. They accomplish this by utilizing descriptive human knowledge from the service provider and capture this knowledge as part of a planning problem to create more flexible and expressive Web service operations that may be made available to service requesters. We use a customer-retailer scenario to show that this method alleviates planning complexities and generates more robust Web service compositions. Empirical testing was performed using this scenario and compared to existing methods to show the improvement attributable to our method.

1. Introduction

Web services are software systems designed to support interoperable machine-to-machine interaction over a network. They are the preferred standards-based way to realize Service Oriented Architecture computing. A problem that has seen much interest from the research community is that of automated composition of Web services to realize Web service compositions or Web processes by leveraging the functionality of autonomously created services. Previous work in this regard has considered various approaches to composition, and have included use of HTN [1], Golog [2], classic AI planning and constraint satisfaction [3]. Much of the previous work has not considered the ordering constraints that may be imposed service providers on the invocation of their operations. For example, a service provider may require the requestor to login before executing any other operation and this ordering constraint must be respected in any composition that uses this service.

We refer to the ordering constraints on the operations of a service as the Interaction Protocol of the service (also called conversation protocol in previous literature [4]). These constraints may range from simple sequential ordering of operations to very complicated structures involving advanced constructs such as nested looping and branching dependencies.

Our focus is on automatically composing Web services based on user requirements in the presence of ordering constraints specified by the service providers. We use semantic representation of the functionality of operations as a building block for this work. The semantic descriptions of the operations are based on preconditions and effects, as well as, the data semantics of the operations [5]. The semantic representation is based on the formal semantics of recent W3C member submission WSDL-S [6] and is used by our planning algorithm to generate the compositions. In order to handle interaction protocol during planning, we provide an expressive construct named pseudo operations, to represent such dependencies. A pseudo operation can be defined as a collection of service operations with ordering constraints that can be viewed at a higher level of abstract as a single operation. We also introduce a novel extension to classical AI planning techniques that incorporates the concept of semantic data type matching. This work was done as part of the METEOR-S project[7] which focuses on creating a broad framework of semantics (data, functional, non-functional and execution) for supporting the complete lifecycle of Web services.

The contributions of this paper are the following:

- This is one of the first papers to talk about composition in presence of interaction protocols.
- Using pseudo operations in planning allows certain workflow patterns such as loops to be included in the compositions, something that is not easily done using simple planning techniques.
- Our evaluation shows that the complexity of the planning problem is significantly reduced; representing multiple operations as one pseudo operation allows the planner to perform a less intensive searching strategy.

The remainder of this paper is organized as follows. We first introduce a motivating scenario. In section 3, we discuss the problem of protocol heterogeneity, and define our approach of using Semantic Template and pseudo operation for service composition in Semantic Web service. Section 4 discusses the system architecture, and section 5 gives the evaluation result. Finally a discussion of related work in section 6 and conclusions and future work in section 7 are presented.

2. Motivating scenario

In order to illustrate the need for interaction protocol we present a motivating scenario specified by the Semantic Web Services Challenge 2006[8] shown in Figure 1. A customer (depicted on the left side of the figure) desires to purchase goods from a provider (depicted on the right side of the figure) offering retail services. The customer sends a request for an order. The expectation of the customer is that upon

the request being submitted, the order will be processed and a purchase order confirmation will be

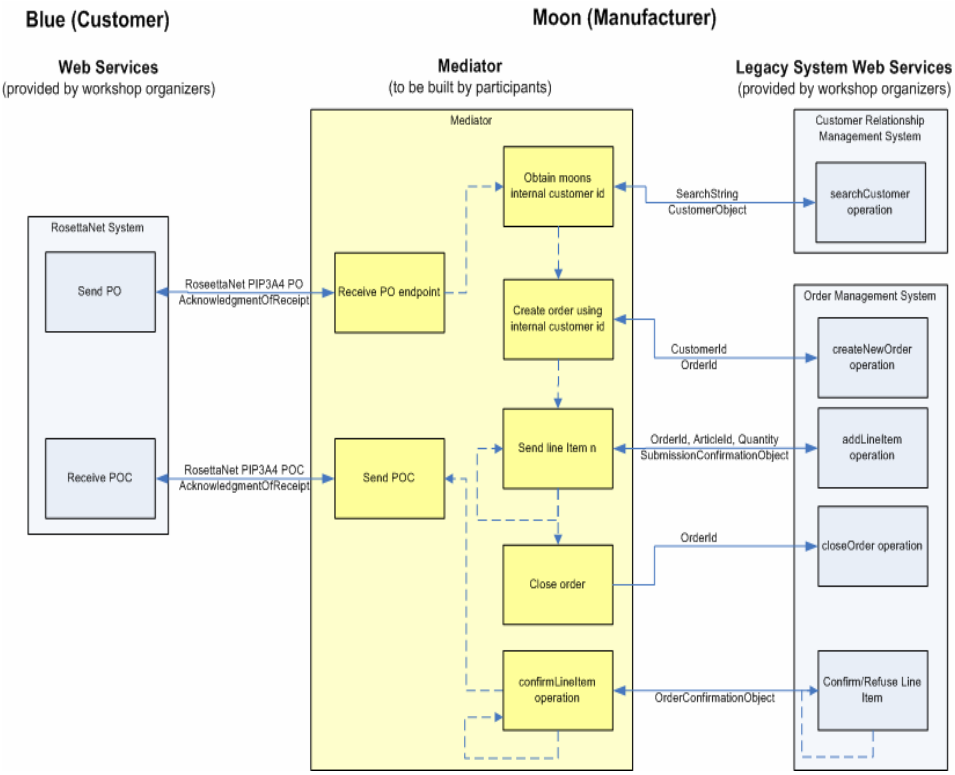

Fig. 1. Customer – Retailer scenario[1]

received, verifying that the order was received and processed by the provider. The provider owns an array of legacy services defined in WSDL that consist of operations performing many different tasks. However, none of the operations of these services explicitly performs the functionality of the customer's request.

It is clear that the provider does not have one atomic service that has the functionality of receiving a purchase order request from the customer as input, processes this request and supplies a purchase order confirmation to customer. Thus, the customer has no way to simply invoke one service to get its desired purchase order process. This exemplifies *protocol heterogeneity*.

The operations offered do, however, have the ability to perform the customer's desired task when synthesized together in a suitable manner. The provider will have to utilize a sequence of these operations and make them available through some

[1] http://sws-challenge.org/wiki/index.php/Scenario: _Purchase_Order_Mediation

interface that the customer can use. First, the provider must create a customer identification number using the search customer operation, then invoke the createOrder operation to initialize the purchase order, followed by invoking addLineItem operation (possibly many times) to allow individual products to be placed in an order, the closing the order when all desired products have been added to the purchase order, and finally sending an acknowledgement back to the customer. Therefore, without a mediation mechanism to introduce a service that satisfies the customer's requirements there would be no method of interaction between the customer and the provider.

3. Interaction Protocol approach for Web service composition

The scenario illustrates an important problem in the Web services composition domain. Ideally, a client would prefer to invoke any provider's services without effort or risk. Conversely, a provider needs to maintain its services such that its constraints are satisfied. Our approach attempts to alleviate efforts by both parties by introducing a protocol mediation mechanism using pseudo operations.

3.1 Background

There are two categories of partners that are described within the web services domain, namely the service provider and service requester. [1] A service provider presents its web service functionality by providing a set of operation specifications (or operations). These operations allow service requesters to use these services by simply invoking them. These operations might be inter-dependent. Using these available operations, a service requester performs one or more inter-related steps, also known as tasks, to achieve the desired goal. Tasks can be best viewed as activities in a process and can be divided into smaller and more concrete sub-tasks [1]. These tasks may be organized into a process specification which may consist of one or more activities that may be executed one or more times.

Service requesters and providers are autonomously created. This causes heterogeneity to exist between the requester and provider when Web services are aggregated to perform a task. These heterogeneities in general may exist at many different levels, such as the data level or at a communication/protocol level. We say that protocol heterogeneity exists when any activity instances (i.e. an execution of an activity) cannot be finished by atomically invoking exactly one operation once, or the invocation violates the dependencies on a service provider. There are three types of protocol heterogeneity that may exist:

1. One activity instance can be finished only by invoking one operation more than once or by invoking more than one operation. We call this a 1:n problem. Our motivating scenario described above related to the 1:n problem, since there are two independent tasks (Send PO, Receive POC), and each task has no sub-task.

[1] " Web Services Glossary" (http://www.w3.org/TR/ws-gloss/), and the discussion of terminologies (http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/#wordonspr).

2. More than one activity instance can be finished only by invoking exactly one operation once. We call this a n:1 problem.
3. More than one activity instance can be finished only by invoking one operation more than once or more than one operation. Naturally this is called a n:n problem.

Figure 2 shows the different types of protocol heterogeneity.

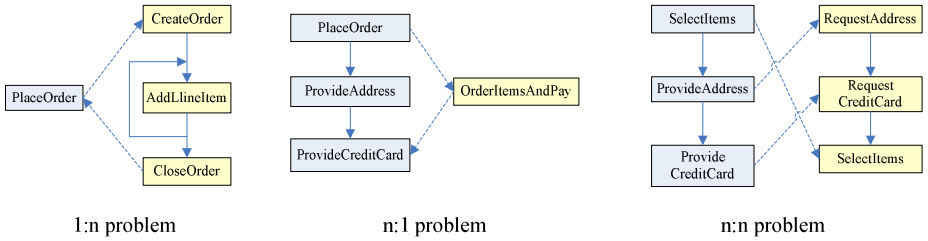

Fig. 2. Three types of Protocol Heterogeneity

This paper focuses on 1:n problems (the others are left to future work).

3.2 Formal Model

A method that can be used to solve the problem of protocol mediation in Web services is the application of AI planning techniques. Since our approach does not restrict to a specific planning technique, we can simply adopt any planning technique that uses propositional logic. Most classical AI planning problems are defined by the STRIPS representational language (or its variants like ADL), which divides its representational scheme into three components, namely, states, goals, and actions. Planners that use STRIPS suffer from the branching problem and combinatorial searching complexities. This places additional burdens on the planner, especially for Web service composition problems with many available operations.

Our approach to protocol mediation attempts to alleviate these drawbacks. First, we introduce an extension of the STRIPS language as the base representational language of our method. Second, we allow the service provider to represent its interaction protocol in an expressive way by using pseudo operations, which encompass composite processes that the service provider makes available based on its own functional constraints. Then, the planner exploits this knowledge in effectively constructing a plan. In order accommodate this new methodology, we must use a definition that extends the classical AI planning problem. This extension will focus on two aspects of the problem - semantics (by using ontologies) and type checking (considering input/output of operations). Our extension can defined as the following:

– **Extended State.** We extend a traditional notion of STRIPS state by adding a set of semantic data types in order to ensure the input of an operation is available before it is invoked. An extended state *s* has two components: s = <SF, ST>, where:

- *SF* is a conjunction of a set of **status flags** (literals) representing the status of a process by using concepts in a domain ontology. We use propositional logic for status flags, thus a status flag is a propositional variable. Status flags with *False*

value are prefixed by ¬. We also assume an open-world assumption, i.e., any status flag not mentioned in the state is unknown.
- *ST* is a set of **semantic types** representing the availability of data by using concepts in a domain ontology. We use description logic for semantic types.

An example state could be:
<ontology1#orderPlaced & ¬ ontology1#orderPaid, (ontology2#orderID)>

– **Condition.** A condition also has two components as a state, except that a condition is a requirement for a state.
– For example, <ontology1#orderPaid, ontology2#paidAmount>

– **Semantic Web Service**[9]. Our definition of a semantic Web service is based on the definition of WSDL and proposed SWS specifications – OWL-S [10] and WSDL-S, with Interaction Protocol included.

$$SWS = (\bigcup_i \{sop_i\}) \cup (\bigcup_i \{psop_i\})^1$$

Where, SWS is the union of a set of semantic operations (sop) and a set of pseudo semantic operations (psop).
sop = <op:FunctionalConcept, input:SemanticType, output:SemanticType, Pre, Effects, fault:SemanticFault>

Where, a semantic operation (or operation) (sop) is defined as a 6- tuple of the following:
- *op* is an operation mapped to a functional concept in a domain ontology.
- *Pre* is the precondition. It is a conjunction of status flags stating which status flags must be true (or false) before an operation can be executed.
- *Eff* is the effect. It is a conjunction of status flags describing how the status flags in a state changes when the action is executed. Status flags prefixed by ¬ are called negative effects, whereas others are called positive effects.
- *input* is mapped to a set of semantic types stating what data are required in order to execute the operation.
- *output* is mapped to a set of semantic types stating what data are available after the operation is executed.
- *fault* is the exceptions of the operation represented using concepts in a domain ontology.

A pseudo operation can be used to define the Interaction Protocol expressively. It is defined as follows.
sop = <op:FunctionalConcept, input:SemanticType, output:SemanticType, Pre, Effects, fault:SemanticFault, Process>

Where, a pseudo semantic operation (or pseudo operation) (psop) is defined as a 7-tuple. All the elements are the same as in sop, except that *Process* is a process calculus expression borrowed from CCS[11]. We define the syntax of *Process* in a pseudo operation in EBNF as follows.

Process := flow | process '|' flow

[1] We union a set of operations and a set of pseudo operations by ignoring the "process" component in the pseudo operations.

flow := seq | flow '||' seq
seq := loop | seq ';' loop
loop := sop | sop '*' | sop '+' | '(' process ')'

Figure 3 illustrates an example of the representation of the provder's services described in our running scenario. Using our definition, we allocate the operation parameters for "CreateNewOrder", "AddLineItem", "CloseOrder", and "PlaceOrder" accordingly. "PlaceOrder" is a pseudo operation is composed of the other operations.

Operation name	Precondition	Input	Effect	Output
CreateNewOrder	haveCustomerID	CustomerIdentification	haveOrderID^ ¬ completeOrder^ ¬ closeOrder	OrderIdentification
AddLineItem	haveOrderID^ ¬ completeOrder	LineItemEntry	completeOrder	LineItemSubmission
CloseOrder	completeOrder	OrderIdentification	closeOrder	ConfirmedOrder
PlaceOrder	haveCustomerID	CustomerIdentification, LineItemEntry	haveOrderID^ completeOrder^ closeOrder	ConfirmedOrder

Fig. 3. Representation of Order Management System Web Service

Figure 4 gives the flow diagram of the "PlaceOrder" and its corresponding expression using the syntax shown above. Here, the grammar states a defined sequence of atomic operations. The operation "createNewOrder" is to be invoked first. Following this invocation, the "addLineItem" operation is invoked one (or more) times - the "*" denotes that the operation is contained within a loop structure – and then the "closeOrder" operation is invoked to complete the sequence.

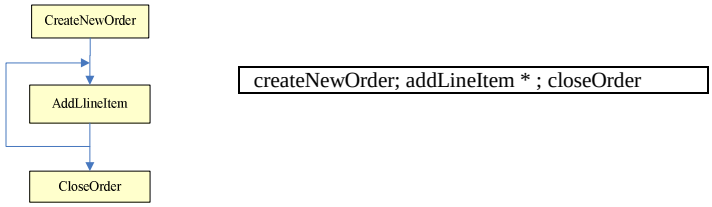

Fig. 4. "PlaceOrder" pseudo operation

– **Semantic Template.** While a semantic Web service definition represents a service provider on a concrete level, a semantic template captures the semantic capabilities of a service provider on an abstract level. It is the way a service requester defines its task specifications.

$ST = \bigcup_i \{sopt_i\}$

$sopt = <Action_o, I_o, O_o, Pre_o, Eff_o, F_o>$

Where, a semantic operation template (sopt) is an abstract representation of the functionality of an operation. It is defined as a 6-tuple that is similar to a sop, except that it is the requirement for an operation.
The following table illustrates the sopt "SendPO" from the scenario.

Operation name	Precondition	Input	Effect	Output
SendPO	haveCompanyInfo	OrderInformation	completeOrder^ closeOrder	Acknowledgement

A semantic Web service composition problem is composing a set of semantic Web services (SWSs) to fulfill the requirement of one specific semantic operation template. Before we give the definition of the problem, we need to define the initial state, the goal, the "apply" and "satisfy" operator for the problem.

For convenience, we use Eff(a) to represent the effect of operation a, SF(s) for the set of status flags in state s, SF(eff) for the set of status flags in eff, value(sf_s) for the value of status flag sf in state s, value(sf_{eff}) for the value of status flag sf in effect eff, ST(s) for the set of semantic types in state s, and ST(c) for the set of semantic types in condition c.

- **Initial state.** One of the possible extended states when the service requester's process runs to an activity[1] which defined by the semantic operation template $sopt_p$. It is defined by the precondition and input of the semantic operation template.

$s_0 = <Pre(sopt_p), In(sopt_p)>$

Initial state, like the extended state, includes a set of status flags and semantic types.

- **Goal.** A goal is a condition which is defined by the effect and output of the semantic operation template $sopt_p$.

$g = < Eff(sopt_p), Out(sopt_p)>$

- **"Apply" operator.** We use the notation "+" for "apply" operator which is a function mapping an extended state *s* and an operation (or pseudo operation) *a* to a new extended state *s'*:

$+: (s, a) \mapsto s'$ (Alternatively, we may write s + a = s')

$$SF(s') = SF(s) \bigcup SF(Eff(a))$$

$$\forall sf \in SF(Eff(a)), value(sf_{s'}) = value(sf_{Eff(a)})$$

$$\forall sf \in SF(s) \wedge sf \notin SF(Eff(a)), value(sf_{s'}) = value(sf_s)$$

That is, a positive/negative status flag in *eff* is also in s' with the same value, while any status flag in *s* but not in *eff* is assumed to remain in *s'* with the same value in *s*.

[1] There may be more than one initial state for an activity because of the non-deterministic behavior of previous activities. For the time being, we only handle one initial state.

$ST(s') = ST(s) \cup out$
That is, a semantic type in *out* is added to *s'* if it is not in s, while other semantic types in *s* are also in *s'*.

– **"Satisfy" operator.** We use the notation "→" for "satisfy" operator which is a function mapping an extended state *s* and a condition *c* to T or F:

$\rightarrow$: (s, c) $\mapsto$ {T, F}

This function maps to T (we call it "*s* satisfies *c*" and we may write it as: s→c) if and only if:

- c$\vdash$s, that is, for each status flag $sf_c \in SF(c)$, there is a semantically equivalent status flag $sf_s \in SF(s)$, such that their values are the same.

$\forall st_c \in ST(c), \exists st_s \in ST(s)$ |$st_s \sqsubseteq st_c \vee (st_c$ is part of $st_s)$, that is for each semantic type $st_c \in ST(c)$, there exists a semantic type $st_s \in ST(s)$, such that st_c subsumes st_s. or st_c is part of st_s

– A semantic Web service composition problem is the following function:

$p: (sopt, SWS_s) \mapsto plan$

Where,

- *sopt* is a semantic operation template.
- SWS_s is the union of the given semantic Web services.
- *plan* is a partially ordered set of operations. Every sequence (total order) of operations (say $a_1, a_2, \ldots a_n$) that satisfies the partial order must conform to the following restrictions:

$s_0 \rightarrow <Pre(a_1), In(a_1)>$
$s_0 + a_1 = s_1$
$s_i \rightarrow <Pre(a_{i+1}), In(a_{i+1})>$
$s_i + a_{i+1} = s_{i+1}$
$s_n \rightarrow g$

Where s_0 is the initial state, g is the goal, and $a_i \in SWS_s$. Remember that a condition has two components: set of status flags and set of semantic types, so the condition for s_0 consists of the precondition and input of a_1.

3.3 Extending the graphplan Algorithm

Our planner devises a plan using an extension of the graphplan algorithm [7]. In addition to modeling the preconditions and effects, our extension effectively handles the semantic typing of data that flows within the Web service operations of the plan. This allows the planner to account for the semantic and state dependencies that exist within the provider's protocol.

Although our approach may be utilized by a broad range of planning algorithms, we have chosen to implement the graphplan algorithm for a variety of reasons. First, the algorithm has the ability to extract a plan without suffering from any inaccuracies (eg illogical web service compositions). Second, the algorithm can be easily applied to larger scale planning problems, such as the problem that we are attempting to resolve. Third, the algorithm can be easily extended to use the model

we defined above. Finally, the application of this algorithm materializes quite well the improvements that pseudo-operations within the interaction protocol will contribute.

The general idea of the graphplan algorithm is to represent the combination of possible states and actions that may exist in the plan as an expandable graph data structure, where nodes may be represented as state parameters (status flags and semantic types) and actions, and edges representing relations between the state parameters and actions. The structure consists of a sequence of alternating state and action levels which consist of state parameters and actions (operations in the context of Web services), respectively, that may exist at that particular level. The graph is expanded by adding the effects of actions to the existing state. When the goal literals are contained in a state level, the algorithm attempts to extract a plan by backtracking through the action levels of the graph starting from the goal state. Our approach requires an extension to the *EXPANDGRAPH* function to accommodate the semantic type parameters defined above. Pseudocode for this procedure is shown in figure 5.

```
function EXPANDGRAPH(graph)
  create new action level a_i
  create new state level s_i
  for all op ∈ A
    if pre(op) ∈SF(s_i-1) and in(op) ∈ ST(s_i-1)
       add op to a_i
       add eff(op) to s_i
       add out(op) to s_i
```

Figure 5 – Pseudocode for EXPANDGRAPH

An operation may only be invoked when its preconditions exist in the current state level of the graph (ie *pre(op) ∈ current graph level StatusFlags*) *and* there is an input data type (*in(op) ∈current graph level semantic type*). When an operation is placed in the graph, its effects as well as output data types are added to the existing conditions of the previous state. Thus, the new current state level becomes the new effects and output data of the operations in addition to the previous state's defining literals and data types.

The introduction of pseudo-operations allows a significant simplification of the graphplan algorithm. This is because pseudo-operations effectively aggregate multiple operations that can be read by the algorithm as one action. Therefore, the graph structure does not need to grow large and the EXTRACT-SOLUTION function has fewer combinations of operations to select for searching. Let us refer to figure 6, which depicts part of the planning graph for our scenario. At state level n, there exists the status flag "haveCustomerID" and semantic type "CustomerID". Thus sufficient state parameters exist in state level *n* for pseudo operation "PlaceOrder" to be placed in the corresponding action level. Similarly, state parameters exist for the atomic service "CreateNewOrder" to be placed in the same action level. Using the "PlaceOrder" pseudo operation allows the new state level *n+1* to add its three effects ("haveOrderID", "closeOrder", and "completeOrder") to the existing status flags in state level n. The "CreateNewOrder" atomic operation only provides the "haveOrderID" status flag. In order to obtain the "closeOrder" and "completeOrder" status flags, the graph would have to expand again to subsequent state levels (*n+2*, *n+3*, etc). The graph structure becomes unnecessarily complicated and the planner

would have to search through many combinations of the atomic operations at each subsequent level created.

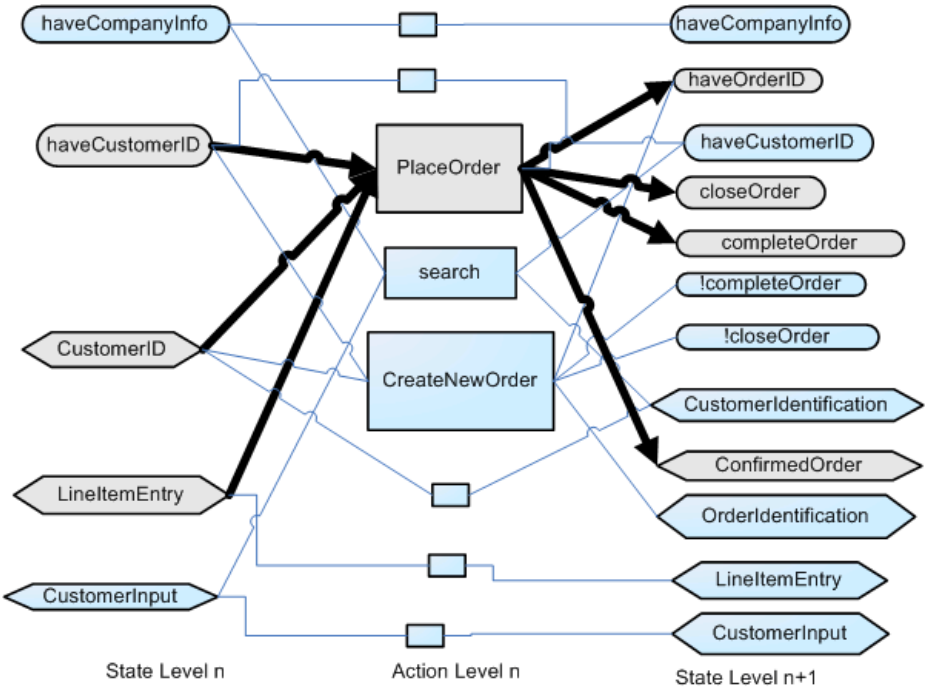

Figure 6 – Planning Graph Level *n* with pseudo operation "PlaceOrder"

4. Architecture

Our implemented architecture is depicted in figure 7. The system accepts both the semantic template from the service requester and the set of WSDL-S files from the service providers, which gives the available operations, to create the desired BPEL process file. We utilize WSDL-S4J as a parser to handle both the semantic template file and WSDL-S files from providers. When the user invokes the mediation process, the initial state, goal, and action set will be created from the semantic template file and the WSDL-S files from providers and the plan will be created. If the plan includes a pseudo operation, transformation is performed. When a target process (BPEL) is generated, we use BPWS4J to serialize it into a file.

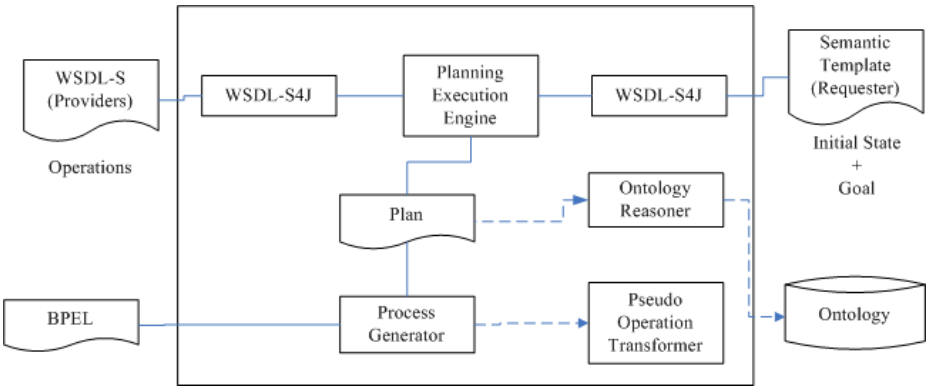

Figure 7 – Architecture

5. Evaluation

We have said that pseudo operations improve existing planning methods in two ways. First, pseudo operations effectively reduce planning complexity (ie the runtime) of composing plans of services. We ran 8 separate simulated instances of our running scenario to illustrate. The simulations attempted to juxtapose the run times of composing the process using the pseudo operation "PlaceOrder" (illustrated earlier in figure 3) and composing the process consisting of the atomic operations "createNewOrder, "addLineItem" and "closeOrder". Figure 8 shows the comparison results of these tests. The plot in general shows that the use of pseudo operations reduces the run time of the planner using strictly atomic services (the first test has a greater runtime due to the existence of a memory cache). We expect that the runtimes will exhibit a greater differential for larger scale processes. Second, pseudo operations have the ability to model complex workflow patterns. Perhaps the greatest strength of the pseudo operation is the fact that it has the ability to represent complex workflow patterns. Such is the case in our example, which requires a possible looping structure over the atomic operation "addLineItem". Without pseudo operations, a typical planner could not model this structure, because it represents a non-deterministic operation (most planners are design for deterministic applications). In a sense, the true desired operation of the service requester could not be fulfilled (the atomic operations would be invoked many times unnecessarily).

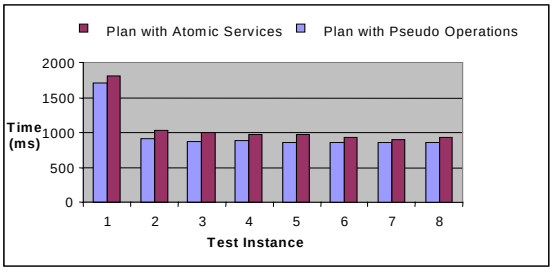

Figure 8 – Empirical Test Results

6. Related work

The authors in [19] found a correlation between Hierarchical task network (HTN) planning and web service representation in the OWL-S framework. HTN planning uses the approach of refining plans by applying action, or task, decompositions. The idea is to divide high-level tasks into smaller and smaller sub-tasks until more primitive tasks can be performed directly and correctly using partial-order planning over a smaller subset of actions. The benefit of this approach is that planning complexity may be reduced for tasks that require many actions. Web service activities can be modeled in much the same way. While our approach similarly attempts to alleviate planning complexity, it differs from HTNs in that we attempt to synthesize processes using the constraints defined by the provider. These parameters contribute in creating well-defined and useable sub-services that contribute to less demanding plan construction. Instead of synthesizing a plan consisting of strictly atomic services, the planner can take advantage of pseudo-services that may perform the tasks of multiple atomic services that internally consist of the provder's constraints. Thus, there is a higher likelihood that the planner will have little or no difficulty in finding an execution path in a relatively fast and efficient manner.

WSMO [12] refers to the problem of protocol mediation as process mediation. A significant difference between their approach and ours is that they try to create mediators at runtime to resolve the heterogeneities. We create compositions that satisfy all the protocol requirements eschewing the need for mediation. Duan et al., [13] discusses using the pre- and post-conditions of actions and automatic synthesis of Web services by finding a backbone path as the first step. Using pseudo operations allows considering complex constructs such as loops in the plan.

7. Conclusion and future work

In this paper, we have addressed some of the problematic issues presented by protocol heterogeneities that exist in the Web service domain. We proposed a method to overcome these challenges through the utilization of the Interaction Protocol, which is represented expressively by pseudo operations. Encompassing this method

are novel extensions to the graphplan algorithm, which introduces semantic type checking to process planning, and the flexibility of generating complex workflow patterns that are difficult for a general planner to produce. Finally, through experimentation, we have shown that this method significantly reduces computational complexity of planning systems.

We have identified the following three items for our future work. First, we would like to integrate our approach with METEOR-S [7] discovery so that new services can be considered for composition. Second, we would like to further extend our planning algorithm to consider non-deterministic and hierarchical pseudo operations, which allows more flexible workflow patterns to be utilized. Finally, we would like to investigate resolving heterogeneities at the data level.

8. References

[1] Evren Sirin, Bijan Parsia, Dan Wu, James Hendler, and Dana Nau. HTN planning for web service composition using SHOP2. Web Semantics Journal, (2004) 377-396

[2] Levesque, H.; Reiter, R.; Lesperance, Y.; Lin, F.; and Scherl, R. 1997. Golog: A logic programming language for dynamic domains. Journal of Logic Programming 31:59-- 84.

[3] Russell, S., Norvig, P. 2003. *Artificial Intelligence: A Modern Approach (Second Edition)*. Prentice Hall.

[4] Xiaochuan Yi, A CPNets-based Design and Analysis Framework for Service Oriented Distributed Systems. July 7, 2005.

[5] A.P. Sheth, "Semantic Web Process Lifecycle: Role of Semantics in Annotation, Discovery, Composition and Orchestration," Invited Talk, ESSW Workshop, 2003.

[6] Web Service Semantics - WSDL-S (http://www.w3.org/Submission/WSDL-S/)

[7]METEOR-S Semantic Web Services and Processses, http://lsdis.cs.uga.edu/projects/METEOR-S

[8] Semantic Web Services Challenge 2006 (http://deri.stanford.edu/challenge/2006/)

[9] Kunal Verma, Configuration and Adaptation of Semantic Web Processes, Ph.D. Thesis, Dept. of Computer Science, University of Georgia, 2006.

[10] OWL Services Coalition, OWL-S: Semantic markup for web services, OWL-S White Paper http://www.daml.org/services/owl-s/0.9/owl-s.pdf (2003).

[11] Robin Milner, Calculus of Communicating Systems

[12] Web service Modeling Ontology (WSMO), http://www.wsmo.org

[13] Ziyang Duan, Arthur J. Bernstein, Philip M. Lewis, Shiyong Lu: A model for abstract process specification, verification and composition. ICSOC 2004: 232-241