

# A Formal Model for Mixed Initiative Service Composition

Jan Schaffner

Harald Meyer

Mathias Weske

Hasso-Plattner-Institute for Software Systems Engineering,  
Prof.-Dr.-Helmert-Str. 2-3,  
14482 Potsdam, Germany,  
{jan.schaffner,harald.meyer,mathias.weske}@hpi.uni-potsdam.de

## Abstract

*Automated service composition has been investigated thoroughly during the last five years. Although it promises to alleviate the difficulties of manual service composition, it will only work if complete and correct service specifications are available.*

*In this paper, we present a third approach – semi-automated composition – based on three mixed initiative features which we have derived from an industry case study with SAP. These features, filter inappropriate services, check validity, and suggest partial plans, are defined on the basis of a common formal model. Related approaches, in contrast, are limited to supporting individual mixed initiative features. To show the applicability of our approach, we have developed a prototypical implementation. Our results show that our mixed initiative approach significantly eases the modeling of service compositions.*

## 1 Introduction

In service oriented architectures, the ability to compose services is key. Nowadays, the task of creating service compositions is either performed completely manual or, as research suggests, fully automated. Performing service composition manually is a time consuming, error-prone task that needs to be carried out by experts. Fully automated composition [15, 6, 11, 1] works by applying automated planning techniques to semantic service specifications. Complete and correct semantic service specifications are required in order to produce valid plans. What's more, complete automation might lead to legal difficulties as no human being is involved and responsible for creating and maintaining a particular service composition. We therefore propose a third approach that leaves the user in control. In our semi-automated composition approach, service compositions are

created manually, but features from automated service composition and discovery are used to support the user. In this paper, we present three so-called mixed initiative features and provide a formal foundation for their application.

Mixed initiative and semi-automated service composition are intensively researched topics at the moment [5, 4, 3, 10]. In previous work [8, 9], we identified the three core features that support users in modeling service compositions. *Filter inappropriate services* reduces the set of services to those that are relevant in the current modeling situation. *Check validity* assists the user in modeling semantically correct service compositions by testing if the preconditions of the services in the composition are satisfied and whether the composition contains redundant services. *Suggest partial plans* adopts techniques from automated service composition in order to not only suggest individual services, but to find sub-processes that can fill gaps in the service composition that is currently being modeled.

Related approaches in semi-automated composition are limited to supporting individual mixed initiative features. Thus, the formal models underlying these approaches cannot be used to describe a complete semi-automated composition environment providing all three mixed initiative features. In this paper, we present our approach together with a complete, holistic formal model with which we describe all aspects of semi-automated service composition.

In order to validate our approach, we implemented a modeling environment supporting all three mixed initiative features in cooperation with a large vendor of business applications. To ensure the industrial relevance of our approach, we have used their business processes and services, which span across multiple functional areas of enterprise computing. Our results show that semi-automated composition based on our three mixed initiative features significantly eases the process of creating composed services.

This paper is organized as follows: In next section, we introduce a scenario that will be used throughout this paper to illustrate the proposed mixed initiative functionality. In

Section 3, we provide the necessary formal foundations for the introduction of the mixed initiative features in Sections 4 to 6. Section 7 concludes the paper, discussing related work and outlining future research directions.

## 2 Scenario Overview

The following scenario is taken from Duet<sup>1</sup>, a recent software product by SAP and Microsoft. Duet extends the reach of SAP's business processes to a large number of users by embedding them into Microsoft's Office environment. We extracted the process flow among the ERP Web services Duet is built upon. We are using the process as a case study for the semi-automated composition environment developed in Jan Schaffner's Master's thesis [7]. The leave request scenario consists of two parts: First, an employee files a leave request. Second, his manager approves or denies this request. Therefore, the two roles "employee" and "manager" participate in the leave request process. Due to length constraints, we limit ourselves to the part of the process in the role of the employee. The scenario is depicted in Figure 1 using the Business Process Modeling Notation (BPMN [14]). The activities in the diagram correspond to Web service operations. To describe the semantics of the operations, we extended the BPMN syntax with annotations for inputs and outputs of services.

Before employees file a leave request, he will typically try to get an overview of his time balance and suitable dates for the leave. Duet will collect this information when the leave request application is opened. Therefore, Duet will invoke the following three Web service operations. *Read Employee Time Account* returns the time balance of an employee consisting of paid vacation, overtime, and sick leaves. The operation takes an employee object uniquely representing the employee and a key date, for which the balances are returned, as input. It might be the case that an employee has recently filed other leave requests which are not yet processed. *Find Leave Request By Employee* returns an employee's pending leave requests, so that he or she can consider them together with the time balance. The operation takes an employee object as input. A leave request is approved by the line manager of the employee filing it. In some cases, a different approver or no approval at all is necessary. *Find Leave Request Allowed Approver by Employee* returns the employee object corresponding to the allowed approver for the leave request. It takes an employee object as input.

The information retrieved by the four service operation described above is visualized in Duet and the employee can decide on a day or a period browsing his Outlook calendar. This yields the sequential invocation of the two following

operations. Before a leave request is created in the ERP system, it must be checked for plausibility. *Check Create Leave Request* takes the same inputs as the operation that creates the leave request, which are an employee object, the leave period and the leave type. If the check is successful, the operation returns a positive result. After the plausibility check has successfully been completed, *Create Leave Request* creates the leave request in the ERP system.

## 3 Formal Foundations

In the next sections we will show how the mixed initiative features are derived from the presented scenario. We will describe how and when they can be used and provide a formal specification for each feature. In this section, we provide the necessary formal foundations. We begin with introducing the notion of *service operations*. They are the basic building blocks from which service compositions are built. Each service operation has a semantic specification of its functionality:

**Definition (Service Operation)** A service operation is a tuple  $op = (I, O, Pre, Eff)$  consisting of:

- *I*: List of input parameters consisting of variables
- *O*: List of output parameters consisting of variables
- *Pre*: The precondition of the service is a logical expression and must be satisfied in order to invoke the service.
- *Eff*: The effect of the service is a logical expression. It describes the changes to the current state resulting from the invocation of the service.

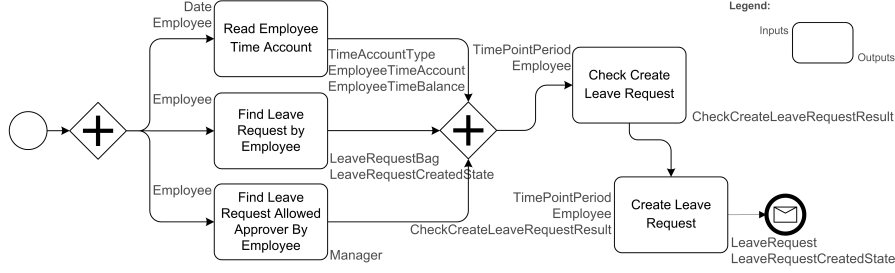
The syntactic interface consists of the input and output parameters. The semantic interface specifies the precondition that must hold true in order to invoke the operation and the effect specifying the state changes resulting from invoking the operation. Both precondition and effect are logical expressions. The logical expressions are sets of literals defined over the relations, functions, constants, and variables. The input and output parameters are typed variables used also in these logical expressions. Formally, this is defined as:

**Definition** The literals  $l \in L$  defined over a language  $L = (R, F, C, V)$  with the set of relations  $R$ , the set of functions  $F$ , the set of constants  $C$ , and the set of variables  $V$  are inductively defined. The function  $type : C \cup V \rightarrow O$  returns the type of a constant or variable.

$T$  is the set of terms, where  $T_{ground} \subseteq T$  contains only ground terms. Terms are defined as follows:

- A variable  $v \in V$  is a term ( $v \in T, v \notin T_{ground}$ ).

<sup>1</sup><http://www.duet.com>



**Figure 1. The leave request scenario**

- A constant  $c \in C$  is a term ( $c \in T_{ground}$ ).

The set of literals  $L$  is defined by:

- If  $r \in R$  is a relation and  $t_1, \dots, t_n \in T$  are terms then  $r(t_1, \dots, t_n) \in L$
- If  $l$  is a literal, so is  $\neg l$  ( $l \in L \Rightarrow \neg l \in L$ )

A logical expression  $e$  is a set of literals. It can be separated into  $e^+$  containing the positive literals and  $e^-$  containing the negated literals. States are a set of negation free literals ( $e^- = \emptyset$ ). The function  $facts : E \rightarrow C \cup V$  retrieves all the facts (variables and constants) for a given logical expression.

Finally, we introduce the notion of a *service composition*. We distinguish between service operations and control flow constructs:

**Definition** A service composition is a triple  $C = (OP, CF, E)$  with

- $OP$  : Set of service operations,
- $CF = AS \cup AJ \cup OS \cup OJ$  : Set of control flow constructs  $AS$  : AND-splits,  $AJ$  : AND-joins,  $OS$  : OR-splits, and  $OJ$  : OR-joins
- $E \subseteq (OP \cup CF) \times (OP \cup CF)$  : Edges connecting operations and control flow constructs.

A composition may not contain control flow cycles.

With these basic definitions of what a service is, how service functionality can be expressed, and how services can be composed, we can start describing the mixed initiative features.

## 4 Filter Inappropriate Services

The number of service operations available as building blocks for the composition can be extremely high. In the context of SAP, for example, the central repository contains

more than 1000 services. This results in a complexity that is hard to oversee. Particularly if compositions are to be created by users from a non-technical background, a modeling tool for service compositions should filter the set of available services.

When the leave request is to be created from scratch, the tool will first retrieve all available services. The modeler begins with adding the role “employee” to the composition by selecting this role from a list of all available roles (e.g. “supplier”, “customer”, “manager”). Our tool then assumes the implicit availability of a variable of the complex type “employee”, representing the person who takes part in the business process in this role. The tool is now able to filter the list of available service operations to those that require an employee object as an input. The operations in the service repository are grouped around so-called enterprise services. In our example, the modeler would therefore now expand the “Time and Leave Management” enterprise service and select the first three operations depicted in Figure 1. As there are no dependencies among these activities, the user connects the operations using a parallel control flow. This is shown in Figure 2.

*Filter inappropriate services* filters those services that are not invocable in the current state. To do so, we need to know what service invocability means. A service is invocable if all its input parameters are available and its precondition is satisfied. Formally, invocability is defined as:

**Definition** (Invocable) A service operation  $op = (I, O, Pre, Eff)$  is invocable in a state  $S$  if

- $Pre^+ \subseteq S$  and
- $\nexists l, \neg l \in Pre^-, l \in S$  and
- $I \subseteq facts(S)$

The current state is given by the effects and outputs of all preceding service operations in the service composition. The state transition is defined as follows:

**Definition** (Service invocation) Given a state  $S$  and an invocable service operation  $op = (I, O, Pre, Eff)$ , The

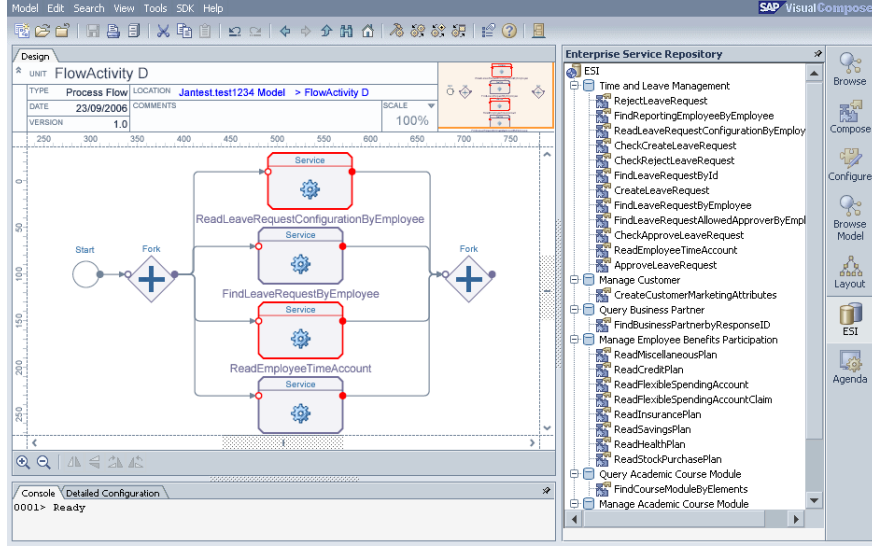


Figure 2. Screenshot of the modeling tool

state transition function  $\gamma : S \times OP \rightarrow S$  is given by  $\gamma(S, op) = S \cup \text{eff}^+ \setminus \{x \mid \neg x \in \text{eff}^-\}$

As defined above, states contain only positive literals. This means that we only add the positive literals from the effect to the state. The negated literals are then used to remove all literals from this state for which a negated literal exists in the effect. During our experiments, we learned that filtering all services which are not invocable is too restrictive. For fully automated composition, such restrictive filtering is appropriate, because the algorithm may not create invalid service compositions. In the mixed initiative environment we target at, in contrast, it might be the case that human modelers want to add services to the composition although they are currently not invocable. Therefore, we introduce the notion of nearly invocable services. A service is nearly invocable to the degree  $k$  if at most  $k$  input parameters are missing. Formally:

**Definition** (Nearly invocable) Given a state  $S$  and a service operation  $op = (I, O, Pre, Eff)$ ,  $op$  is nearly invocable to the degree  $k$  if  $(I = \{i_0, \dots, i_m, \dots, i_{m+k}, \dots, i_n\}$  with  $n = |I|$ ):

- $\forall i_i \in I, i < m, i_i \in \text{facts}(S)$ ,
- $\forall i_i \in I, i > k + m, i_i \in \text{facts}(S)$ ,
- $\forall l \in Pre^+$  with  $l = r(x_0, \dots, x_i), r \in R$  and  $\forall x_i \notin \{i_m, \dots, i_{m+k}\}: l \in S$ , and
- $\forall \neg l \in Pre^-$  with  $l = r(x_0, \dots, x_i), r \in R$  and  $\forall x_i \notin \{i_m, \dots, i_{m+k}\}: l \notin S$ .

If an operation is invocable to a degree  $k$ , then it is also invocable to all degrees  $j$ , with  $1 \leq j \leq k$ .

The first two conditions specify exactly what was described above: it might be the case that  $k$  inputs are not satisfied in order for the operation to be invocable. The last two conditions are necessary to relax the satisfaction requirement for the precondition. Only those literals in the precondition not containing one of the missing inputs need to be satisfied in the state.

Using the notions of invocable and nearly invocable services, the modeler is now able to retrieve more service suggestions through the filtering mechanism by clicking on the merge node of the parallel split. Amongst others, our tool will suggest the operation *Check Create Leave Request* as an invocable service. The modeler adds it to the composition and creates a link between the merge node and the operation.

## 5 Check Validity

When human modeler have full control over the modeling, they are likely to make mistakes. It is therefore necessary to check the semantic validity of the process. As opposed to syntactic validity checking based on structural correctness criteria (e.g. soundness [13]), semantic validity is based on semantic descriptions of individual activities. When semantic descriptions for the activities in a process are available, we are able define correctness criteria for processes on the semantics level. Semantic validation should be interleaved with the actual modeling of the composition by informing the user about problems with the composition in an unobtrusive way. Such problems, which can be seen as unresolved issues, arise from activities in the composition which violate one or more aspect of a set of criteria for

semantic validity.

We formalize the semantic validity into four different criteria. The first two criteria define what it means if a service operation input or precondition is not satisfied:

**Definition (Unsatisfied Input)** An input  $i \in I$  is unsatisfied for a service operation  $op = (I, O, Pre, Eff)$  in a state  $S$  if  $i \notin facts(S)$ .

**Definition (Unsatisfied Precondition)** The precondition  $Pre$  of a service operation  $op = (I, O, Pre, Eff)$  is unsatisfied in a state  $S$  if

- $\exists l \in Pre: l \notin S$  or
- $\exists \neg \in Pre: l \in S$ .

These definitions are inverse to the invocability definition from above. While a service composition violating the first criterion will also be syntactically ill-formed, a service composition violating the second criterion might very well be syntactically correct. It only affects the composition on the semantical level. This means that it is possible to technically invoke a service composition containing operations with unsatisfied preconditions while this is not possible if operations have unsatisfied inputs.

The third criterion defines the relevance of a service operation inside a composition. This is necessary because it can be difficult for human modelers to determine whether each service operation is required in a complex service composition. A service operation in a composition is relevant if one of its outputs is consumed by a successor operation in the composition. If the operation does not have no successor operation, we assume that it is relevant. Formally:

**Definition (Relevance)** A service operation  $op' = (I', O', Pre', Eff')$  in a service composition is relevant if

- $\exists op'' = (I'', O'', Pre'', Eff'')$  and  $op' \xrightarrow{e^*} op''$  with  $\exists x, x \in O' \wedge x \in I''$  or
- $\nexists op'' = (I'', O'', Pre'', Eff'')$  and  $op' \xrightarrow{e^*} op''$  (final activity)<sup>2</sup>.

It might be the case that several operations in a service composition produce the same output. Such activities are potentially redundant. Detecting redundancy in a fully automated fashion is very complex: not only the outputs of the redundant operations, but also the effects must exactly match. This is rarely the case. Instead, operations without matching outputs and precondition are often redundant. We therefore define potential redundancy as a weak criterion: An operation is redundant if another operation produces the same output. Formally:

<sup>2</sup> $op' \xrightarrow{e^*} op''$  denotes that there is a path in the composition connecting  $op'$  and  $op''$ .

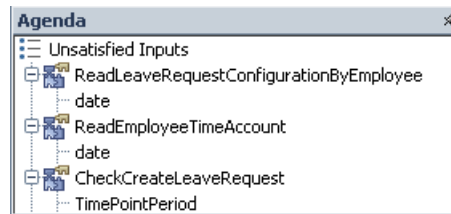
**Definition (Potential redundancy)** A service operation  $op' = (I', O', Pre', Eff')$  in  $OP$  is potentially redundant if another operation  $op'' = (I'', O'', Pre'', Eff'')$  exists with  $o' \in O', o'' \in O'' type(o') = type(o'')$ .

This potential redundancy needs to be addressed by the human modelers. They can either resolve the potential redundancy or flag it as not redundant. This mechanism leads to many potential redundancies. A potential extension could include the ranking of possible redundancies based on the overlapping of operation outputs and to only alarm the user if the match is higher than a predefined threshold. In summary, semantic validity comprises the following aspects:

**Definition (Semantic validity)** A service composition is semantically valid if

- it does not contain activities with unsatisfied inputs or preconditions,
- all activities in the composition are relevant, and
- it does not contain potentially redundant activities that have not been flagged as explicitly not redundant.

As the last step, the modeler adds the nearly invocable operation *Check Create Leave Request*. The tool highlights operations for which problems are tracked. As the added operation is not invocable, but nearly invocable, one input type is missing. The tool therefore marks the operation with a red border. This can also be seen in Figure 2, where two out of four activities are highlighted. By clicking on the *Check Create Leave Request* operation, the user can open a panel showing its input and output types as inferred from the pre- and effects. The user sees that all input types of the operation are currently available in the composition, except *TimePointPeriod*, which is also highlighted using red color in this drill-down view. The user can also get an overview of all current problems with the composition by looking at the agenda, depicted in Figure 3.



**Figure 3. Agenda summarizing the problems with the composition**

The missing parameter *TimePointPeriod* represents the date or period for which the employee intends to request a leave. As our scenario has been taken from Duet, this data

is provided by Microsoft Outlook after a the user selects a date from the calendar. In our example, the modeler therefore creates a human activity (modeling a task such as marking a period in the calendar) that produces a *TimePointPeriod* output. The modeler connects the human activity with the *Check Create Leave Request* operation. The coloring of the operation and the *TimePointPeriod* input type in the parameter view disappear and the issue is removed from the agenda.

## 6 Suggest Partial Plans

Automated planners [15, 6, 11, 1] plan according to an algorithmic planning strategy, such as for example forward- or backward chaining of services. Human planners, in contrast, will not always behave according to this schema when modeling composed service. Users might have a clear idea about some specific activities that they want to have in the process, without a global understanding how the whole will fit together as a process. For example, they start modeling the composed service by adding some operations and chaining them together, and then continue with a non-invokable operation that is intended to be in a later stage of the composition. In such and similar cases, it is desirable for the user to let the editor generate valid service chains that connect two unrelated activities. The task of generating such service chains can be reduced to solving a planning problem using automated planners.

**Definition** (Planning problem) A planning problem  $P = (s_0, s_g, SD)$  is a triple consisting of the initial state  $a_0 \in E_{state}$ , the goal  $g \in E$  and a service domain  $SD$ . A **service domain**  $SD = (OP, O)$  consists of a set of service operations  $OP$  and ontology describing the concepts used to specify services.

In order to suggest a partial plan, it is therefore necessary to determine the initial state and the goal state for the planning problem. The initial state can be determined by adding up all the effects of the services leading to the gap. The initial state then contains all the available information. The goal state can be determined from the preconditions of the services succeeding the gap. But of course, the information already known in the initial state can be removed from the goal state. These are preconditions already satisfied by preceding operations. Additionally, preconditions of services after the gap can be satisfied by other services also succeeding the gap but preceding this service. These preconditions can also be removed from the goal state. Using the generated planning problem an automated planner can be used to fill the gap.

In the last step the modeler resolved a problem with the *Check Create Leave Request* operation. If the user clicks

on the operation to refresh the filtered list of available services, the tool will suggest the *Create Leave Request* operation. From the perspective of the user, this is the final operation. However, the modeler might not be familiar with the fact that a specific check operation needs to be invoked in order to create a leave request in the system. He then directly selected the *Create Leave Request* operation after the merge node depicted in Figure 2. The modeler also creates the human activity producing the *TimePointPeriod* and links it to the *Create Leave Request* operation. Now, the modeler tries to create a link between the merge node of the parallel flow and *Create Leave Request*. The tool will detect that the set of outputs up to the merge node does not satisfy the inputs of *Create Leave Request* (the type *CheckCreateLeaveRequestResult* is missing). The tool instantly queries the semi-automated composition engine which detects that the insertion of the *Check Create Leave Request* operation would satisfy this open information requirement. The user is prompted whether the *Check Create Leave Request* should be inserted. The modeler approves this suggestion and the composition is complete.

## 7 Conclusion

In this paper, we presented a holistic formal model for semi-automated composition based on three mixed initiative features, *filter inappropriate services*, *check validity*, and *suggest partial plans*. All three features proved to be necessary to support human modelers in the error-prone and time-consuming task of modeling service compositions.

### 7.1 Related Work

In the following, we give a brief overview of related work in the field of semi-automated composition.

Sirin, Parsia and Hendler [10] present Web Service Composer. Their tool allows creating executable compositions of Web services that are semantically specified with OWL-S. In order to create a composed service, the user follows a strict backward chaining approach, starting with selecting a Web service that has an output producing the desired result of the composition from a list of available services. Next, the user interface presents additional lists connected to each OWL input type of the service producing the end result. In contrast to the first composition step, these lists do not contain all available services: They contain only those services that generate an output satisfying the particular input type they are connected to. As a consequence, the plans constructed with the tool are not always optimal. For example, when one service operation delivers two outputs each of which satisfies a different input of a downstream service, this service operation occurs twice in the composed service.

The approach, thus, realizes only a subset of the *filter inappropriate services* mixed initiative feature, as it does not the discovery of “nearly invocable” services. Also, there is no formal model underlying the presented functionality.

IRS-III [3] includes a tool that supports a user-guided interactive composition approach by recommending component Web services according to the composition context. Their approach uses the Web Services Modeling Ontology (WSMO) as the language to describe the functionality of the Web services. Similar to Web Service Composer, the available services are filtered at each composition step. It is not possible to further shorten the filtered list based on non-functional properties. Our approach, in contrast, interprets the intended role of a service in form of a nonfunctional property when the weightings are assigned. Regarding the supported mixed initiative features and the underlying formalization, the same restrictions as to Web Service Composer apply.

Kim, Spraragen and Gil introduce CAT (Composition Analysis Tool) [4]. At each composition step, CAT provides a list of suggestions about what to do next. These suggestions resolve errors and warnings, which are also displayed. The idea is that consequently applying suggestions will produce a “well-formed” workflow as a result. The authors therefore introduce a set of properties that must be satisfied by all operations in the composition in order for the process to be well-formed. The tool does not provide filtering functionality nor the ability to create partial plans, but realizes the *check validity* mixed initiative features. While a formal definition of the presented functionality is given in [4], it is limited to describing their criteria for semantic validity and could hardly be extended to describe all three mixed initiative features.

PASSAT (Plan-Authoring System based on Sketches, Advice, and Templates) [5] is an interactive tool for constructing plans. Similar to CAT, PASSAT is not directly concerned with the creation of composed services, but its concepts can be mapped into the context of service composition. PASSAT is based on hierarchical task networks (HTN) [12]. In HTN planning, a task network is a set of tasks (or service calls) that have to be carried out as well as constraints on the ordering of these tasks. The HTN based approach imposes top-down plan refinement as the planning strategy the user must adhere to: The user can start by adding tasks to a plan and refine them by applying matching HTN templates. A template consists of a set of subtasks that replace the task being refined, as well as the postconditions of applying individual tasks and the entire template. The only mixed initiative feature supported by PASSAT is *suggest partial plans*. A formalization of the approach is defined on the basis of HTN, which could possibly be extended to model all three mixed initiative features. However, the strict top-down approach imposed by HTN is

not suitable for the use in modern modeling environments, where flexible and ease-of-use play a central role.

## 7.2 Future Work

One direction for future work is to identify quality metrics for the usefulness of the suggestions that our system presents to the user. These metrics could, e.g. for the *filter inappropriate services* feature, depend on the number of service operations in a given repository, as well as on specific queries and the number of suggested operations for each query. The queries are, in this context, a given state of the composition, as it is outlined in the modeling tool at a given point in time, along with one of the methods realizing the mixed initiative features described in [9]. Another interesting experiment would be to use incomplete semantic descriptions of the services and to measure how this affects the quality of the suggestions.

To increase the value of the presented semi-automated composition approach from a modeler’s perspective, it would be conceivable to go further than only suggesting individual service operations for inclusion in the process. In the case of SAP’s service repository, for example, many operations are semantically coupled with others in the sense that one operation is dependent on the other operation in order to be invocable. To optimally assist the user in modeling service compositions, the system could, therefore, suggest groups of services that are preconfigured with control flow dependencies. This can be seen as an extension of both the *filter inappropriate services* and the *suggest partial plans* mixed initiative feature. To improve the suggestion on the level of the individual operations, it might also be useful to track which operations the users select when creating a composition. The collected data can, then, be used to improve the service suggestions. When a modeler selects an operation A, for example, the system could suggest additional operations that other users have also selected after they selected A. The same technique could be applied to improve the suggestion of groups of operations, as described above.

We enhanced our prototypical modeling tool with basic performance optimizations, such as using asynchronous calls for complex queries to the backend to ensure a non-blocking system behavior. Yet, our prototypical implementation could be improved from a performance point of view. Optimization strategies for the semi-automated composition process are, therefore, subject to further research. We will investigate possibilities to partition the ontology into multiple disjoint parts in order to improve the response time of the system. In doing so, the reasoning on the concepts in the ontology could, accordingly, be distributed across multiple CPUs. Furthermore, we see a potential for performance improvements regarding the algorithms realizing the mixed

initiative features. Most of these algorithms traverse the composition graph and are currently realized using recursion. As one future research activity, we will investigate to what extent it is possible to find iterative algorithms solving these problems.

While we have shown that we are able to obtain reasonable support in the modeling of service compositions using the service repository and typical SAP business processes, validation with end-users is still an open point. It was planned to evaluate the modeling tool together with some of SAP's smaller partners. However, we have not yet been able to conduct user-interviews to further validate our approach.

## References

- [1] D. Berardi, D. Calvanese, G. De Giacomo, and M. Mecella. Composition of Services with Nondeterministic Observable Behavior. In *Proceedings of the 3rd International Conference on Service Oriented Computing (ICSOC'05)*, volume 3826 of *Lecture Notes in Computer Science*, pages 520–526. Springer, 2005.
- [2] F. Hakimpour, D. Sell, L. Cabral, J. Domingue, and E. Motta. Semantic Web Service Composition in IRS-III: The Structured Approach. In *7th IEEE International Conference on E-Commerce Technology (CEC 2005), 19-22 July 2005, München, Germany*, pages 484–487. IEEE Computer Society, 2005.
- [3] J. Kim, M. Spraragen, and Y. Gil. An Intelligent Assistant for Interactive Workflow Composition. In *IUI '04: Proceedings of the 9th international conference on Intelligent user interface*, pages 125–131, New York, NY, USA, 2004. ACM Press.
- [4] K. L. Myers et al. PASSAT: A User-centric Planning Framework. In *Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space*, Houston, TX, USA, 2002. AAAI.
- [5] M. Pistore, F. Barbon, P. Bertoli, D. Shaparau, and P. Traverso. Planning and Monitoring Web Service Composition. *Lecture Notes in Computer Science*, 3192:106–115, Jan 2004.
- [6] J. Schaffner. Supporting the Modeling of Business Processes Using Semi-Automated Web Service Composition Techniques. Master's thesis, Hasso-Plattner-Institute for IT Systems Engineering, University of Potsdam, Potsdam, Germany, 2006.
- [7] J. Schaffner and H. Meyer. Mixed Initiative Use Cases For Semi-Automated Service Composition: A Survey. In *Proceedings of the International Workshop on Service Oriented Software Engineering (IW-SOSE'06), located at ICSE 2006, 27-28 May, 2006, Shanghai, China*. ACM Press, New York, NY, USA, May 2006.
- [8] J. Schaffner, H. Meyer, and C. Tosun. A Semi-automated Orchestration Tool for Service-based Business Processes. In *Proceedings of the 2nd International Workshop on Engineering Service-Oriented Applications: Design and Composition*, Dec 2006.
- [9] E. Sirin, B. Parsia, and J. Hendler. Filtering and Selecting Semantic Web Services with Interactive Composition Techniques. *IEEE Intelligent Systems*, 19:42–49, 2004.
- [10] E. Sirin, B. Parsia, D. Wu, J. Hendler, and D. Nau. HTN Planning for Web Service Composition Using SHOP2. *Journal of Web Semantics*, 1(4):377–396, 2004.
- [11] A. Tate. Generating Project Networks. In *Proceedings of the Fifth Joint Conference on Artificial Intelligence, Cambridge, MA, USA*, pages 888–893. Morgan Kaufmann Publishers, 1977.
- [12] W. M. van der Aalst. Verification of Workflow Nets. In *ICATPN '97: Proceedings of the 18th International Conference on Application and Theory of Petri Nets*, pages 407–426, London, UK, 1997. Springer-Verlag.
- [13] S. A. White. Business Process Modeling Notation, Working Draft (1.0). Technical report, The Business Process Modeling Initiative, Aug 2003.
- [14] L. Zeng, B. Benatallah, H. Lei, A. H. H. Ngu, D. Flaxer, and H. Chang. Flexible Composition of Enterprise Web Services. *Electronic Markets*, 13(2), 2003.