Automated Holistic Service Composition: Modeling and Composition Reasoning Techniques

Wei Zhu, Farokh Bastani, I-Ling Yen
University of Texas at Dallas
{wxz094120, bastani, ilyen}@utdallas.edu

Jicheng Fu
University of Central Oklahoma
jfu@uco.edu

Yuqun Zhang
SUSTech, China
zhangyq@sustc.edu.cn

Abstract. Many real-world applications are complex, involving many user “choices”, such as different functionalities, different ways to achieve a goal, etc. Conventional automated service composition models do not consider such potential choices, or simply consider them independently. Also, existing service composition models do not model exceptions and automated composition approaches require that after an exception, the original system goal should still be achieved. This may not be feasible for some exceptions. Thus, the service composition model should also consider alternate goals after exceptions occur. In this paper, we first define the concept of multi-functionality and develop a holistic service composition model. Since most existing composition reasoning techniques can only handle a single functionality, we extend them and develop new algorithms for automated holistic service composition. A case study system is used to illustrate how our approach automatically generates a holistic workflow for a system with multiple functionalities.

Keywords. Service-oriented computing, automated service composition, planning algorithms, multi-functionality systems.

1 Introduction

With the increasing provisioning of services on the Internet, service composition becomes more and more challenging. A lot of research works have investigated automated service composition to save cost and time for composition tasks. SHOP2 [1] and OWLS-Xplan [2] are widely used automated service composition tools. They are based on Hierarchical task network (HTN) techniques, which are considered as domain dependent solutions due to their need for knowledge about how to decompose tasks. Some other automated service composition works use search-based planning [3] [4], which may incur a higher time complexity, but do not require domain knowledge. The algorithms proposed in these works are in fact the same as a large class of planners such as GraphPlan [5], Fast Forward Planner [6], LPG [7], Fast Incremental planner [8], Model Based Planner [9], etc.

Existing automated service composition literature have omitted some important issues. First, existing techniques consider a single system goal (functionality). However, modern systems are complex and the system may have to be specified by different functionalities. For example, consider a floor cleaning system. The system may offer carpeted floor cleaning, hardwood floor cleaning, tile floor cleaning, etc. All these floor cleaning services start by ordering service, transporting service which transports the people and equipment to the location where the service is to be performed, furniture moving service, and vacuuming service. After vacuuming, depending on the service ordered, different services are performed. It is possible to compose services for each function and obtain multiple workflows for the system. However, from the cleaning service example, we can see that there are quite a lot of overlapping services in the workflows for multiple functionalities. Thus, it is better to specify multiple functionalities for the overall system and compose one workflow with branches for achieving all the desired functionalities. This solution can reduce composition effort and generate a well-integrated workflow.

The second issue is related to services with multiple effects and exceptions. The output of a service may be used as a control parameter to determine what the subsequent processes should be. Consider developing a retail store management workflow that is activated upon store closing. First, a patrol service goes across the store to ensure that there are no customers remaining in the store. Then, a store closing service closes all doors and counters. Next, an inspection service provided by a robotic or manned cart is activated to navigate through the aisles to make inspections. This inspection service may give different outcomes, such as found misplaced items, found some products with low shelf stocks, or found spoiled or broken items. Depending on the outcome of this inspection service, different subsequent sub-workflows with different goals will be invoked to handle the problems.

Similar to having multiple effects, a service may raise exceptions during execution. Exceptions can also result in different effects on the system. Generally, when a service is defined, the potential exceptions that may be raised by the service are also defined. When a service is selected and composed into a workflow, its exceptions should be taken care of to ensure that the system is holistic.

Automated service composition techniques have been developed for handling multi-effect services and exceptions [10] [11], but some gaps still remain. One gap is that formal service definition models, such as OWL-S [12] and WSMO [13], do not have a specific mechanism for specifying the multiple effects or exceptions of a service. Without a proper specification model, the techniques for handling them becomes ad hoc. Another important gap is that when handling exceptions, all works require the system to achieve the same original goal. However, in many cases, after an exception is raised, the original goal can no longer be satisfied and a new goal should be specified. Thus, we need to have a model that supports the specifications of different goals for the exceptions.

The third gap in automated service composition is the consideration of alternative paths toward a goal. Most existing planning and automated service composition techniques only derive one path from the initial condition to the goal state. But

978-1-5386-0752-7/17 $31.00 © 2017 IEEE
DOI 10.1109/ICWS.2017.70

IEEE computer society

Authorized licensed use limited to: Southern University of Science and Technology. Downloaded on October 05,2021 at 11:44:21 UTC from IEEE Xplore. Restrictions apply.

in a holistic system, there may be multiple ways for achieving the system goals for a certain functionality. Sometimes these multiple ways should be presented to the users to provide flexible choices. For example, an online shopping system may be composed of browsing, add to cart, checkout, payment, and delivery services. It is desirable to offer different payment and delivery methods and leave the choices to the users. The service composition model and automated composition techniques should be able to construct a workflow with choices and identify the user interaction points for making the choices.

In this paper, we consider the problem of automated holistic service composition. Here, "holistic" refers to the composition of a complete system. It is necessary to consider multiple functionalities of the system, multiple effects and exceptions of services, different goals for exceptions, and multiple methods as user choices for achieving the goals. We build a comprehensive model and integrated techniques to facilitate automatic service composition to obtain a holistic workflow for the desired system. Our contributions include:

(1) To facilitate a formal treatment of the holistic composition problem, we extend the classical OWL-S service model with multi-effect and exception definitions. Also, we define a separate "system" model to facilitate the more precise specifications of the multi-functionality composition problems. Different from a composite service, a system can have a goal structure, including the goals for multiple functionalities, special goals for exceptions when the regular functionalities cannot be achieved, and the goals for choices of methods for achieving some of the functionalities.

(2) We develop the automated service composition procedure for composing workflows for holistic systems. The procedure is designed to generate a workflow that can achieve multiple functionalities, provide choices of multiple methods for achieving some system functions, take care of multiple effects of the services, and handle exceptions to achieve the original or new goals as desired.

(3) Based on the multi-functionality model, we extend existing planning techniques and develop a new planning algorithm, Multiple Functionality Planning (MFP), to achieve efficient multi-functionality planning. We use a case study system to illustrate how our approach can generate a holistic workflow for a system with a multi-functionality specification.

The rest of this paper is organized as follows. In Section 2, we introduce our extended model for automated holistic service composition. The composition reasoning algorithms for achieving holistic composition are presented in Section 3. Section 4 presents a case study to illustrate how to use our model for holistic service composition problem specification and how our composition reasoning algorithm is used to obtain a holistic workflow. Section 5 concludes the paper.

2 A Holistic Service Composition Model

In OWL-S, a service can be defined by its (I, O, P, E), where I and O are the input and output of the service, P is the precondition, when satisfied, the service can be executed, and E is the effect, which specifies the condition that will be satisfied after the execution of the service. However, there are some shortcomings in the OWL-S model as well as some other existing service composition models. We develop a holistic service and composition model which extends OWL-S to facilitate automated composition of holistic workflows. Our extensions are discussed in the following subsections.

2.1 Holistic Service Specifications

In a comprehensive service specification, it is necessary to consider that the service may cause different effects that should be specified separately to facilitate composition reasoning. Also, it is necessary to consider the specification of exceptions that the service may raise.

First consider multiple effects that a service may cause. In fact, multiple effects can be specified in the OWL-S model by a predicate in "disjunctive normal form". Consider the inspection service (InspectS) in the workflow given in the introduction for store management upon closing. The input to the service may be a "situation" that needs to be taken care of and the effects can further define the situation output by literals "item-misplaced", "item-low-shelf-stock", "item-spoiled", "item-broken" and "no-situation". An OWL-S specification for the effects of InspectS will be

"item-misplaced" ∨ "item-low-shelf-stock"
∨ "item-spoiled" ∨ "item-broken"

As can be seen from the example, the effect specified by the disjunctive predicate can be used for constructing conditional branches in the workflow. For "item-misplaced", services for determining the locations of the items and for moving items back to their locations should be composed to achieve the function of "no-misplaced-items". For "item-low-shelf-stock", subsequent sub-workflow will check whether the items have sufficient inventory in the store, and if so, restock the shelves from the inventory; otherwise, order the items to replenish the inventory and the shelves. Similar handling can be made for situations "item-spoiled" and "item-broken".

Handling multiple effects of a service have been widely considered in automated service composition literature [4] [14] [15]. The technique is to create multiple "virtual services" to represent one concrete service during composition reasoning, one for each disjunctive clause in the effect predicate. For the example above, we need to create four virtual services for InspectS. InspectS-1 has effect "item-misplaced", InspectS-2 has effect "item-low-shelf-stock", and so on. All 4 virtual services have the same input, pre-condition, and output specifications as the original service, but with different effects.

To facilitate virtual service creation when converting a service composition problem into a planning problem, it is better to specify the disjunctive effects separately so that there is no need to perform low level effect predicate analysis every time the service is considered as a candidate service for composing a system. Thus, we modify the OWL-S service model to support the specification of multiple effects.

Next, we consider exceptions of a service. When a service raises an exception, the functionality of the service is not fulfilled since the execution is disrupted. We will discuss exception specifications later. Here we only consider that each

Authorized licensed use limited to: Southern University of Science and Technology. Downloaded on October 05,2021 at 11:44:21 UTC from IEEE Xplore. Restrictions apply.

service can be associated to a set of exceptions and we extend the OWL-S service model to support such specifications.

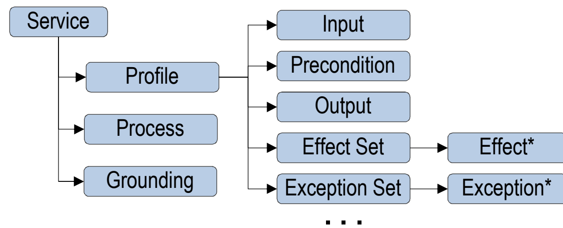

Figure 1. The extended service model.

A partial view of the extended service model for holistic service specification is given in Figure 1. Note that since the extensions are all under the service profile, the figure only shows the Profile class for the service.

The service profile defined in OWL-S is extended. It still has Input, Output, and Pre-condition classes. The extended model has a class "Effect Set", which specifies the set of effect clauses for the service. Effects can be instantiated by one or more "Effect" classes, where each Effect is the same as the original Effect specification in OWL-S. Note that even though a service may have multiple effects, there is only a single output specification (output has to be a unified one in order to allow proper use of the service). Multiple effects specify the properties of the output parameters or the system state in different situations. The extended model also has an "Exception Set" class under the service profile, which can be instantiated by one or more "Exception" classes.

To facilitate the discussion of the reasoning algorithm for service composition, we also define the notations to represent the holistic service model.

Definition 1 A web service s is defined by a tuple $<I(s), O(s), P(s), ES(s), XS(s)>$ where

$I(s)$ is the set of input parameters for s;

$O(s)$ is the set of output parameters for s;

$P(s)$ is the preconditions of s;

$ES(s)$ is the set of effects after the execution of s and $E_i(s)$ is the i-th effect clause in $ES(s)$. $ES(s)$ can be defined by positive effects $ES^+(s)$ and negative effects $ES^-(s)$. Negative effects indicate the state facts that are no longer true after executing s. If st is the system state before executing service s, then the resulting state after executing s will be $(st - ES^-(s)) \cup ES^+(s)$.

$XS(s)$ specifies the set of exceptions s may raise and $X_i(s)$ is the i-th exception defined in $XS(s)$. □

2.2 Exception Specifications

Exception is a very important concept in software and systems. Some widely used service models, such as OWL-S and WSMO, do not support the specifications of exceptions. Some web service models consider exceptions, but they do not offer the formal specification mechanism for them [3] [15]. Here we provide a model for exception specification which can be integrated with the service and system specification models to facilitate holistic service composition.

Similar to the formalism of IOPE in OWL-S, we need to have the "effect" specification for exceptions in order to enable composition reasoning. Such effect needs to be considered carefully. Some segments of a service may have been executed when an exception is raised and, hence, it looks as though the effect of an exception depends on at which point of execution it is raised. However, such concerns are internal to the service. We assume that each service can handle their internal exceptions and the only visible exceptions are the external ones. The execution effects that need to be cleaned up are taken care of by the internal exception handlers. The external exceptions are those that require external workflows to handle and their effects can be defined by externally known literals.

Another consideration about an exception is the goal for exception handling. For most exceptions, it is desirable to still achieve one of the original system functionalities. But in some cases, the original system goals (of one of the functionalities) can no longer be satisfied after an exception. Thus, a different goal should be specified for the exception. For example, consider a tour booking site which helps users book tickets for various types of tourism trips. One cruise workflow has one goal predicate "cruise trip booked". It helps users book airline tickets, cruise tickets, hotels, rental cars, etc., to obtain complete trip bookings. At the checkout, the payment service may raise an external exception "payment failure" after multiple internal attempts to get different credit cards and different payment methods. This exception has to be external because it cannot be handled by the payment service. Also, the original system functionality of trip booking can no longer be satisfied, and a new functionality "cleanup" should be associated to this exception and causes the system to cancel all the reservations. Thus, for some exceptions, the composition model should support the specification of new functionalities that the system should reach after the exceptions.

Based on the exception specification requirements, we define the exception ontology in Figure 2.

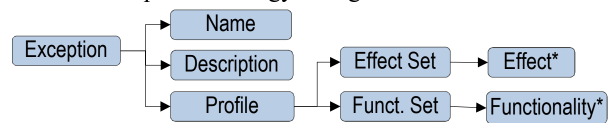

Figure 2. The exception model.

An exception has a Name, a Description, and a Profile. The Profile includes the "Effect Set" class which can have one or more "Effect" specifications and the "Functionality Set" class, which links to one or more functionalities of the system.

Based on the model given in Figure 2, we also define the notations for relevant exception specifications.

Definition 2 An exception e can be defined by a tuple $<ES(e, s), EGS(e)>$ where $EGS(e)$ is the set of functionalities in which one of them should be achieved after exception e is raised. $EG_i(e)$ is the i-th functionality in $EGS(e)$.

$ES(e, s)$ is the set of effects that hold when exception e is raised by service s, and $E_i(e, s)$ is the i-th effect clause in $ES(e, s)$. Note that even though we assume that $ES(e, s)$ does not depend on internal states of s, but it is still service dependent. On the other hand, $EGS(e)$ includes the system level goals (goals for regular functionalities and special goals

Authorized licensed use limited to: Southern University of Science and Technology. Downloaded on October 05,2021 at 11:44:21 UTC from IEEE Xplore. Restrictions apply.

for exceptions) that should be achieved after e is raised, no matter which service raises it. □

2.3 Holistic System Specifications

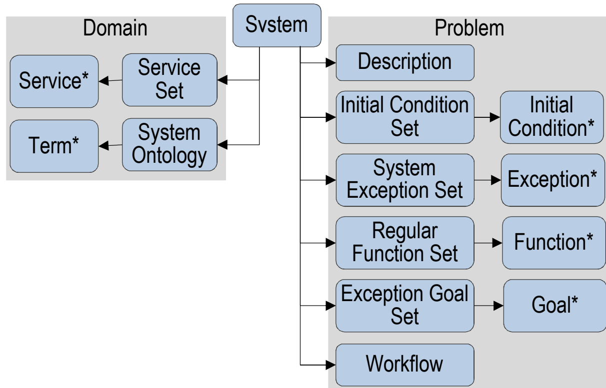

Figure 3. The System model.

In OWL-S, a composite service to be composed can have its IOPE specifications, which are used as the rules to govern the composition. Such model can facilitate the specification of value-added services and hierarchical composition. WSMO treats service and composition separately. The composite service to be composed is referred to as a service, while the available concrete services are referred to as Web services. In many applications, service composition is for rapidly developing and deploying a comprehensive system, not just a value-added service. Though a composite service and a comprehensive system have similarities, there are differences in their specifications. Thus, we define a separate "System" model (like the service model in WSMO) to support a clear definition of a holistic composition problem. The ontology for the "System" model is given in Figure 3.

We consider the "System" model in two views, the composition domain view and the composition problem view. The composition domain view of the system provides information that are common to all composition problems in the domain. It has the "Service set" (similar to the Web service class in WSMO) and the "State Ontology" (similar to the ontology class in WSMO) which defines the terms used for system state definitions. The "Service set" is the set of all services that can be considered for composition. All the terms in the IOPE specifications for the services in the "Service set" are defined in the "State Ontology". Mediators in WSMO are also an important class in service composition, but they are beyond the scope of this paper and are omitted in our model.

The composition problem view of the system specifies the system to be composed, including a "Description" class, the "Initial Condition set" class, the "System Exception set" class, the "Regular Functionality set" class, and the "Exception Goal set" class. The "Regular Functionality set" includes one or more "Functionality" of the system and each "Functionality" consists of one or more "Goal" subclasses. The "Exception Goal set" also consists of one or more "Goal" subclasses. The "Workflow" class specifies the workflow that, starting from the initial conditions, can achieve the functionalities of the system. Before composition, the Workflow is null. After composition, the workflow is grounded to a concrete one. Next, we define the composition domain in the System model.

Definition 3 The composition domain can be defined as a tuple $<SS, STS>$, where SS is the set of all the services available for compositions in the domain and s_i is the i-th service in SS. STS is the set of terms (literals, etc.) used by the services and the composition problems to define the relevant states, such as those used in preconditions, effects, exceptions, input and output data types, etc., of services. □

In Definition 4, we define the tuple that specifies the relevant attributes for a specific composition problem in the System model under a composition domain.

Definition 4 A target system to be composed under the composition domain $<SS, STS>$ can be defined by a tuple $<IS, FS, SXS, XGS, WF>$, where IS is the initial condition set for the target system. FS is the set of regular functionalities for the target system and $FS = \{F_1, F_2, \ldots F_N\}$, where N is the number of functionalities to be achieved and F_i is the i-th functionality in FS. F_i can be considered as a goal state, It could have conjunctive and/or disjunctive goal clauses. Since modern planners can plan for any form of the goal state, we consider F_i as one integrated goal.

SXS is the set of exceptions that may be raised in the system and X_i is the i-th exception in SXS. Note that SXS is defined after the regular composition (without exception) is defined.

XGS is the set of goals for the exceptions that are not in FS and XG_i is the i-th goal in XGS. XGS is supposed to be composition independent, but may be defined or validated after regular composition definition to ensure that there are no missing goals under exceptions.

WFS is the set of workflows defined during the composition process for the target system. □

Note that the literals used in the definition of IS, XS, FS, XGS are defined in STS.

In this paper, our discussion only involves a single composition problem under one composition domain. Thus, we do not specify which composition domain and which target system the tuples are for.

There are a few semantics in the "System" model that need to be clarified and also compared with the "Service" model. First, consider the initial conditions IS of the system, which have a quite different semantics from the pre-conditions of a service in the holistic composition model. Note that the workflow being constructed should be able to handle all possible initial conditions. Essentially, the initial conditions of the system can be viewed as the effects of a special service, which takes "True" as the precondition and generates effects equivalent to the initial conditions of the system. Thus, similar to the effects of a service, the initial conditions should be represented in a disjunctive normal form. Each conjunctive clause in the predicate becomes an "Initial Condition" in IS.

Now consider the regular system functionalities. The "Functionality set" includes one or more functionalities to be achieved by the system. These functionalities can be achieved

Authorized licensed use limited to: Southern University of Science and Technology. Downloaded on October 05,2021 at 11:44:21 UTC from IEEE Xplore. Restrictions apply.

separately by separate branches in the system workflow, but each of them has to be achieved. This is different from the "Effect set" specification in the service model. Consider the set of system functionalities $FS = \{F_1, F_2, \dots F_N\}$. Also, consider the effect set of a service s, $ES(s) = \{E_1(s), E_2(s), \dots, E_n(s)\}$, where n is the number of effects s has. $ES(s)$ is expressed as:

$$E_1(s) \vee E_2(s) \vee \dots \vee E_n(s)$$

But the goals for the functionalities of the system do not have the same semantics as service effects. If we consider the multiple functionalities as $F_1 \vee F_2 \vee \dots \vee F_N$, then it will lead to the derivation of a workflow that only satisfies one of the system functionalities. Neither can the system functionalities be specified as $F_1 \wedge F_2 \wedge \dots \wedge F_N$ because it is not the intention that the multiple system goals are satisfied at the same time (and it may not be possible to satisfy these multiple functionalities at the same time). A closer semantics for the system functionalities may be described as

If <$cond$> then F_1
elsif <$cond$> then F_2
…
else F_N

where each <$cond$> is independent and can be any condition to be derived in the holistic workflow or to be set as a user selection point.

Also, each individual effects in the effect set of a service has to be a conjunctive clause while an integrated goal for a regular functionality or a special goal for some exceptions can be any logic predicates.

We also need to clarify the exception scopes in the system model. Some exceptions are specific to individual services, some exceptions are common to many services, and yet some exceptions may be raised by the system itself. For example, a "timeout" exception may be raised by all services with interactions to users. An "interrupted" exception may be raised by the system to stop the current workflow in execution. This can happen, for example, when the system offers a cancel button which can be clicked by the user at any time, resulting in the termination of the current service in execution. An exception that is common to multiple services should have a single specification and should be associated to the individual services that have this exception. This can avoid duplicated derivations of the exception handling workflows. System exceptions should be associated to the system, not to services.

As discussed earlier, each exception should be associated to some goals. The goals of an exception, namely, $EGS(e) = \{EG_1(e), EG_2(e), \dots\}$, should satisfy the following constraints:

For all i, $EG_i(e) \in (FS \cup XGS)$, and the semantics for $EGS(e)$ should be $EG_1(e) \vee EG_2(e) \vee \dots$. Specifically, each goal in the goal set of e can be from the goals for regular functionalities of the system or the special goals for exceptions. The exception handling workflow for handling exception e just needs to achieve one goal predicate in e's goal set.

2.4 Handling Multiple Methods

The goal for automated service composition is to derive a "system", not just a "plan". Thus, it should consider not only multiple functionalities of the system, but also alternative ways for achieving some of the functionalities. One way to derive a system workflow to cover alternative methods for achieving a functionality is to let the composition reasoning algorithm find all possible paths for achieving a functionality and incorporate them in the workflow. However, not all alternative paths are desirable choices to be included in the system workflow and it is difficult to automatically determine which choices should be incorporated. Thus, we consider to incorporate multiple method choices as multiple functionalities at the system design time and use multi-functionality composition reasoning to derive the alternative paths in the workflow.

Generally, one goal predicate is specified for a system functionality. For the choices of different methods for achieving a system functionality, we add additional method-related predicates in its goal predicate. For the purchasing process example, the functionality goal predicate is

"items delivered" ∧ "payment succeed"

The alternative delivery methods can be specified by method-related predicates such as "home delivery", "store pickup", etc. Thus, the new functionality set FS of the purchasing process would include F_1 = "home delivery" ∧ "items delivered" ∧ "payment succeed" and F_2 = "store pickup" ∧ "items delivered" ∧ "payment succeed", etc.

Generally, the IOPE definitions of many services have already incorporated the method-related predicates to support flexible workflow derivations. Thus, the above method can derive a workflow with multiple branches for using multiple methods to achieve the same functionality. When we just want to achieve a functionality via any method, then we only need to specify the goal predicate for the functionality. For the purchase process example, if we specify the goal as "items delivered", then only one of the methods will be selected by the reasoning process. If we need to achieve a functionality with method choices, then the goal specification for the functionality should include both the method-related goal predicate and the functionality-related goal predicate.

3 Reasoning for Holistic Composition

Based on the service composition models defined in Section 2, we leverage traditional planning approach and develop new composition reasoning algorithms to achieve automated holistic service composition. The service composition problem we discuss in this Section is <IS, FS, SXS, XGS, WF> under the domain <SS, STS>. In section 3.1, we focus on building the basic workflow for the system with multiple goals. In section 3.2, we provide a systematic approach to handle exceptions (external ones). In section 3.3, we discuss the planning algorithm for the multiple functionality service composition problem.

3.1 Basic Multi-Functionality Reasoning

First, we discuss the algorithm for construction a workflow WF_1 for the basic multi-functionality problem. In WF_1, multiple functionalities will be reached with branching paths,

Authorized licensed use limited to: Southern University of Science and Technology. Downloaded on October 05,2021 at 11:44:21 UTC from IEEE Xplore. Restrictions apply.

but it does not consider exceptions. The steps of the algorithm are sketched as follows.

1. The first step is to identify the functionality set FS for the system to be composed. The goal predicate for each desired functionality is added to FS. For each functionality F_i in FS, if it is desirable to consider alternative methods to achieve F_i, then combine the method-related predicates with F_i as new goal predicates, i.e., add $mp_j \wedge F_i$, for all j, to FS, and remove the original F_i from FS, where mp_j is the predicate for the j-th alternative method for achieving functionality F_i.

2. Consider the initial condition IS. If IS has a single clause, then it will be used as the initial state. If IS has multiple disjunctive clauses, then construct an "initial service" s_0 and add s_0 into the service set SS. Service s_0 should have its precondition $P(s_0)$ set to predicate "Initial" and its effects $ES(s_0)$ is set to IS. IS is then set to "Initial". Predicate "Initial" should be unique so that it does not duplicate with the pre-conditions or effects of any service s_i in SS. The purpose of doing so is to let the multi-conditional initial state to be processed uniformly as a multi-effect service, instead of having to have a separate special processing method for it.

3. For each service s_i in SS, if s_i has multiple effects, decompose it into virtual services $vs_{i,1}$, $vs_{i,2}$, … and replace s_i in SS by its virtual services $vs_{i,1}$, $vs_{i,2}$, ….

4. Formulate the planning problem as $P = (SS, IS, FS)$. Here SS is the updated set of services (updated in Steps 2 and 3). IS is the initial state, which could be the predicate "Initial". FS is constructed in Step 1.

5. Use DMFP (discussed in 3.3) to reason for P and obtain the multiple-functionality plan, which is a weak plan in the sense that some conditional branches have not been considered yet. Convert the plan into the first workflow draft WF_1. During plan to workflow conversion, we need to consider the branches due to the multiple functionalities $F_1, F_2, \ldots, F_N$. These functionalities are multiple choices offered to the users and a user interface should be constructed to allow users to make the choices. Let $W_1 = (w_1, w_2, \ldots, w_{m_1})$ denote the list of m_1 services used in WF_1. Let $w_{br_1}, w_{br_2}, \ldots \in W_1$ be the services after which there are multiple branches for reaching some of the functionalities in FS and assume that w_{br_1} is closest to the beginning of the workflow among w_{br_i}, for all i. We create a user interface service $Fchoice$ with output $choice$ for choosing among $F_1, F_2, \ldots, F_N$ and insert $Fchoice$ right after w_{br_1}, i.e., the first branching point. Also, after each w_{br_i}, we insert a conditional node in WF_1 to test $choice$ to see whether it is equal to the corresponding functionality of the branch.

6. This step handles the conditional branches needed due to multiple effects of some services in W_r, where r is initialized to 1 and increased in each round.

6a. Let $W_r = \{w_1, w_2, \ldots, w_{m_r}\}$ denote the bag of m_r services used in WF_r (not ordered). For each $w_i \in W_r$, if w_i is a virtual service, and w_i is constructed from a concrete service s_j, and s_j has virtual services $vs_{j,1}$, $vs_{j,2}$, …, then add $vs_{j,1}$, $vs_{j,2}$, … into V, where V is the bag of virtual services that has not been processed. Since each $w_i \in W_r$ has already been processed (i.e., a subworkflow has been constructed for w_i to reach the goal or to reach a state in WF_r), remove w_i from V.

6b. For each virtual service $vs_{j,l}$ in V (where $vs_{j,l}$ is a virtual service of s_j), formulate the planning problem

$$P_{j,l} = (SS, IS_{j,l}, FS)$$

to obtain the sub-workflow $WF_{r,j,l}$ for $vs_{j,l}$'s conditional branch to reach the goal. SS and FS are defined earlier. Here we derive $IS_{j,l}$, for all $vs_{j,l}$.

Let st_{w_i} denote the state before w_i in WF_r is executed. From the initial state, there may be one or more paths in WF_r that reach st_{w_i} and let $br_{w_i,1}, br_{w_i,2}, \ldots$, denote these paths. Let $(w_{i,x,1}, w_{i,x,2}, \ldots, w_{i,x,nb_{i,x}})$ be the sequence of $nb_{i,x}$ services forming path $br_{w_i,x}$, $w_{i,x,l} \in W_r$, for all x, l. Also, let $st_{i,x,l}$ denote the state before $w_{i,x,l}$ is executed. For each $br_{w_i,x}$, we have $st_{i,x,1} = IS$ and $st_{i,x,l+1} = (st_{i,x,l} - ES^-(w_{i,x,l})) \cup ES^+(w_{i,x,l})$, $1 \le l < nb_{i,x}$. Then, $st_{i,x,nb_{i,x}}$, for all x, can be derived. Subsequently, we can obtain the state before w_i is executed, i.e., $st_{w_i} = \bigcup_x st_{i,x,nb_{i,x}}$.

Let $st_{j,l}'$ denote the effect state after executing $vs_{j,l}$ in WF_r and $st_{j,l}' = (st_{w_i} - ES^-(vs_{j,l})) \cup ES^+(vs_{j,l})$. Finally, we have $IS_{j,l} = st_{j,l}'$. Next, $MRPMF$ or $SGEMF$ can be used to reason for $P_{j,l}$ to obtain the sub-workflow $WF_{r,j,l}$.

6.c. After 6b, all virtual services in $W_r = \{w_1, w_2, \ldots, w_{m_r}\}$ have been processed, i.e., if w_i is a virtual service of s_j, then the sub-workflows $WF_{r,j,l}$ for all s_j's virtual services $vs_{j,l}$, for all l, to reach the goals have been constructed. Now merge $WF_{r,j,l}$ into WF_r by:

For each w_i in W_r and w_i is a virtual service of s_j:

(i) Replace w_i by s_j, and

(ii) Add the following conditional branch

if $Ef(s_j) = Ef(vs_{j,l})$ then $WF_{r,j,l}$.

after s_j for each virtual service $vs_{j,l}$ of s_j.

Let the new workflow constructed from this step be WF_{r+1}.

7. Note that all virtual services in W_r have been processed. But there still may be unprocessed virtual services in WF_{r+1} if there are virtual services in $WF_{r,j,l}$. Thus, we have to continue the virtual service processing. Now set r to r+1 and go back to Step 6a till the new V from WF_r is empty.

8. Let WF_B denote the basic plan constructed after exiting the loop above. Also, W_B is the set of all the services in WF_B. It is possible that a service in SS is used more than once. In this case, we consider them as different services in W_B because they will be treated differently when handling their exceptions.

3.2 Process Exceptions

From the workflow WF_B constructed up to now, services that have the potential of raising external exceptions can be identified based on their exception set definitions (XS). Also, at design time, the sets of system exceptions SXS and the set of exception goals XGS should have been defined. At this stage, manual intervention can be introduced if desirable to ensure that the system exceptions in XS have been properly identified. Also, with the identified exceptions, now the

Authorized licensed use limited to: Southern University of Science and Technology. Downloaded on October 05,2021 at 11:44:21 UTC from IEEE Xplore. Restrictions apply.

designer can check whether the corresponding exception goal set XGS is complete and if not, complete it. Moreover, manual decisions on exception goals can be made for some exceptions as desired because some exceptions may have their known goal sets. For each exception e_j, if its goals are known, then define the goal set $EGS(e_j)$ for them, where $EGS(e_j) \subseteq FS \cup XGS$. Otherwise, by default, the goal set for e_j is $FS \cup XGS$.

We now perform composition reasoning for each exception e_j raised by service w_i with effect $ES(w_i, e_j)$. We construct the sub-workflow WF_{e_j} and merge it into WF_B as follow.

1. Formulate the planning problem

$$P = (SS, IS_{e_j}, EGS(e_j))$$

to obtain the workflow WF_{e_j} to handle exception e_j, where $EGS(e_j)$ is given in the specification of e_j and IS_{e_j} can be determined in a similar way as the initial state derivation for conditional branches discussed in Section 3.1 and

$IS_{e_j} = (st_i - ES^-(w_i, e_j)) \cup ES^+(w_i, e_j)$.

2. Use a traditional planner to reason for P and obtain the sub-workflow $WF_{e_j,1}$.

3. Apply Step 6 in Section 3.1 repeatedly to handle all the conditional branches for multi-effect services in $WF_{e_j,1}$ and obtain the final sub-workflow WF_{e_j}.

4. Merge WF_{e_j}, for all e_j, into WF_B for handling e_j.

The above procedure should be performed for all exceptions of all services in the workflow WF_B. Let WF_{XS} denote the final workflow constructed that can handle all the exceptions defined in $XS(w_i)$ for all services $w_i \in W_B$.

Next, we perform composition reasoning for each system exception in SXS. System exceptions are defined globally and may be raised at any time during workflow execution. Thus, we need to consider the state when it is raised in order to support proper exception handling. Also, generally each system exception e_j would have a clearly defined goal set $EGS(e_j) \subseteq FS \cup XGS$. Similar to the exceptions for the services, we can construct the sub-workflow WF_{e_j} for each exception $e_j \in SXS$ and merge it into WF_{XS}. Since the construction of WF_{e_j}, $e_j \in SXS$, is similar to the steps defined above, we do not repeat all the steps, but only discuss the step that is different, namely, Step 1, below.

1. Formulate the planning problem

$$P = \left(SS, IS_{e_j}, EGS\left(e_j\right)\right)$$

Assume that w_i is in execution or to be invoked when exception e_j is raised. Let st_{w_i} denote the state before w_i is executed and st_{w_i}, for all w_i, can be derived the same way as discussed in 6b of Section 3.1. We need to ensure that $EGS\left(e_j\right)$ is reached no matter when e_j is raised so we have $IS_{e_j} = \vee_{w_i} st_{w_i}$ (here we assume that if w_i is in execution and gets interrupted by e_j, w_i will clean itself up).

Let WF_X denote the workflow derived based on WF_{XS} and after processing all the exceptions in SXS. In other words, WF_X is the workflow that can handle all the exceptions.

3.3 The MFP Algorithm

We have constructed the multi-functionality composition problem and now we discuss how to efficiently derive the multi-functionality plan. In traditional planners, only one goal predicate is considered (the goal can have multiple propositions composed conjunctively and/or disjunctively). From the initial state, applicable actions are applied to generate a new state and form a plan graph. The expansion continues till the goal can be satisfied by the current state. Then, a backward search is performed to find the sequence of actions that can actually lead to the goal state.

If the multi-functional goals (MFG) is one predicate, then it can be treated the same as the classical planning problem. For example, if MFG is in fact composed disjunctively, then, during graph expansion, we simply check whether the goal predicate for one of the functionalities is satisfied. If MFG is composed conjunctively, then we check whether the current state can satisfy all the goals of all the functionalities. However, in the correct way, we can have one functionality of MFG satisfied in one state and another functionality satisfied by another state. MFP is designed based on this principle.

In MFP, the goals of multiple functionalities is tentatively considered in disjunctive form. Thus, during graph expansion, the planner breaks as long as one of the functionalities is satisfied. After the first break, backward search is performed from the goal state till it reaches the initial state. Now the first plan is formed. Next, we remove the already achieved functionality from MFG and continue graph expansion from the last state (where the planner breaks) till one of the remaining functionalities in MFG is satisfied. After the new break, backward search is again performed, but from the new goal state, till it reaches any state in the existing plan or the initial state. If it reaches the initial state, then the final workflow start the branching from the initial state. If it reaches a state in the existing plan, then the workflow branches at that point. Graph expansion and backward search continues till all the functionalities are satisfied.

4 An Example Case

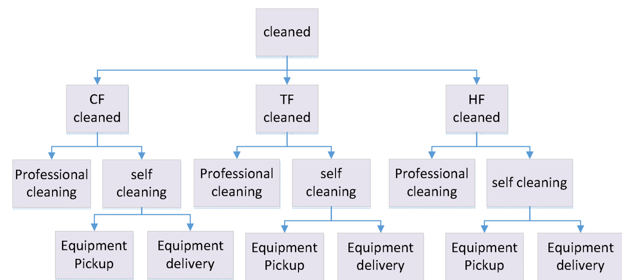

Figure 4. Goal hierarchy for the online cleaning service.

Consider the case about the online cleaning system. The system provides different functionalities, like “carpet floor cleaning”, “tile floor cleaning” and “hardwood floor cleaning”. For each cleaning function, there are two ways to achieve the goal, book the profession cleaning service or rent the equipment and do self-cleaning. These two ways should be

Authorized licensed use limited to: Southern University of Science and Technology. Downloaded on October 05,2021 at 11:44:21 UTC from IEEE Xplore. Restrictions apply.

presented as user choices. Thus, as discussed in Step 1 of Section 3.1, we add two additional predicates, “self cleaning” and “professional cleaning” to the goal. By doing so, our approach can treat them as multiple functionalities and plan paths for them. If the user chooses to rent the equipment, she may pick up the equipment by herself or wait for the equipment to be delivered.

The overall goal set RG for the case study is shown in Figure 4. Some sample goals in RG are given as follows.

G_1 =“self cleaning” ⋀ “equipment delivery” ⋀ “deposit refund” ⋀ “carpet cleaned”

G_2=“professional cleaning” ⋀ “carpet cleaned”

G_3 =“self cleaning” ⋀ “equipment return” ⋀ “deposit refund”⋀ “tile cleaned”

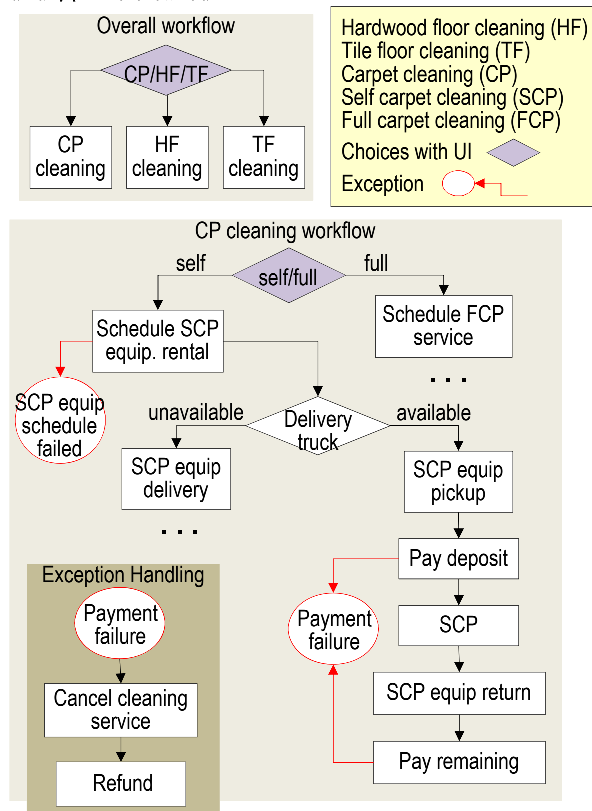

Figure 5. Automatically generated workflow for the online cleaning services case study.

We use MFP to derive the multi-functionality workflow as shown in Figure 5 (some sub-workflows are omitted). Note that the automated workflow generation based on the algorithm given in Section 3.1 will not include the choices (purple diamonds) which are added after further processing steps. Also, exception handling processes are added to the workflow after applying the algorithm discussed in Section 3.2.

5 Conclusion

We have presented a model for formally specifying systems requiring holistic service composition procedures and have developed an efficient reasoning algorithm for obtaining a holistic workflow for such applications. Our model extends existing models to enable more comprehensive composition requirement specifications, considering systems with multiple functionalities, offering users more choices, and handling exceptions. The reasoning procedure augments existing planning methods to address the need for holistic composition. A case study is presented to show the resulting workflows generated from our reasoning procedures.

Future research directions include: (1) Incorporate methods to enable the system to continuously evolve by autonomously identifying new functionality requirements and compose services rapidly for them. (2) Support real time composition of services to handle newly emerging unexpected operational conditions. (3) Investigate planning algorithms to further enhance the composition procedure to meet optimality goals of the multi-functionality systems.

6 References

[1] D. S. Nau, T.-C. Au, O. Ilghami, U. Kuter, J. W. Murdock, D. Wu and F. Yaman, "SHOP2: An HTN planning system," Journal Of Artificial Intelligence, pp. 379-404, 2003.

[2] M. Klusch, A. Gerber and M. Schmidt, "Semantic web service composition planning with owls-xplan," in AAAI 2005.

[3] L. A. da Costa, P. F. Pires, M. Mattoso, "Automatic composition of web services with contingency plans," in ICWS, 2004.

[4] D. Berardi, D. Calvanese, G. D. Giacomo and M. Mecella, "Composition of services with nondeterministic observable behavior," in ICSOC 2005.

[5] A. L. Blum and M. L. Furst, "Fast planning through planning graph analysis," Artificial Intelligence, pp. 281-300, 1997.

[6] J. Hoffmann, "FF: The Fast-Forward Planning System," AI magezine, pp. 57, 2001.

[7] A. Gerevini and I. Serina, "LPG: A planner based on local search for planning graphs with action costs," in ICAPS 2002.

[8] J. Fu, V. Ng, F. B. Bastani and a. I.-L. Yen, "Simple and fast strong cyclic planning for fully-observable nondeterministic planning problems," in IJCAI 2011.

[9] A. Cimatti, M. Pistore, M. Roveri and P. Traverso, "Weak; strong; and strong cyclic planning via symbolic model checking," Artificial Intelligence, pp. 35-84, 2003.

[10] H. Yang, X. Zhao, C. Chao and Z. Qiu, "Exploring the connection of choreography and orchestration with exception handling and finalization/compensation," in FORTE 2007.

[11] K. Christos, V. Costas and G. Panayiotis, "Enhancing BPEL scenarios with dynamic relevance-based exception handling," in ICWS 2007.

[12] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith and S. Narayanan, "OWL-S: Semantic markup for web services," 2004. [Online].

[13] J. Domingue, D. Roman and M. Stollberg, "Web service modeling ontology (WSMO) - An ontology for semantic web services," W3C Workshop on Frameworks for Semantics in Web Services, 2005.

[14] N. Milanovic and M. Malek, "Search strategies for automatic web service composition," in IJWSR 2006.

[15] A. Mediratta and B. Srivastava, "Applying planning in composition of web services with a user-driven contingent planner," IBM Research, 2006.

Authorized licensed use limited to: Southern University of Science and Technology. Downloaded on October 05,2021 at 11:44:21 UTC from IEEE Xplore. Restrictions apply.