

# Automated Planning in a Service-Oriented Architecture

Hilmar Schuschel, Mathias Weske  
Hasso-Plattner-Institute for Software Systems Engineering  
at the University of Potsdam  
Prof.-Dr.-Helmert-Strasse 2-3, 14482 Potsdam, Germany  
{Schuschel, Weske}@hpi.uni-potsdam.de

## Abstract

*Service-oriented computing is a new paradigm to improve the performance of information systems in environments demanding for flexibility and interconnectivity. Additionally, the support of intra- and inter-organizational processes has become crucial for the competitiveness of companies. Recent research approaches propagate to plan processes automatically to improve quality and flexibility. This paper discusses the application of AI planning for service composition in a service-oriented architecture. Three alternative levels of abstraction for service descriptions are defined and their impact on automated planning is investigated in regard to the provider and requestor role of service composers. It is shown that different abstraction levels of service descriptions are necessary to take full advantage of AI planning in a service-oriented architecture.*

**Keywords:** *service-oriented architecture, service description, automated process planning, AI planning*

## 1 Introduction

Today's business environments are characterized by an increasing demand for flexibility and interconnectivity. Information systems are supporting companies to handle this challenge. Therefore, new paradigms like service-oriented computing [12, 8, 4, 5] are emerging. A *service* is a commonly available resource with a published description. Additionally, support of intra- and inter-organizational processes is crucial for the competitiveness of companies [3]. Recent research approaches propagate the feature to plan processes automatically to improve quality and flexibility [1, 6, 7, 14, 18]. This paper discusses the application of automated planning in a service-oriented architecture and focuses on the abstraction level of service descriptions.

In a service-oriented architecture *service providers* pub-

lish descriptions of services they offer in *service repositories*. To find a service, *service requestors* search repositories for suitable services. Once a requestor has found a particular service, it is possible to directly negotiate with the provider to concretize its requirements on the service [11]. Complex tasks can require the execution of more than one service. In this case *atomic services* can be composed to processes. A *process definition* is a defined set of partially ordered *activities* intended to reach a goal. Executing the process implies changing a given situation in order to fulfill a company's goal. The information about this situation including all relevant documents is called a *case*. Examples of cases are an incoming purchase order that has to be handled or a sick patient who has to be cured. *Planning* is the generation of a process definition to reach a particular goal.

Traditionally, processes are planned manually by human experts. Recently, advances in the automation of business process planning have been made. Automated planning avoids the need for predefined process definitions and improves quality and flexibility while saving time and costs. For instance, in [1] ontologies of domain services and domain integration knowledge are used that serve as a model for workflow integration rules. *DY<sub>flow</sub>* [18] avoids the use of predefined process definitions and allows the dynamic composition of web services to business processes by applying backward-chain, forward-chain and data flow inference. Another approach dealing with automated composition of web services is described in [10]. Semantics for a subset of DAML-S [15] are defined in terms of a first-order logical language, to enable automated planning. In [6] the composition of single tasks or subgraphs of workflows is embedded in a case-based framework for workflow model management. In [7] the application of contingent planners to existing workflow management systems is discussed. Case individual process definitions and automated replanning are identified as two conceptual advantages of an integrated planning and coordination system in [14]. To generate process definitions some of these approaches use Artificial Intelligence (AI) planning algorithms, while others use

proprietary developed algorithms. Although AI planning algorithms have been in research for more than 30 years [2], advances in recent years make their application on real business domains promising [9, 16].

This paper discusses the application of AI planning for service composition in a service-oriented architecture. Provider of composed services – called *service composer* hereafter – automatically plan processes using AI planning techniques. Concerning the service-oriented architecture, a service composer has always two roles. On the one hand, its role is provider for requestors who want to execute a composed service and on the other hand its role is requestor of service used for its composition. An important question is, how detailed the published service descriptions should be and which aspects should be left to negotiation. Three alternative levels of abstraction for service descriptions are defined and their impact on automated planning is investigated in regard to the provider and the requestor role of a service composer. It is shown that different abstraction levels of service descriptions are necessary to take full advantage of AI planning in a service-oriented architecture.

In Section 2 the relevant foundations on AI planning algorithms are introduced. Section 3 describes the application of AI planning in a service-oriented architecture. Finally, in Section 4, results are discussed and additional conclusions are drawn.

## 2 Preliminaries - AI Planning

In this section foundations on AI planning are presented which are relevant for application in a service-oriented architecture. Furthermore, an example is introduced that is used throughout the paper.

Planning algorithms [2, 17] take a description of the current state of a case, a goal and a set of activities as input. The *goal* constitutes the desired state of the case. The means to transform the case from its actual state to the goal are the activities. An *activity* is a piece of work that forms one logical step within a process. A *domain* is the set of available activities. The conditions under which an activity can be executed are called *preconditions* and its impacts on the state of the case are called *effects*. *Planning* is the task of finding a partially ordered set of activities that when executed transforms the case from its current state to the goal. The description of this set of activities and their ordering – the *process definition* – is the output of the planning algorithm.

To illustrate the basic procedure of a planning algorithm and the relationship between its input and output, the planning of a simple process is presented. The example domain *Dom* consists of four activities *A*, *B*, *C* and *D* that are described using the Planning Domain Definition Language [13] (PDDL). The corresponding definition given in extracts:

```
(define (domain Dom)
...
(:durative-action A
 :duration (= ?duration 30)
 :condition (at start (x))
 :effect (and (at end (y))
              (at end (increase (costs) 50))))

(:durative-action B
 :duration (= ?duration 20)
 :condition (at start (x))
 :effect (and (at end (z))
              (at end (increase (costs) 20))))

(:durative-action C
 :duration (= ?duration 20)
 :condition (and (at start (x))
                 (at start (y)))
 :effect (and (at end (z))
              (at end (increase (costs) 10))))

(:durative-action D
 :duration (= ?duration 25)
 :condition (and (at start (x))
                 (at start (z)))
 :effect (and (at end (w))
              (at end (increase (costs) 10))))
)
```

Activity *A* has the precondition *x* and the effect *y*. For instance, *x* is the fact that a customer name is determined and *y* is the fact that the address of the customer is determined. Thus, activity *A* determines the address of a customer based on his name. This is the activities functionality. Besides functional properties, non-functional properties are specified: The costs for the execution of *A* are 50 and the time needed for execution is 30. To define a concrete planning problem, not only the domain has to be specified, but also the initial state and the goal. The corresponding PDDL definition given in extracts:

```
(define (problem T)
(:domain D)
...
(:init
 (x)
 (= (costs) 0)
)
(:goal (and (w) (y)
            (<= (duration) 70)))
(:metric minimize (costs))
)
```

The planning problem *T* is specified as follows: The initial state consists of fact *x* and the fact that the overall costs are zero. The goal is that the facts *w* and *y* hold and that the duration of the process is less or equal 70. Besides this, costs are defined as a metric to minimize. To complete the input

for a planning algorithm, two extra activities are introduced: an activity *Start* that has no preconditions and whose effect is the initial state of the case and an activity *Finish* that has the goal as a precondition and no effects. The task of a planning algorithm is to find a process definition that solves *T*. Process definition  $P_1$  depicted in Figure 1 is a solution of *T*. The activities are partially ordered in a way that the effects of preceding activities satisfy the preconditions of subsequent activities. For example *B* has the effect *z* that is needed by *D* as a precondition. Such a link between the effect of one activity that satisfies the precondition of another activity is expressed by a *causal link*, depicted as a dashed arrow. Every causal link implies an *ordering constraint*. Ordering constraints are also inserted to protect causal links. For example, consider *A* has an effect that negates effect *z* of *B*. In this case an additional ordering constraint would have to be inserted to avoid that *A* is executed between *B* and *D*. In the presented example there is no negation of a fact. Thus, no ordering constraint derives from protection of causal links. Process definition  $P_1$  is an optimal solution of *T* regarding the costs metric to minimize; the overall costs are 80. Process definition  $P_2$  depicted in Figure 1 also produces the facts *w* and *y*, if started in an initial state in which fact *x* holds. Thus it has the same functionality as  $P_1$ .  $P_2$  only differs in the non-functional properties: the overall costs are 70 and the duration of the process is 75. Since the execution time limit of *T* is not kept,  $P_2$  is not a valid solution of *T*.

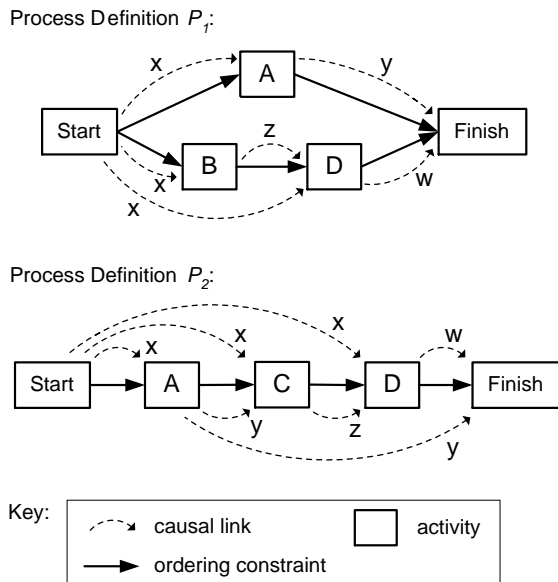


Figure 1. Process Definitions  $P_1$  and  $P_2$

### 3 Service Discovery and Negotiation

This section discusses the application of AI planning for service composition in a service-oriented architecture. In subsection 3.1 three alternative levels of abstraction for service descriptions are defined. Their impact on automated planning is analyzed from the provider perspective in subsection 3.2 and from the requestor perspective in subsection 3.3.

#### 3.1 Abstraction Levels of Service Descriptions

The following paragraphs specify three abstraction levels of service descriptions, starting with the most concrete one and ending with the most abstract one.

**Completely Described Services** The lowest level of abstraction is the complete description of a service. It covers the functional and non-functional properties of a service. If a provider wants to offer the same functionality with different combinations of non-functional properties it has to publish a corresponding number of services. For instance an expensive and fast service and a cheaper but slower, where both services are functionally equivalent.

**Functionally Described Services** An alternative to a complete description is to publish only the functional properties of a service. In this case the non-functional properties are negotiated directly with the service provider. The negotiation can be arranged in the way that the requestor tells the provider its requirements on some non-functional properties like time and security and the provider answers if this is possible at which price.

**Service Categories** An even more abstract way for a provider to describe its services, is publishing service categories instead of concrete services. A service category subsumes a set of related services the provider wants to offer. Nevertheless, the requestor has to know to which category the required service belongs. For this category the requestor finds a number of candidate providers in the repository that it can ask for the needed service. This approach has an analogy in the yellow pages of telecommunications. There is no functional (and non-functional) description of a specific service like "pipe will be fixed (in 2 hours for 80 Euro)", but the corresponding entry would be "plumber". This high level of abstraction is needed to limit the number of service entries in the repository. In the context of web services a typical service category would be "invoice processing services" that could aggregate a large number of invoice-related, but functionally different services.

### 3.2 Service Provider Perspective

In general, a service composer in its role as a service provider does not offer every possible process over the set of all available services, but only a subset. Thus, a service composer has to publish information on the services it offers, like any other service provider. The question here is, what information about the provided services of a service composer should be published in the repository. Or the other way around: what details should be left to negotiation with the service requestor, to take full advantage of the application of automated planning.

**Completely Described Services** To be able to publish complete service descriptions, a service composer has to anticipate what services requestors will demand and which combinations of non-functional properties they will need. Therefore, it has to plan concrete combinations of functional and non-functional properties of a service like demonstrated in Section 2, to determine whether the service can be composed and what non-functional properties can be offered. For example in Figure 1 two process definitions are shown:  $P_1$  with a duration of 45 and overall costs of 80 and  $P_2$  with a duration of 75 and overall costs of 70. This illustrates that a service composer can plan the same functionality with different combinations of non-functional properties. By planning in advance to publishing, process definitions exist for all published services. If a service is requested, the corresponding process definition can be taken to handle the request.

The integration of automated planning allows to plan an updated process definition, before the request is executed. Using manual planning, this would usually not be feasible due to the effort in time and costs. With automated planning the predefined process definition  $P_{old}$  planned before publishing, can be discarded. A new process definition  $P_{new}$  is planned, giving the planner the same functional and non-functional requirements, as for planning  $P_{old}$ . Nevertheless, the input of the planner can be different, because the domain may have changed. Thus, planning an updated process definition can be advantageous, if in the meantime between the planning  $P_{old}$  and the actual request, the repository got updated information on available services. Because of the high volatility of services in a service-oriented architecture [8] this can likely be the case. By taking up-to-date information into account when planning a new process definition, unavailable services can be replaced. Even if all services are still available, possibly a more optimized process definition can be planned by using services that were not available before. This can mean a simple replacement of one service or a structural change of the process definition. For example, in addition to the activities described in Section 2 an additional activity G becomes available:

```
(:durative-action G
  :duration (= ?duration 15)
  :condition (and (at start (x))
                 (at start (z)))
  :effect (and (at end (w))
              (at end (increase (costs) 15))))
```

Initial state and goal remain unchanged. Due to the changed domain,  $P_1$  depicted in Figure 1 is no longer the optimal solution. The process definition  $P_3$  shown in Figure 2 only has overall costs of 75. As shown in this example, changes in the set of available activities can lead to structural changes of the process definition. In this regard, planning updated process definition is more flexible than using predefined process definitions combined with late binding of services. To summarize, if complete service descriptions are published, requestors needs have to be anticipated and processes are planned in advance to publishing. Thus, the only advantage of automated planning is the option to individually update and optimize process definitions before execution.

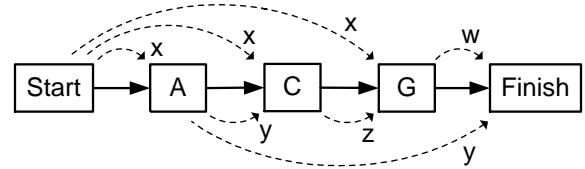


Figure 2. Individual Process Definition  $P_3$

**Functionally Described Services** Publishing complete service descriptions, a service composer has to plan every combination of non-functional properties of a given functionality in advance. With a growing number of combinations, this approach does not seem to be feasible. If only a selection of combinations is published, this incompleteness prevents providing optimal services that exactly match the customer needs. To avoid these disadvantages, a service composer can publish only a functional description of its services. In this case, the agreement on non-functional properties is postponed to negotiation. A requestor can find a composer for a needed service by the published functional description. The requestor gives the composer its exact needs on non-functional properties. Using automated planning, the service composer now plans an individual process definition exactly matching the needs of the requestor, based on up to date information of currently available services.

In the example of Section 2, a functional description for the planning problem only specifies that the initial state consists of fact  $x$  and the goal of the facts  $w$  and  $y$ . This information is published and used by the requestor to find the

service composer. For negotiation the requestor then gives the composer its exact needs on non-functional properties. For example the duration of the process may not exceed 80 and the costs should be minimized. Process definition  $P_2$  depicted in Figure 1 is a solution for this problem. The duration of the process is 75 and the overall costs are 70. To sum up, using functional service descriptions, it is still necessary to anticipate the functional needs of requestors. After negotiation on non-functional properties, an individually optimized process definition is planned. However it is still necessary to plan every process definition in advance to publishing, to check if the functionality can be realized without any restrictions on non-functional properties.

**Service Categories** A more flexible approach from the provider perspective is to only publish service categories. In this case, it is not necessary to anticipate the exact needs of requestors – neither the functional nor the non-functional. The service category of a service composer is derived from the set of services that can be used to build processes. A requestor simply searches in the repository for a suitable service category. Then for negotiation it asks the corresponding composer directly for a service by specifying the needed functional and non-functional properties. Using automated planning a service composer then tries to plan a process that meets the demands of the requestor.

For example, a requestor asks a service composer for a service with the following functional and non-functional properties: The initial state consists of fact  $x$  and the fact that the overall costs are zero. The goal is that the fact  $y$  holds and that the duration of the process is less or equal 50. Next to this, costs are defined as a metric to minimize. Let the set of available services be the domain specified in Section 2. Using automated planning, the the composer then generates a process definition with the activities  $B$  and  $D$  in sequence. The duration of the process is 45 and the overall costs are 30.

### 3.3 Service Requestor Perspective

In this section the perspective changes to the service composers role as a service requestor. Starting point is a service composer that wants to plan a process. Functional and non-functional requirements of the process are specified. To complete the input for planning, information on the available services – the domain – is needed. In this context, the service composer has the service requestor role.

**Completely Described Services** If the service descriptions in the repository are complete, all the domain information that is needed can be retrieved from the repository. To plan a process definition, information on the functional

and non-functional properties of the activities are taken into account as described in Section 2.

**Functionally Described Services** If only functional descriptions of services are published, planning becomes more difficult. The information on non-functional properties of services is missing in the repository, i.e. the input for the planner is not complete. Information on non-functional properties can be determined by direct negotiation with the service provider. Of course it is not feasible to negotiate with every provider of every service. One approach to solve this problem is a two step procedure: 1.) Planning without considering non-functional properties 2.) Negotiation with the providers of the services about non-functional properties. The major problems with this approach lies in the interdependencies between the non-functional properties of the services in the process definition.

Consider a composer  $Z$  wants to offer a functionality that can be realized by executing the services  $A$  and  $B$  in sequence. A non-functional requirement is that the execution time may not be longer than 30 seconds and the price should be minimized. The provider  $X_1$  of  $A$  and provider  $X_2$  of  $B$  published only a functional description of their services. Non-functional properties – in this example execution time and price – have to be agreed on by direct negotiation. To negotiate,  $Z$  tells a provider his requirement on execution time and the provider quotes a price. The problem is that  $Z$  does not have a fixed requirement for the execution time of  $A$  nor  $B$ .  $Z$  only knows that the total execution time must not exceed 30 seconds.  $Z$  could ask  $X_1$  and  $X_2$  for the price of an execution time limit of 15 seconds. This generally would not minimize the overall price.  $Z$  can try other combinations of time limits for  $X_1$  and  $X_2$  and compare the resulting overall costs to find an optimal process definition or at least to avoid the worst. Apparently, in this case many iterations of negotiations between the service composer and the providers of composed services become necessary. This high amount of communication between the service composer on the one hand and the repository and multiple providers on the other hand, makes frequent planning and therefore individual planning for each case unfeasible. Another problem is that the iterated negotiations with the providers are part of the planning process and thus have to be embedded in the planning algorithm.

**Service Categories** If only service categories instead of single service descriptions are published, a service composer has to take a completely different approach. In this case a top down approach instead of a bottom up approach is adequate. Instead of searching through combinations of single services, the composer has to speculate on partitions of the problem, so that he can find services solving the partial problems. To the best of our knowledge, no planning

algorithm supporting such a top down approach, automation of planning based on service categories appears not to be feasible.

## 4 Discussion

In this paper, the application of AI planning in a service-oriented architecture was discussed. Planning, publishing, discovering and negotiating composed services was described and analyzed, based on three alternative abstraction levels of service descriptions: complete, functional and category. Thereby, the two roles of a service composer – provider and requestor – were taken into account. In its role as a service provider, a service composer using automated planning benefits from the ability to plan updated process definitions, without regard to the abstraction level of the published service description. Nevertheless, the abstraction level of the service description is crucial for the feature of generating individually tailored process definitions. While completely described services do not allow any individualization, functionally described services facilitate planning that is individually optimized with regard to non-functional properties. In contrast to the other abstraction levels, services categories do not require detailed anticipation of needed services and thus avoid the need for predefined process definitions. Functional and non-functional requirements of requestors are taken into account when planning an individual process definition.

In its role as a service requestor, a service composer depends on completely described services. Applying current AI planning algorithms, only completely described services can be used. It was illustrated that generally planning becomes very complex and communication intensive, if services are only published with a functional or category descriptions. To sum up, a service composer relies on completely described services to specify the input domain. On the other hand, service categories enable the best customization of composed services. Thus, only a repository that supports multiple abstraction levels of service descriptions allows to take full advantage of automated planning. Furthermore, nested composition of services is unfavorable. To build composed services based on also composed services, they have to be completely described. Thus, no individualization can take place and it can not be guaranteed, to find an optimal solution.

As already mentioned in section 1 some approaches use proprietary developed algorithms to automate planning, while this paper is focused on AI planning. Nevertheless, one key result can be broadened to automated planning in general: The application of automated planning allows individual customization of process definitions. This advantage of automated planning is lost, if complete service descriptions are published.

**Acknowledgments** The authors would like to thank Jens Hündling and Harald Meyer for their valuable suggestions.

## References

- [1] S. A. Chun, V. Atluri, and N. R. Adam. Domain knowledge-based automatic workflow generation. In *Proceedings of the 13th International Conference on Database and Expert Systems Applications*, pages 81–92. Springer-Verlag, 2002.
- [2] R. E. Fikes and N. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.
- [3] M. Hammer and J. Champy. *Reengineering the corporation*. Harper Collins Publishing, New York, 1993.
- [4] J. Hündling and M. Weske. Web services: Foundation and composition. *EM - Electronic Markets*, 2003.
- [5] F. Leymann, D. Roller, and M. Schmidt. Web services and business process management. *IBM Systems Journal*, 41(2), 2002.
- [6] T. Madhusudan and J. L. Zhao. A case-based framework for workflow model management. In *Proceedings of the Business Process Management Conference*, volume 2678 of *LNCS*, pages 354–369. Springer, 2003.
- [7] MD R-Moreno, D. Borrajo, and D. Meziat. Proces modelling and AI planning techniques: A new approach. In *Proceedings of the 2nd International Workshop on Information Integration and Web-based Applications Services*, 2000.
- [8] C. Mohan. Dynamic e-business: Trends in web services. In *Proceedings of the third VLDB workshop on Technologies for E-Services*, number 2444 in *LNCS*. Springer, 2002.
- [9] K. Myers and P. Berry. The boundary of workflow and ai. In *Proceedings of the AAAI-99, Workshop on Agent-Based Systems in the Business context*, 1999.
- [10] S. Narayanan and S. McIlraith. Simulation, verification and automated composition of web services. In *11th International World Wide Web Conference*, 2002.
- [11] J. O’Sullivan, D. Edmond, and A. ter Hofstede. What’s in a service? towards accurate description of non-functional service properties. *Distributed and Parallel Databases Journal - Special Issue on E-Services*, 12(2-3):117–133, 2002.
- [12] M. Papazoglou and D. Georgakopoulos. Service-oriented computing. *Communications of the ACM*, 46(10):25–28, 2003.
- [13] Planning Competition Committee. PDDL – The planning domain definition language. *AIPS-98*, 1998.
- [14] H. Schuschel and M. Weske. Integrated workflow planning and coordination. In *14th International Conference on Database and Expert Systems Applications*, volume 2736 of *LNCS*, pages 771–781. Springer, 2003.
- [15] The DAML Services Coalition. DAML-S: Web service description for the semantic web. In *The First International Semantic Web Conference (ISWC)*, 2002.
- [16] D. S. Weld. Recent advances in AI planning. *AI Magazine*, 20(2):93–123, 1999.
- [17] Q. Yang. *Intelligent Planning: A Decomposition and Abstraction Based Approach*. Springer, 1997.
- [18] L. Zeng, B. Benatallah, H. Lei, A. H. H. Ngu, D. Flaxer, and H. Chang. Flexible composition of enterprise web services. *Electronic Markets - Web Services*, 13(2), 2003.