SHOP2 and TLPlan for Proactive Service Composition

Maja Vukovic and Peter Robinson

University of Cambridge Computer Laboratory
15 JJ Thomson Avenue
Cambridge CB3 0FD, UK
{firstname.lastname}@cl.cam.ac.uk

Abstract. Recent advances in computer technology are making the development of context aware applications possible. Such applications are complicated by the variety of contextual types that must be accommodated, together with the range of values for each context type. This makes it difficult to write and extend them. We are addressing this by building context aware applications as dynamically composed sequences of calls to Web services, considered as an AI planning problem. We identify the following three specific technical requirements for planning systems in order to handle Web service composition problem: (1) richness of domain description, (2) control constructs for assembling complex actions, and (3) a mechanism for plan optimization. In this paper, we compare two hand-coded planners, SHOP2 and TLPlan, and discuss their applicability to modelling and composing of Web services, using a specific context aware composition problem.

1 Introduction

The development of context-aware applications has become a complex task due to the need to accommodate for the potentially vast variety of – possibly even unanticipated – context types and their values that may be encountered. Simply hard-coding the mapping between all possible combinations of context values and the corresponding application behavior, is not only impractical, but also makes systems difficult to later extend to take into account new values of existing context types and new context types.

We are addressing this problem by constructing context aware applications as dynamically composed sequences of calls to fine granularity Web services [1]; where different service compositions of such sequences will result from different contexts such as: resources available, time constraints, user requirements and location.

By explicitly declaring Web services as processes in terms of their inputs, outputs, preconditions and effects, this paper shows how we employ goal-oriented inferencing from planning technology for service composition. We compare two hand-coded planning systems, Simple Hierarchial Ordered Planner 2 (SHOP2) [4] and TLPlan [5], and evaluate their suitability for handling the Web service composition problem.

Hand-coded planners are domain-independent planners, which use domain-specific control knowledge to help them plan effectively. SHOP2 is based on hierarchical task network (HTN) planning. The central motivation for using SHOP2 was to devise a set of (abstract) HTN methods that will encode something akin to “standard operating procedures” capturing multi-step techniques for refining a task, to further facilitate design of

patterns for Web service composites. TLPlan does a forward-chaining search in which it applies planning operators to the current state to generate its successors. In contrast to SHOP2, TLPlan uses temporal logics to express search control knowledge.

The remainder of this paper is structured as follows. Section 2 analyzes related work. Section 3 defines the main requirements for the planning systems to handle composition of Web services. Based on the scenario presented in Section 4, Section 5 compares applicability of SHOP2 and TLPlan to the problem of proactive service composition. We conclude and outline areas of future work in Section 6.

2 Related Work

Planning technology has been used in a variety of application domains including robotics, process planning, web-based information gathering, and spacecraft mission control. It recently gained much attention to support enterprise application integration as Koehler et al. analyzed [6]. We discuss a number of related projects, which employ planning approach to Web service composition.

Automatic Web service composition using SHOP2 is also investigated by Wu et al. [7]. They observe that exclusion of concurrent processes (split and join constructs) in SHOP2 imposes a serious limitation on the usefulness of this methodology.

McIlraith et al. [8] extend Golog [9], a high level logic programming language built on top of Situation Calculus [10], for use in service composition. They use an off-line planning technique to construct general templates, which are then modified based on user preferences, resulting in a composite plan. They employ sensing actions when the knowledge of the initial state is incomplete, or many actions exist that can change world in unpredictable ways.

McDermott [11] investigates applicability of estimated regression planners, which use a backward analysis of difficulty of a goal to guide a forward search through situation space. By extending the Unpop planner [12] McDermott created Optop ("Opt-based total-order planner"). The main motivation for this work is to relax the assumption of complete knowledge required by classical planners, and to formalize what they do not know and how could they find out more about the world. McDermott also points out the necessity for planners to support synthesis of branching and looping plans.

3 Composition Challenges for Planning Systems

Planning systems generate (partially) ordered sequences of actions (or plans) that solve a goal. They start from a domain specification, consisting of valid action descriptions, which includes both the conditions under which an action applies (the preconditions) and the expected outcome of applying that action (the effects). Potentially enormous search space and the difficulty in fully and accurately representing real-world problems are two key challenges for the planning systems.

In this section we pose specific technical requirements that need to be met by planning systems to provide automatic, context aware, Web service composition.

3.1 Richness of Domain Descriptions

The Planning Domain Definition Language(PDDL) [13] is the standard, action-centered, language for the encoding of planning domains, based on STRIPS [14] formalism.

PDDL 2.1 is an extension of PDDL for expressing temporal planning domains, and is separated into different levels of expressivity. The following four are required for modelling composite Web services:

Level 1 ADL [15] Planning: Includes the ability to express a type structure for the objects in a domain, typing the parameters that appear in actions and constraining he types of arguments to predicates, actions with negative preconditions and conditional effects and the use of quantification in expressing both pre- and post-conditions.

Level 2 Numerical constructs: Allows for numerical variables and the ability to test and update their values instantaneously

Level 3 Discrete durative actions: Explicit representation of time and duration.

Level 4 Continuous durative actions: Actions with continuous effects

3.2 Control Constructs

Structured composite services prescribe the order in which a collection of activities (services) take place. They describe how a service is created by composing the basic activities it performs into structures that express the control patterns, data flow, handling of faults and external events, and coordination of message exchanges between service instances.

We identify the following four groups of control constructs for assembling primitive actions into a complex actions that collectively comprise an applications:

1. Sequential ordering	2. Iteration
3. Nondeterministic choice	4. Concurrency and synchronization

3.3 Plan optimization

In the real world, services consume resources, such as network bandwidth, and have a monetary cost associated with their execution. Therefore a mechanism is required to be able to present metrics and resource constraints on each service as well as the resulting plan.

4 Scenario: Mail Replication System

We use Web service composition to synthesize a suitable procedure for mail replication dynamically based on user location, activity, computing device and network bandwidth. Mail replication consists of two subprocesses executed in parallel: *retrieve mail* and *send mail*.

Table. 1 shows the different context types and the expected behavior of the simplified mail retrieval subprocess. Activity and the location of the user determine the presentation mode of the incoming mail. Network bandwidth, type of the computing

	Input: Context data				Output: Expected behavior
Case	**Activity**	**Network**	**Device**	**Location**	**Retrieve Mail**
1	Walking	GPRS	Smart Phone	Street	Display headers
2	Driving	GPRS	Embedded	In-vehicle	Read out headers
3	Not Driving	WLAN	Embedded	In-vehicle	Display adapted mail
4	Working	LAN	Laptop	At desk	Display full mail

Table 1. Context and expected application behavior in simplified mail retrieval process.

device (and consequently its screen size and color depth), affect the mail retrieval. For example, rather than retrieving all the mail over the slow connection only the mail headings are initially downloaded.

5 SHOP2 and TLPlan for Web Service Composition

In this section we describe how SHOP2 and TLPlan can be applied to the problem of orchestrating activities (i.e. individual Web services) in order to achieve a complex task (i.e. a composite Web service). We evaluate how SHOP2 and TLPlan meet the requirements set out in Section 3, highlight their benefits and discuss their limitations.

5.1 Web Service Composition Using SHOP2

SHOP2 is a domain-independent HTN-based planner. It uses the idea of *hierarchical task network decomposition* to decompose an abstract task into a group of operators that forms a plan implementing the task. Planning progresses as a recursive application of the methods to decompose tasks into smaller and smaller subtasks, until the primitive tasks, which can be performed directly using the planning operators, are reached. In the case where the plan later turns out to be infeasible, SHOP2 will backtrack and try other applicable methods.

Domain Definition To model the reference scenario in SHOP2 we initially devised a general problem-solving strategy for mail retrieval, consisting of abstract tasks. We have then implemented these as a description of a planning domain in SHOP2, in terms of a set of axioms, operators, and methods, which prescribe how to decompose tasks., as shown in Figure 1.

An *operator* is used to indicate how a primitive task can be performed. For example, operator for mail decomposition is shown in Fig. 2(a). Operators are STRIPS-based, and are at least as expressive as Level 2 PDDL actions. In addition, each operator also has an optional cost associated with it, which can be used to find the best plan given the objective function. At present we have used the default cost 1.0.

A *method* is used to define the decomposition of a compound task into a partially ordered set of primitive or compound subtasks. Fig. 2(b) shows the method for mail processing. *Conditional expressions* in the method descriptions may be used to enumerate possible flows in the process, and therefore address the lack of branching constructs in SHOP2. This approach is however impractical for context awareness, where

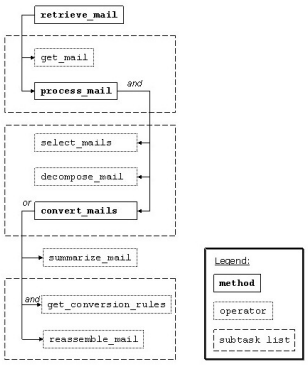

Fig. 1. SHOP2 Task hierarchy for the simplified mail retrieval subprocess

```
(:operator
    ;head: name and parameters
    (!decompose_mail ?mail)
    ;precondition
    (know ?mail)
    ;delete list: negative side effects
    (session_created)
    ;add list: positive side effects
    ((know ?mail_id)
     (know ?mail_header)
     (know ?mail_body)
     (know ?attachment)
     (know ?attachement_size)
     (know ?attachement_type)))
```

(a) SHOP2 operator: decompose mail

```
(:method
     ;head
      (process_mails ?list_of_mails)
     ;precondition
      ((session_created)
       (know ?list_of_mails))
     ;subtasks
      (:ordered
       (!select_mail ?list_of_mails)
       (!decompose_mail ?mail)
       (:task convert_mail ?mail)
       ))
```

(b) SHOP2 method: process mail

Fig. 2. Sample SHOP2 operator and method definition.

the data range has greater magnitude. Enumerating all the possible conditions that must be planned for is not only infeasible but would result in exponential growth with the number of steps in the plan.

Problem Definition The description of a planning problem in SHOP2 consists of an initial state and a task to be accomplished, defined in STRIPS, as shown in Fig. 3(a). The goal is the task "`retrieve_mail`", with input parameters username and password, as well as the type of the device used (e.g. `in_vehicle_inf_sys`). Context data, such as `location` and `connection_type`, also forms the description of the initial state.

Plan A plan consists of a list of operators, which can be applied to achieve the goal.

```
(defproblem mail_case2 mail_system(
 (activity driving)
 (location in_vehicle)
 (connection_type GPRS)
 (has bandwidth 9600)
 (device_type embedded_system)
 (embedded_system
    in_vehicle_inf_sys)
 (username john)
 (password testpswd)
 (valid_login john testpswd))
 )
```

(a) SHOP2 problem definition: case 2.

```
((retrieve_mail
    john testpswd
    in_vehicle_inf_sys)))
```

```
(!GET_MAIL SERVER1 JOHN) 1.0
(!GET_MAIL SERVER2 JOHN) 1.0
(!GET_MAIL SERVER3 JOHN) 1.0
(!SELECT_MAIL    #:?LIST_OF_MAILS1789) 1.0
(!DECOMPOSE_MAIL #:?MAIL1807) 1.0
(!SUMMARIZE_MAIL
    #:?MAIL_HEADER1819
    #:?ATTACHMENT_TYPE1820
    #:?ATTACHMENT_SIZE1821)  1.0
(!TXT_TO_SPEECH
    #:?LIST_OF_MAILS1789) 1.0
```

(b) SHOP2 plan: case 2.

Fig. 3. SHOP2 Problem definition and resulting plan for use case 2.

For example, plan for case 2, shown in Fig. 3(b), is a sequence of the following steps: getting the mail from three different servers, selecting mails, decomposing them, summarizing and then ”presenting” them to user using the text to speech service. In SHOP2 notation `!` is a prefix for operator symbol.

5.2 Web Service Composition Using TLPlan

TLPlan uses domain specific search control information to control simple forward chaining search, where the planning operators are applied to the current state to generate its successors. TLPlan therefore knows the current state of the world at every step of the planning process. Control rules, which are written in temporal logic, provide domain-specific knowledge to tell the planner which states should be avoided, therefore allowing the planner to backtrack and try other paths in the search space.

Domain Definition The domain definition in TLPlan, partially shown in Fig. 4(a), consists of predicate and function symbols, which can be *described* and *defined*; and operators. Predicates and functions are specified by name and their arity. For example, predicate `device_type` takes one parameter. There must be some described symbols, which are essentially predicated and functions that are updated by actions. For example `know_conversion_rules` gets updated once the knowledge of this predicate is acquired. Furthermore, one can define new predicates and symbols (in the form of first order formulas). For example the predicate `(same ?x ?y)`.

Operators, in either STRIPS or ADL form, are then specified using these declared and defined predicates and functions. They consist of list of preconditions, adds (predicates that become true) and deletes (negative effects that should be removed from the

```
;Described symbols
 (predicate device_type 1)
 (predicate know_conversion_rules 0)
;Defined symbols
 (predicate same 2)
 (def-defined-predicate (same ?x ?y)
 (= ?x ?y))
;Operator to decompose mail
(def-strips-operator
  (decompose_mail ?mail)
     (pre
        (incoming_mail ?mail)
        (has_bandwidth ?b)
        (<= ?b 9600))
     (add
        (know_mail_body mb ?mail)
        (know_attachment a ?mail)
        (mail_decomposed ?mail)))
```

(a) TLPlan domain definition

```
;; Goal
 (define (retrieve_mail_case2)
 (mail_fetched) (inbox_displayed))
;; Plan
(login john testpswd)
(get_mail john server1)
(get_mail john server2)
(get_mail john server3)
(decompose_mail mail)
(summarize_mail mail)
(txt_to_speech mail)
(mail-retrieved)
```

(b) TLLPlan problem and resulting plan.

Fig. 4. TLPlan domain, problem and plan for use case 2.

world).

Problem Definition Problem definition in TLPlan is similar to that in SHOP2, and consists of predicates and functions describing the initial state of the world.

In contrast to SHOP2, where the goal is the task (i.e. method) to be achieved, in TLPlan the goal is specified using a list of predicate and function specifications, as shown in Fig. 4(b).

Plan The plan generated by TLPlan is quite similar to the one generated by SHOP2, shown in Fig. 3(b). The key difference is the order of operators. This is mainly due to the method abstractions and their ordering constructs used by SHOP2 to define the subtasks.

5.3 SHOP2 and TLPlan Comparison

In this section we distill some general observations about SHOP2 and TLPlan and their applicability to Web service composition, given the experience with implementation of

[1] ADL includes the ability to express a type structure for the objects in a domain, typing the parameters that appear in actions and constraining the types of arguments to predicates, actions with negative preconditions and conditional effects and the use of quantification in expressing both pre- and post-conditions.

[1] Bacchus et al. [16] extend TLPlan to handle concurrent actions (with variable duration).

	Planning System	
Feature	**SHOP2** [4]	**TLPlan [5]**
Planning methodology	HTN	Forward chaining
Richness of domain descriptions		
PDDL 2.1 Level 1: ADL Planning	✓	✓
PDDL 2.1 Level 2: Numeric Constructs	✓	✓
PDDL 2.1 Level 3: Discrete Durative Actions	?	✓
PDDL 2.1 Level 4: Continuous Durative Actions	?	✓
Control Constructs		
Sequential Ordering	✓	✓
Iteration	✓	✓
Concurrency and Synchronization	✓	?[1]
Non-deterministic choice	×	?
Plan optimization	✓	✓

Table 2. Suitability of SHOP2 and TLPlan for Web service composition.
Legend: × = not supported, ? = partially supported or work-around available, ✓ = fully supported

the sample scenario. Table 2 compares relevant features of each planner based on the criteria set out in Section 3.

Planning methodology and its implications SHOP2 and TLPlan are both hand-coded planning systems, however they differ in the kind of control knowledge they use. SHOP2 employs HTN methods to guide which parts of the search space should be explored. On the other hand, TLPlan uses the temporal formulas to tell which part of the search space should be avoided. SHOP2's HTN approach gives more structure to the domain and the way a problem should be solved. Furthermore, this concept could be exploited to create patterns of composite Web service.

The main disadvantage of both planners is that whilst hand-coded search does help them plan effectively, it creates a significant overhead. Consequently it requires expertise in both the domain and specifics of the planner, and therefore put limitations on level of automatization of Web service composition process.

Expressiveness and support for PDDL These two planning systems have equivalent expressive power and are similar in many respects. They are both Turing-complete, because they allow function symbols. Furthermore both SHOP2 and TLPlan allow attached procedures and numeric computations. They know the current state at each step of the planning process, and use this to prune operators. Both SHOP2 and TLPlan support external subroutines.

TLPlan is capable of reading the problem definition and generating the plan in this format, but does not support PDDL-based domain specification. SHOP2 supports actions of at least Level 2 in PDDL, and even though it does not provide explicit support of the durative actions in Level 3 of PDDL, it has sufficiently expressive power to rep-

resent durative and concurrent actions given the following three characteristics.

Control constructs While SHOP2 allows for tasks to be sequentially ordered, there is no mechanism to handle the control constructs related to concurrency, namely: parallel split, synchronization and exclusive choice. At the moment this is resolved by enumerating every possible flow in the process using conditional expressions in the method descriptions. This increases the complexity of search space, and planning. Bacchus et al. [16] extend TLPlan to handle concurrent actions (with variable duration).

Parametric overloading A further syntactical issue is the problem of parametric overloading, where a number of operators have the same name but different signatures, nevertheless providing the same functionality. It is not supported by SHOP2 at the operator level. Whilst there is a workaround, the lack of support for parametric overloading conflicts with the conceptual model where we associate the planning operators with executable (Web) services. (Because planning involves matching each operator description this concept is not commonly supported by conventional planning systems, as an optimization of the search process.)

Goal representation In contrast to TLPlan, in SHOP2 the goal can not be stated declaratively. SHOP2 has to know in advance which method it should call. Consequently the planner fails if asked to solve a completely new, unknown problem for which no method definition exists.

Domain and problem complexity The number of the axioms in the problem description impairs planner's performance. In our experiment each problem definition is described with a limited set of facts and our domain is highly simplified, with the intention to keep the search space minimal.

This raises one of the central challenges in optimizing the composition process—where does the information about the state of the world come from and at which point of time? For example, how and when does one retrieve axioms describing the attachment conversion rules for the mail system? One approach is to create a set of so-called "sensing" actions, which when necessary retrieve additional axioms about the world, as McIlliarth et al. [8] demonstrate.

6 Conclusion and Future Work

We are tackling the increasing complexity required for context awareness by building context aware applications through the dynamic, planning-based, composition of Web services.

In this paper, we compared two hand-coded planning systems, SHOP2 and TLPlan for their suitability to automate Web service composition, based on the following three technical requirements: (1) richness of domain description, (2) control constructs for assembling complex actions, and (3) a mechanism for plan optimization.

By composing a specific context aware application automatically, using SHOP2 and TLPlan planners, we identified the lack of complex control structures involving

concurrency, iteration and nondeterministic choice to generate expressive compositions as the key shortcoming. Another open problem arises from the deterministic nature of SHOP2 and TLPlan, as they assume that that the state of the world is always accessible, static and deterministic. In contrast, Web services tend to create new objects at runtime, and this needs to be accommodated for.

Our future work will involve investigating nondeterministic planners, motivated by the unpredictability of pervasive computing environments.

Acknowledgements. IBM Zurich Research Laboratory is supporting this work.

References

1. Vukovic, M., Robinson, P.: Adaptive, Planning Based, Web Service Composition for Context Awareness. In: Advances in Pervasive Computing. Volume 176. (2004) 247–252
2. Shaw, M., Garlan, D.: Software Architecture: Perspectives on an Emerging Discipline. Prentice-Hall, Inc. (1996)
3. Russell, S.J., Norvig, P.: Artificial Intelligence: A Modern Approach. Prentice Hall, Upper Saddle River, NJ (1995)
4. Nau, D., Munoz-Avila, H., Cao, Y., Lotem, A., Mitchell, S.: Total-Order Planning with Partially Ordered Subtasks. Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, Morgan Kaufmann, San Francisco (2001) 425–430
5. Bacchus, F., Kabanza, F.: Using Temporal Logic to Control Search in a Forward Chaining Planner. In: Proceedings of Second International Workshop on Temporal Repre sentation and Reasoning (TIME), Melbourne Beach, Florida (1995)
6. Koehler, J., Srivastava, B.: Web service composition: Current solutions and open problems. In: ICAPS 2003 Workshop on Planning for Web Services. (2003)
7. Wu, D., Sirin, E., Hendler, J., Nau, D., Parsia, B.: Automatic Web Services Composition Using SHOP2. In: 13th International Conference on Automated Planning & Scheduling. Workshop on Planning for Web Services., Trento, Italy (2003)
8. McIlraith, S., Son, T.: Adapting Golog for Composition of Semantic Web Services. In: Eighth International Conference on Knowledge Representation and Reasoning (KR2002), Toulouse, France (2002)
9. Levesque, H.J., Reiter, R., Lesperance, Y., Lin, F., Scherl, R.B.: GOLOG: A Logic Programming Language for Dynamic Domains. Journal of Logic Programming **31** (1997) 59–83
10. McCarthy, J., Hayes, P.J.: Some Philosophical Problems From the Standpoint of Artificial Intelligence. In Meltzer, B., Michie, D., eds.: Machine Intelligence 4. Edinburgh University Press, Edinburgh (1969) 463–502
11. McDermott, D.V.: Estimated-Regression Planning for Interactions with Web Services. (In: AI Planning Systems Conference (AIPS)) 204–211
12. McDermott, D.V.: A Heuristic Estimator for Means-Ends Analysis in Planning. (In: AI Planning Systems Conference (AIPS)) 142–149
13. Ghallab, M., Howe, A., Knoblock, C., McDermott, D., Ram, A., Veloso, M., Weld, D., Wilkins, D.: PDDL—The Planning Domain Definition Language (1998)
14. Fikes, R.E., Nilsson., N.J.: STRIPS: A new approach to the application of theorem proving to problem solving. Artifical Intelligence **2** (1971) 189–208
15. Pednault, E.: ADL and the state-transition model of action. Journal of Logic and Computation (1994)
16. Bacchus, F., Ady, M.: Planning with Resources and Concurrency: A Forward Chaining Approach. In: IJCAI. (2001) 417–424