ENHANCING PLANNING BASED AUTOMATED SERVICE COMPOSITION MODELS AND TECHNIQUES

by

Wei Zhu

APPROVED BY SUPERVISORY COMMITTEE:

____

Dr. I-Ling Yen, Chair

____

Dr. Farokh B. Bastani

____

Dr. Lawrence Chung

____

Dr. Weili Wu

ENHANCING PLANNING BASED AUTOMATED SERVICE COMPOSITION MODELS AND TECHNIQUES

by

WEI ZHU, BS, MS

DISSERTATION

Presented to the Faculty of

The University of Texas at Dallas

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY IN

COMPUTER SCIENCE

THE UNIVERSITY OF TEXAS AT DALLAS

May 2017

ACKNOWLEDGMENTS

This dissertation would not have been possible without the support of many people. I wish to express my gratitude to my supervisor, Dr. I-Ling Yen, who was abundantly helpful and offered invaluable assistance support and guidance. Deepest gratitude is also due to the members of my supervisory committee, Dr. Bastani, Dr. Wu and Dr. Chung, without whose knowledge and assistance this study would not have been successful. I also express my love and gratitude to my beloved family and friends for their understanding and endless love through the duration of my studies.

December 2016

ENHANCING PLANNING BASED AUTOMATED SERVICE COMPOSITION MODELS AND TECHNIQUES

Wei Zhu, PhD
The University of Texas at Dallas, 2017

Supervising Professor: I-Ling Yen

Service-oriented architecture (SOA) has been widely adopted by government and industry to enable rapid systems development and deployment via composing existing services. To further reduce manual efforts in service composition, planning techniques are used to automate the service composition process. However, some gaps still exist in automated service composition research. First, real world systems are complex and need to consider multiple functionalities. Existing service composition models do not support the specification of multiple functionalities and existing planning techniques cannot be used directly to generate a composite service with multiple functionalities. Secondly, a lot of work exists for improving the performance of planners for automated service composition, but none of them consider the scalability problem due to the number of services. With the growing adoption of SOA and open source development, more and more concrete services are becoming available, which makes the scalability issue highly pressing. Thirdly, existing service models are based on software services, while physical services have quite different characteristics. Though some works consider modeling physical services, they are still confined to the same issues of the software services. When considering automated service composition, these models are insufficient.

In this dissertation, the three issues in automated service composition are thoroughly investigated and methods for coping with them are developed. For the first issue, we extend existing service models to support multi-functionality specification and develop planning techniques to facilitate service composition for multi-functionality systems. To cope with the second issue, we develop an approach that integrates service clustering and planning techniques to improve the performance of the automated service composition process and make it scalable with the number of services. We also develop a specification model for physical services and their compositions to ensure that automated service composition can be correctly applied to cyber physical systems and Internet of things applications. Our work significantly enhances the state-of-the-art technologies in automated service composition, making it more efficient and more applicable to a wider variety of application domains.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Service oriented architecture (SOA) has become a popular framework for software development in many application domains. A lot of companies and organizations provide their services as SaaSs (software as a service) and Apps over the internet. However, these services usually have their focused core functions and cannot alone satisfy many user requests. To fulfill a user request, it is generally necessary to compose multiple services into a single composite service to achieve more sophisticated functionality. Due to the growing number of services and the complexity of the systems to be composed, it can be very ineffective and time consuming to perform service composition manually and automated service composition is, thus, highly desirable. Techniques for automated service composition have been investigated; however, there are still challenges for applying the automated service composition techniques in different application domains. In the following three sections, we discuss the major gaps in existing automated service composition models and techniques, and the research efforts in this dissertation for bridging these gaps. In Section 1.4, we discuss the layout of this dissertation.

1.1 Efficiency Issues with Large Number of Services Composition

To reduce the time and effort required in manual based composition, various automated service composition techniques have been proposed in the literature [1] [2]. Early works on automated service composition mainly apply logic-based reasoning to derive the composition solutions [3] [4]. Generally, logic-based reasoning techniques cannot be scaled well to large-scale problems. SHOP2 [1] and OWLS-Xplan [2] are widely used systems for automated service composition. 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. There are also automated service composition methods using search-based planning techniques [5] [6], similar to GraphPlan [7], Fast Forward (FF) Planner [8], Local Search for Planning Graphs with Action Costs (LPG) [9], Fast Incremental planner (FIP) [10], etc.

Due to the wide adoption of SOA, growing attention in open source, and the rapid development of services, SaaSs, and App libraries, the number of available services is growing very rapidly. Though systems can be quickly developed and deployed by composing these available services, service discovery and composition face challenges. Some research works attempt to reduce the service discovery and composition efforts by clustering services with similar functionalities together [11] [12] [13] or by constructing service hierarchies [14]. Services can be selected from the clusters with the desired functionalities and used for composition.

However, existing service clustering techniques cannot help automated service composition because the derived service clusters do not provide information that can be used by the composition reasoners. With a large number of services, the state space of the corresponding planning problem grows rapidly, making the planners difficult to scale.

We use planning techniques as the basis for composition reasoning and make use of the service clusters for coping with the scalability problem in conventional planning techniques due to the increasing number of services. We develop a two-level planning approach based on service clustering and identify the requirements for service clusters to facilitate two-level planning. Services are clustered into a hierarchy, which includes multiple clusters of service with one leading abstract service for each cluster. The specification of the leading abstract service of a cluster is a summarization of the common properties of the services in the cluster. Composition planning is first performed on the set of leading abstract services. The initial plan is then expanded to a complete plan by considering an expanded service set, which is constructed by replacing the leading abstract services by all the services in their clusters. To enable this two-level planning, some requirements for the service hierarchy should be satisfied. We develop the theoretical foundation of service hierarchy requirements and the methods for achieving two-level planning.

1.2 Issues toward Composing Services into Holistic System

Besides the performance problem, existing automated service composition research have also omitted some important issues toward composing services into a holistic system. Standard service specification, orchestration, and choreography models, such as OWL-S [15] and WSMO [16] have been widely used. Various models for automated service composition have also been considered [5] [6] [17]. But these models have omitted some important issues toward composing services into a holistic system. First, all these techniques consider a single system goal, like a plan does. 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 perform service composition 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 the composition effort and generate a well integrated workflow.

The second issue in automated service composition is the consideration for services with multiple effects. 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 is no customer remaining in the store. Then, a store closing service closes all the 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 finding misplaced items, finding some products with low shelf stocks, or finding 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 to 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 for a system, 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 [18] [19], but some gaps still remain. One gap is that formal service definition models, such as OWL-S and WSMO, do not have a specific mechanism for specifying the multiple effects or exceptions of a service [15]. Without a proper specification model, the techniques for handling them becomes ad hoc. Another important gap in the literature 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 service composition model that supports the specifications of goals for the exceptions.

The third gap in automated service composition is the consideration of alternative paths toward a goal. Most existing planning techniques, especially those used for automated service composition, only derive one path from the initial condition to the goal state. But 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 dissertation, 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. To facilitate a formal treatment of the holistic composition problem, we extend the classical OWL-S service model [15] with multi-effect specifications 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. Also, 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.

Also, based on the multi-functionality model, we extend existing planning techniques and develop new planning algorithms to achieve efficient multi-functionality planning, including the Multi-Round Planning for Multi-Functionality (MRPMF) algorithm and the Single Graph Expansion for Multi-Functionality (SGEMF) algorithm. In both algorithms, all the goals specified for multiple functionalities are composed by the disjunctive relation and the integrated goals are called the Disjunctive Multi-Functionality (DMF) goals. MRPMF includes multiple planning rounds. In the first round, the classical planner is modified to plan for the DMF goals, but the termination condition is as long as the goals for one of the functionalities are reached.

Then, the corresponding “first plan” is retrieved. In the second round, the DMF is updated by removing the goals for the already-achieved functionality. The set of initial conditions is updated to include not only the original initial conditions, but also all the states in the “first plan”. Thus, the second round planning will obtain a “second plan” that starts from one of the states of the “first plan” and reaches the goals of another functionality. The planning rounds continue till all the functionalities are achieved. In SGEMF, plan graph expansion only takes place once, but breaks multiple times. It expands normally for the DMF goals, but takes a break when the goals for one of the functionalities are achieved. After the break, the backward search is performed from the goal state till it reaches any state in the existing plan or the initial state. Of course, when the first functionality is achieved, there has been no plan, so the backward search goes back to the initial state. After each break, the plan graph expansion continues till the next break condition is met. We compare MRPMF and SGEMF algorithms with the conventional approach, in which automated service composition method is applied for each functionality of the system to obtained multiple workflows for a single system. We also analyze the tradeoffs between MRPMF and SGEMF. Also, experimental studies are conducted to compare the performance of various automated service composition methods, including MRPMF, SGEMF, and the conventional approach. The results show that SGEMF has the best performance for multi-functionality planning and MRPMF can generate multi-functionality workflows with more overlapping services (i.e., the total number of services in the workflow is the least) because the branches are planned from the existing plan.

1.3 Service Computing for Physical Services

In recent years, internet-of-things (IoT) and cyber-physical system (CPS) have gained a lot of attention. Both IoT and CPS consider a vast number of static and/or mobile physical devices networked together into a system. Service computing technologies have been applied in CPS/IoT systems to achieve rapid discovery and composition of physical things for new or dynamically arising tasks [20] [21] [22] [23] [24]. However, existing service models are mainly designed for software services and are not suitable for physical services provided by CPS/IoT.

Some research works [25] [26] [27] use the event-based model from software systems to capture the event-driven characteristics of physical services. The event based model may be very useful in CPS/IoT application domains, but it does not offer additional features compared to the event model for software services. Also, because CPS/IoT entities may involve low-level control commands with different communication mechanisms and control sequences, [26] [28] and [29] extend the wrapping and encapsulation techniques in software world to unify the access interfaces for the physical services and hide the details for interacting with CPS/IoT devices. Though the encapsulation techniques are critical for CPS/IoT services, there is no differential treatment for CPS/IoT in applying these techniques relative to software services.

In fact, software services and physical services have some differences, but none of the existing works consider them. One major difference is the role that the physical thing (PT) plays in service provision. In software services, the PT is the computing and storage hardware. However, due to the sufficient uniformity in the computing facilities for software services and the high speed communication among them, though there are still issues like communication costs and workloads, the PTs for software services do not have a significant role. However, in physical services, the characteristics of a PT can impact the service it provides. For example, different types of vehicles can be used to transport people from a disaster site to a safe evacuation area. But each type of vehicle has its own characteristics, such as load capacities and number of seats. Also, even for the vehicles that are exactly the same, when grounding the service for transporting people, it is necessary to specifically determine the number of vehicles required and the number of trips each vehicle may have to make. Thus, the PT that provides a physical service should have a significant role in the specification of the physical service. Unfortunately, current service models, such as OWL-S and WSMO, do not support specifications of the physical things.

Another issue to be considered is the context of the PT. For example, in a rescue mission, some robots may be used for survivor detection. But the physical location of the robots must be at the rescue site. If not, additional services are required to bring them to the rescue site. Also, the side effect of a software service can be specified independently of other software services. This may not be true in physical things. For example, a car may transport a robot to a disaster site for a rescue search. In this case, the states of the service provider and the PTs the recipient of the service is may change together. Such impact need to be specified explicitly and existing software service models do not have such a feature.

Granularity is also an issue in CPS/IoT systems. (a) For example, a swarm of robots may provide some services as one unit. But each robot may also be used to provide different services. (b) Many PTs may have components. Generally, they provide services as single units, but require separate maintenance services or control software.

In this dissertation, we develop an ontology model for the specification of services in CPS/IoT systems. We define PTs and services separately and associate them in the upper ontology. We consider "context" for physical services and specify the "context precondition" and "context effect" for physical things. Same as precondition, the "context precondition" should be satisfied before activating a physical service. To specify the side effect of a service, we define recipient PTs (not services) and support the effect specifications not only for the service itself, but also for the recipient PTs. The ontology model for CPS/IOT can facilitate automated service composition and we illustrate this capability via a case study system.

1.4 Dissertation Organization

This dissertation is organized as follows. In Chapter 2, we survey service composition models and techniques and automated service composition algorithms. Chapter 3 discusses the multiple level planning algorithm for achieving more efficient automated service composition considering a large number of services. Chapter 4 discusses extended service model, exception model and system model for a holistic service composition problem. It also discusses the automated composition solutions for composing services into a holistic system. In Chapter 5, we propose a novel PT-SOA (PT stands for physical things) model, which extends OWL-S to model CPS/IoT services and systems. Also, a multi-stage automated service composition reasoning technique is developed to support efficient and automated composition of CPS/IoT services while considering the physical properties and constraints of the physical things that provide the CPS/IoT services.

CHAPTER 2

RELATED WORKS

SOA has become the major architecture model in modern software development process. Among various research issues, how to discover and compose services into an integrated system is one of the most important and challenging problem. The first step toward service composition is the specification of individual services. Without proper service specifications, designers will have no basis to know what each service can do, not to mention whether or how to use it in the target system. Thus, many research works have investigated the models and languages for specifying the services. In Section 2.1, some important service specification models are discussed. After services are selected and composed into a system, it is necessary to specify how the services are composed and how they interact. Some important service composition specification models and tools have been developed. In Section 2.2, we survey some important models for service composition.

With the advances in various hardware and networking technologies, computing devices becomes more and more pervasive in human society and in our daily lives. Thus, cyber physical systems (CPS) and Internet of things (IoT) have gained increasing importance. SOA technologies, especially the service discovery and composition techniques, have been used in these systems. Since CPS and IoT services have different properties and focuses compared to conventional software services, new service specification and composition paradigms for CPS and IoT systems have been proposed. In Section 2.3, we discuss some important issues and corresponding models for CPS and IoT services and compositions. We also outline the deficiencies in existing models and discuss the idea in a new model we propose.

Given the models for service and service composition specifications, the next issue is how to select services from the vast pool of available services provided by different venders and how to compose them such that the system requirement can be satisfied. Various service composition techniques have also been investigated to help users systematically develop the workflow for a desired system. These service composition techniques can be categorized into functional service composition, QoS-driven service composition, and context-aware service composition. Functional service composition can further be categorized into manual and automated composition methods. In Section 2.4, we discuss some service composition techniques that help users to manually but systematically compose services into a system. In Section 2.5, we first discuss the differences of automated service composition from the general reasoning process. Then for each issue, we survey existing planning techniques and corresponding usage of the planning techniques in existing automated service composition systems. We also address the gaps in existing automated service composition techniques and discuss our efforts in bridging the gaps. In Section 2.6, we briefly discuss existing works in QoS-driven and context-aware service composition research. Finally, in Section 2.7, we discuss the gaps in existing service composition research and how our research bridges these gaps.

1.5 Service Specification Models

The first specification language for Web services is WSDL (Web Service Definition Language) [30]. WSDL is an XML based definition language and it focuses on Web service functionality description. It provides an informal description of what the service functions are and formally specifies the IO ports of a service. For each port, the communication message structures, the actual communication protocol such as SOAP, HTTP, etc., can be specified.

WSDL also supports the definition of the operation modes of a port, including notifications, one-way, solicit-response, and request-response modes. The partner-link in WSDL can specify the connection of a port of one service to the port of another service, which is the only hint of composition in WSDL. Due to its well defined specification model for the service invocation process, many other semantic Web service models embed WSDL in their models.

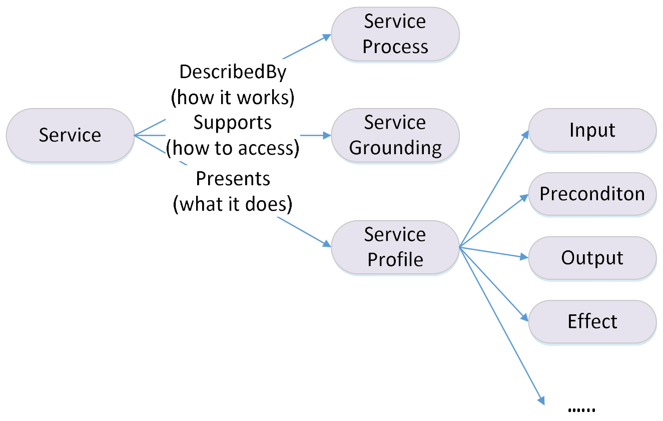

Figure 1. OWL-S Service Top Level Ontology

WSDL focuses on concrete service invocation specifications, but lacks semantic definitions of the functionality of the services. The Ontology Web Language for Services (OWL-S) is, hence, proposed to achieve semantic service specifications. OWL-S defines the service specification ontology, which includes three classes, profile, process, and grounding (as illustrated in Figure 1). The profile class support some informal descriptions of the properties of a service for human reading, including the service name, service category, description, publisher, limitations on applicability of the service, quality of service, etc. It also provides formal specifications of the inputs, the outputs, the preconditions and the effects (IOPE) of the service. IOPE specifications are essential for automated service composition. The process class is used to describe the composition logic for a composite service. The grounding class provides detailed invocation information, including the physical binding information and the link to the WSDL of the service.

Web Service Modeling Ontology (WSMO) is another semantic service specification model [31] [32]. In WSMO, a Web service s is specified by five classes, including interface, capability, non-functional properties, ontology, and mediators. The interface class includes two specifications, choreography and orchestration. Choreography specifies the communication patterns during the invocation and execution of s. It is a more comprehensive IO specifications for s, including transactional specifications. Orchestration specifies how other services are composed into s, which is actually internal to s, not exactly an interface specification. The capability class provides an axiom-based formal specification of the preconditions and effects of a service. WSMO separates the precondition and effect into two parts. The precondition and assumption specify the conditions to be met by the input space and by the world, respectively. The post-condition and effect specify the effect in the view of the invoker and the effect to the environment (the world), respectively. Non-functional properties class is self-explanatory. The ontology and mediator classes specify the ontologies imported and the mediators used, respectively, for the service.

To facilitate automated composition, we need a formal model for service specifications and we choose to use the OWL-S model as the basis.

1.6 Service Composition Specification Models

In the literature, there are two major paradigms for service composition, orchestration and choreography [33]. The major difference between orchestration and choreography is in service execution and control. In orchestration, service execution is managed by a central orchestrator (conductor). The conductor is responsible for the invocation of and interaction with each atomic service in a composite service following a predefined sequencing. These atomic services do not interact between themselves in the orchestration paradigm. In choreography, the services in a system are executed in a peer-to-peer manner. Each service operates according to a set of rules, which define how the service should be consumed and its interaction patterns with other services. Many earlier service composition models are based on the orchestration paradigm, including WS-BPEL, OWL-S, etc. The WSMO model combines both orchestration and choreography.

BPEL [34] is the most widely used workflow description model in industry. It is not specifically designed for SOA, but has been adapted to WS-BPEL for service composition specifications. For workflow execution, WS-BPEL links services in the workflow to their WSDL specifications which guide the invocations of individual services. It also supports common orchestration sequencing control constructs, such as sequence, if, while/for-each/repeat-unit loops, etc. Some additional constructs provided in WS-BPEL facilitate more convenient workflow specifications. The pick construct allows the specification of alternative branches in a workflow and the first ready branch is executed. The assign construct allows the state of service execution to be preserved in a variable and used in later stages of the workflow execution. In the orchestration paradigm, service composition can be concrete or at an abstract level. The concrete model provides the specification of a composition, and the abstract model enables the specification of a composition problem. WS-BPEL supports both concrete and abstract compositions.

In OWL-S [15], services and compositions (processes) are treated uniformly. The process class for a service s specifies how other services are composed together to fulfill the functionalities specified by the profile (such as IOPE) of s. Subsequently, s can be treated as a concrete service and used in the compositions for other services. OWL-S also offers common execution sequencing constructs, including the sequence, if-then-else, and iterate/repeat. OWL-S sequencing also focuses on execution ordering and offers constructs: split (parallel execution of all branches), split-join (all split branches should finish execution before join), any-order (execute branches sequentially in any order), and choice (execute one of any branch). Also, same as WS-BPEL, OWL-S supports both concrete and abstract composition models, allowing the specifications of a concrete composition as well as a composition problem.

WSMO [16] integrates choreography and orchestration (only for composition specification, not for centralized execution control) to support the detailed specification of the external interface and internal composition of a concrete Web service. Its upper ontology model includes the specifications of four classes: goals, mediators, ontologies and Web services. The Web services provide the set of available concrete services (both atomic and composite) that can be used for composition. Ontologies and mediators are generally imported from the predefined corresponding sets. The terminologies used in the specifications for the set of Web services and the goals are defined in the ontologies. Mediators are used to overcome interoperability problems between interacting Web services with mismatching IOs. Different from OWL-S and WS-BPEL, WSMO considers a composition problem (an abstract service to be composed) different from a concrete service (a Web service). The goals specify what are to be achieved for a service composition request.

To facilitate automated service composition specification, we need formal specifications of the composition problem as well as the concrete services. Also, we need a clear and simple model for composition and composition problem specification. Thus, we construct a model that is a combination of BPEL, OWL-S and WSMO. We leverage the formal specification model in OWL-S, the concept of separation of composition and concrete services in WSMO, and the separate composition specification (the workflow) in BPEL to form our automated service composition model.

1.7 Service Models for CPS and IoT Systems

Due to the advances of computer and networking technologies, many physical systems are computerized and networked. In recent years, cyber physical systems (CPS) and Internet of things (IoT) have become very popular. Some research works attempt to adapt the service computing models to CPS and IoT systems. But existing service models are mainly designed for software services and are not suitable for services provided by CPS/IoT. In the following subsections, we survey existing works related to the modeling of CPS/IoT services.

1.7.1 Event Driven CPS/IoT Service Models

Similar to some software services, physical services may be invoked by requests and/or by events. For example, air conditioner may be turned on (invoked) due to a high temperature reading from the temperature sensor or by the user. Also, a transport service provided by a car may be invoked by a user to transport the user from one location to another. If the gas tank of the car is low, the gas filling service should be invoked before providing the car service. Before the car can fill gas, it has to drive to the gas station, namely, invoking the move service of the car.

Some research works use the event-based model from software systems to capture these event driven characteristics in physical services.

In [25], an event-driven service oriented architecture (ED-SOA) for IoT systems has been proposed. In this model, events are treated at the same level as services. A service can subscribe to a set of events and it takes corresponding actions when some of these events are delivered to it. Also, events may be generated during service executions and they will be delivered to their subscribers. In DPWS by OASIS [27], a similar event-based model is proposed for IoT devices. DPWS also consider device access protocol specification and service specification and discovery for the IoT devices.

The SenaaS (sensor as a service) system [26] is a virtualized IoT framework realizing the event-driven SOA in the IoT domain. Its virtualization layer receives events and manages and sends them to subscribers to take appropriate actions.

SOCRADES [22] also uses the same event model for modeling production processes in manufacturing systems. In SOCRADES, the status information of the physical entities and sensors can be treated as events and can be subscribed by other services through the SOCRADES event system. Also this model uses a virtual composition language to specify the bindings of events, event handling services, and the corresponding physical entities together.

In [28], an event based model is also used as the underlying system model. But unlike SenaaS and SOCRADES in which the event handling logic is manually determined in advance, this paper emphasizes to dynamically compose services to handle events. After an event is raised, the control layer determines a control decision for the event. Then the services are composed together to realize the control decision. Also, since some physical devices are configurable, the services incorporate configurability to support flexible provisioning. Though the dynamic control decision making and on-the-fly service composition are important for unexpected situations, there are no effective methods in this framework to support such goals.

1.7.2 Service Middleware for Consistent CPS/IoT Service Invocation

Due to the complexity in the invocation of physical services, some CPS/IoT works develop middleware to encapsulate the devices and provide a uniform interface to access devices that offer similar functionalities. For software services, the differences in interfaces of similar software services are less significant. But CPS/IoT entities usually have very different interfaces because CPS/IoT entities may involve low-level control commands with different communication mechanisms and control sequences. Thus, it is very important to extend the wrapping and encapsulation techniques in software world to unify the access interfaces for the physical services.

The SenaaS [26] middleware consists of three layers: the service virtualization layer (discussed earlier), the semantic layer and the real-world access layer. The functionality of the virtualization layer has been discussed in Section 2.3.1. The real-world access layer provides unified interfaces for accessing similar services provided by functionally similar IoT devices with different communication mechanisms. The semantic layer provides the needed ontologies in the middleware to support the service specifications in different layers, including the sensor ontology, an event ontology, and the service access policies. These ontologies can facilitate cross layer mappings and enhance the effectiveness of device encapsulation.

In [28], the proposed system consists of 3-tiers. The environmental tier encapsulates the physical devices. The control tier consists of controllers. Each controller subscribes to specific monitored data gathered by sensors, analyzes the sensor data to make control decisions, and composes services to realize the control decision. The service tier analyzes common services needed by the control tier and composes the functionalities provided by the IoT devices to realize these identified common services. Thus, the specific accesses to the devices are not exposed to the control tier or the users to avoid complex access procedures. In [35], a similar 3-tier architecture is considered and the goal is also to encapsulate the interaction protocols with the devices and conduct service composition to react to situations.

In ScriptIoT [29], a common script interface is provided to access IoT sensors with different data formats and communication mechanisms and to activate different IoT devices. For example, a common "fetch(d)" command can be used to fetch data from a sensor d of any type and the underlying access protocol for d and conversions of d's data format are encapsulated. Such sensor data accesses can also be registered as an event and only when the data satisfies a certain condition will the data be delivered to the request issuer.

1.7.3 Modeling Physical Things

The physical things (PT) in CPS and IoT systems have a significant role in the composition reasoning of physical services, which is very different from composition reasoning for software services. In software services, the PT is the computing and storage hardware. However, due to the sufficient uniformity in the computing facilities for software services and the high speed communication among them, though there are still issues like communication costs and workloads, the PTs for software services do not have a significant role. In CPS/IoT, the physical thing that provides a service and its properties are very important. For example, different types of vehicles can be used to transport people from a disaster site to a safe evacuation area. But each type of vehicle has its own characteristics, such as load capacities and number of seats. Also, even for the vehicles that are exactly the same, when grounding the service for transporting people, it is necessary to specifically determine the number of vehicles required and the number of trips each vehicle may have to make. The second issue for physical service composition is that a PT may be able to provide several different types of services. However, it is frequently not possible for one PT to fulfill multiple services it provides at the same time. The schedule of individual PT will impact the service composition result. The PT context is also an issue when doing the physical services composition. The context is defined as the dynamic changing states of a PT. For example, in a rescue mission, some robots may be used for survivor detection. The physical location of the robots must be at the rescue site. If not, additional services are required to bring them to the rescue site. Last issue that needs to be considered is that the side effect of a software service generally can be specified independently of other software services. This may not be true in physical things. For example, a car may transport a robot to a disaster site for a rescue search. In this case, the states of the service provider and the PTs the recipient of the service is may change together. Such impact need to be specified explicitly and existing software service models do not have such a feature.

Existing service specification and composition models do not address the above issues. The event based model discussed in Section 2.3.1 may be useful in CPS/IoT application domains, but it does not offer additional features compared to the event model for software services. The frameworks discussed in Section 2.3.2 borrow the wrapping and encapsulation solutions in software systems and apply them to CPS/IoT systems to hide the details for interacting with CPS/IoT devices. None of these models can handle the issues discussed above.

The interaction protocols in UPnP (universal plug and play), the lower level specifications in DPWS (Devices Profile for Web Services), and the DDL (device description language) provide device specific specifications, but they focus on the interactions with the devices, not about the properties of the devices themselves. In [22], the availability of a device for service provisions is considered as an event and the broker will deliver this type of events to the subscribers. It offers some help with issue (1), but only on the availability of the devices, not the quantitative capacity of the devices. To facilitate CPS/IoT service modeling and composition, a new model is really essential.

1.8 Manual Based Service Composition Techniques

1.8.1 Basic Composition Tools

The basic tools to help with manual based service composition are the GUI provided by the composition specification models discussed in Section 2.2. These tools allow users to select individual services and the control flow constructs for composing a workflow (composite service). For example, the BPEL engine, Oracle BPEL Process Manager, provides the GUI and the translator for building and deploying workflows in BPEL language. The OWL-S editor [36] provides a GUI and the translator to help create OWL-S specifications of a concrete atomic/composite service. Web Service Modelling eXecution environment (WSMX) [37] is a reference implementation of WSMO. It is an execution environment for specifying the business process integration.

1.8.2 Pattern Based Service Composition

Pattern based service composition has been proposed in the literature to help achieve easier service composition. A service pattern reuses previous composition results or provides incomplete composition template to support various groundings. Workflow templates is a type of pattern based service composition, in which each template is a service pattern. Generally, workflow templates are in a form of abstract workflows, where services in the workflow can be concrete or abstract. An abstract service has well defined IOPE and is ready to be grounded to a concrete atomic service or a concrete composite service. Generally, workflow templates are manually constructed and it can help with the composition of many different concrete workflows by having different instantiations. However, workflow templates do not consider flexible IOPE specifications and configurable control flows. To make the service patterns more flexible and extensible, [38] proposes a new service pattern model which allows the specification of IOPEs of the service pattern and the services in the workflow to have variables. Thus, grounding the pattern not only requires grounding the abstract services in the pattern, but also instantiations of variables in those flexible IOPEs, making the functionality of the pattern changeable. The service pattern model in [38] also supports adaptive control flow structure, allowing a service and/or a sub-workflow to be skipped completely, offering additional dimension of flexibility in service patterns.

Some research works consider automated pattern discovery by mining composite services and extract useful, common service patterns [39] [40]. This approach can eliminate the manual efforts required for service pattern development. In [41], a service pattern model based on OWL-S is proposed. It develops AI planning-techniques to achieve automated pattern-based service composition. In addition, functional operators are provided to compose service patterns, including concatenate, splice (for splicing loops in two patterns to improve the performance), and invert (for ensuring the proper usage of service pairs such as encoding and decoding).

1.8.3 Model-driven Service Composition

Model-driven architecture (MDA) has been widely used in software engineering to facilitate systematic development of software systems and automated generation of program skeletons. It has also been applied to service composition in various degrees.

In [42], a model-driven service composition approach is proposed. It extends UML to model service composition and specify services. An abstract meta-model called Information Model (IM) is defined in UML to simulate current standard service models such as BPEL. IM defines the composition rules, including structure rules, data rules, behavioral rules, resource rules and exception rules. These rules are defined in the formal language OCL [43] to facilitate automated composition synthesis. When a user request arrives, the system performs composition in four phases to satisfy the request. In the Definition phase, it selects a set of activities that can be used to satisfy the request. For each activity, its input/output messages and their behaviors as well as its exceptions are specified in the behavior and exception rules. For each exception, additional activities are identified for handling it. The composition constraints for each activity, similar to the pre/post conditions, are also specified. In the Schedule phase, the order for executing the activities are determined and specified in the structure rules. Also, based on the execution structure, input/output messages are associated and constraints are added. For example, in a travel plan, a hotel reservation should be done after the flight reservation and the input message for the hotel reservation activity, namely, the check-in date, should be confined by the output message of the flight reservation activity, namely, the arrival date. In the Construction phase, a concrete workflow is built by selecting the concrete service and the execution role for each abstract activity. The final Execution phase is the only phase involving automation. It translates all the composition rules defined in OCL into an executable composition language such as BPEL, making the workflow ready for execution. Though the paper proposes a systematic service composition procedure, it is mostly manual and does not take much advantage of MDA.

In [44], a similar model-driven approach is presented. It uses the native UML language for composition specifications. The UML activity models are used to capture the control and data flow of the composition at an abstract level. The UML class models are used to model the concepts defined in semantic and QoS ontologies. Also, it is believed that the semantic descriptions in OWL-S and WSML are at too low a level and are thus being reverse engineered into UML models. Based on the abstract model, concrete services are selected to fulfill the activities and satisfy various constraints. Finally, the concretized model is ready to be translated into any executable language, such as BPEL.

Another similar work is proposed in [45], which also uses native UML to specify the compositions. Transformation rules are specified in ATL and used to transform UML specifications into BPEL workflows.

All the works based on MDA seem to be "yet another model" for service composition. Besides potentially being at a higher level, it does not seem to have significant benefit in easing the composition tasks.

1.8.4 Service Clustering

Due to the wide adoption of SOA, growing attention in open source, and the rapid development of SaaS, the number of available services is growing very rapidly. Systems can be quickly developed and deployed by composing these available services. But on the other hand, with the very large number of services, it is very difficult to effectively discover them and select them for composition. Some research works attempt to reduce the service discovery efforts by clustering similar services together [11] [12] [13]. or by constructing service hierarchies [14].

One fundamental issue in service clustering is to measure the similarities between services, i.e., how to construct the feature vector for each service such that similarity between two services can be computed from their feature vectors. Generally, similarity considerations include the similarities between the input/output parameters and the similarities in service descriptions, such as their functionality descriptions and the IOPE descriptions. There is a large body of research in word-based semantic analysis that can be used for measuring the similarities between services based on their word-based specifications. Thus, most of the service clustering research focuses on (1) From where the information about the services should be obtained. (2) Since the service related descriptions are short texts, there are insufficient words to be analyzed. How to extract more keywords out of a service, such as from its IO parameter names that potentially consist of compound words, etc. (3) Service specifications have specific structure. For example, if an input parameter of service $s1$ and an output parameter of $s2$ are similar, or if a keyword from $s1$'s functionality description and a keyword from $s2$'s input data type specification are similar, should these contribute to the similarity of $s1$ and $s2$?

In [46], the WSDL definitions and service descriptions of each service are converted into a feature vector, including the keywords with their corresponding counts. The source for service description is from SALCentral.org. The port types, operations and messages defined in WSDL, including the names as well as comments, are also processed for keywords extraction. The keywords from different sources are placed in different bags, named as A, B, C and D. Bag A includes keywords from the general functionality description. Bag B includes keywords from the port types and operations defined in WSDL. Bags C and D include keywords from the input and output related contents, respectively. The service clustering technique is an ensemble of several classifiers, one for each feature bag or each feature bag group. Multiple outcomes are voted to determine the final clustering of the services. A problem with this work is the pure keyword based approach, which has the limitation of not recognizing the similarities between the keywords themselves. Later works use the help of WordNet or normalized Google distance (NGD) to enhanced similarity computation.

In [47], an improved similarity metric is used. After extracting keywords from various sources, keyword expansion is applied using the WordNet ontology. The similarity between two services is computed by the ratio of common terms between the two services. It uses the hierarchical agglomerative clustering method which merges the clusters together from bottom up until the stopping criterion is satisfied. In [12], the keywords from the descriptions and IOPE definitions of each service are extracted to form a service description vector (SDV). Then the set of keywords in SDV are expanded by their anonyms defined in a concept ontology such as WordNet [11]. Services are then clustered based on the expanded keyword set and each cluster is labeled by the common concepts in the cluster. [48] further improves the similarity computation by using Natural Language Processing (NLP). After keyword extraction and expansion, NLP is used to help with disambiguation of word senses based on the context of the word. Instead of calculating the term similarity, sense similarity is computed and it is the distance of two senses via their Least Common Subsumer (LCS). The hierarchical agglomerative clustering method is then used to cluster the services. Though better similarity metrics are used in these works, a major problem with them is that the keywords extracted from different sources are treated uniformly, i.e., the service specification structure is ignored and some similarity contributions may be counted in incorrectly (Point 3).

Besides [46], several other works also divide keywords into different sets based on the sources (input, output, etc.) and compute similarities for each set and then integrate them together. In [49], similarity of input, output, and service functionality are computed separately based on keywords extracted from WSDL. Similarity between each pair of keywords are computed based on their distance in WordNet ontology. These pairwise keyword similarities are averaged to obtain the similarity for each keyword set. The overall similarity between two services is the sum of the similarities of all the sets. Then, the K-medoids algorithm is used to cluster the services into different clusters. [14] uses a similar method as in [49], but the overall similarity between two services is the weighted sum of the similarities of individual sets. Agglomerative hierarchy clustering is used to build the service hierarchy from bottom up.

In [50] keywords are extracted from the WSDL specifications, including the WSDL content, WSDL types, WSDL messages, WSDL ports and the Web service name. The high frequency keywords which do not contribute much to the clustering are eliminated. Then, normalized google distance (NGD) is used to quantify the similarity between two terms. Overall similarity considers the count of matching terms in WSDL type, WSDL port and WSDL message, average NGD between service names, etc. K-means algorithm is then used for clustering services.

In [51], keywords are extracted from the service name, operation name, input and output parameters. For each set of keywords, most similar keywords from two services are paired together. Similarity within a set is measured by the ratio of the number of paired parameters to the number of all parameters. Similarities of the input and output are combined by a weighted sum and similarities of multiple operations are combined by simple average. Then the clustering is designed based on a bottom-up method.

1.9 Automatic Service Composition Methods

To reduce the time and effort required in manual based composition, various automated service composition techniques have been proposed in the literature. These works take the specifications of a pool of services into the reasoning process to determine the sequence of services that can be composed together to achieve the overall system goal. In most of these works, composition reasoning is achieved by a planner. Originally, planners are supposed to only generate a sequential plan for a given problem, including a sequence of actions to be taken to achieve the goal. Subsequently, planning research considers actions that may have nondeterministic effects, and a planner should take care of every effect to assure that the goal will be reached no matter which effect takes place. Such consideration is generalized to consider uncertain initial conditions. These considerations are essential in automated service composition. Here, we survey the important literature in planning techniques and in automated service composition. In Section 2.5.1, we discuss classical planning techniques. The planning techniques for handling conditions, loops are discussed in Section 2.5.2. Section 2.5.3 discusses automated service composition techniques that make use of planning and other reasoning techniques.

1.9.1 Basic Planning Techniques

GraphPlan [7] is a classical planner. It includes two phases, the graph expansion (forward) and the plan extraction (backward) phases. In the expansion phase, it takes the current state s_i (starting from the initial state s_0), applies the set of all fireable actions as_i (the preconditions of an action in as_i are satisfied by s_i and its effects do not conflict with other actions in as_i) and derives the set of new state s_{i+1} from the effects of actions in as_i. This is performed stage by stage till the goal state is satisfied. After graph expansion, a plan with the shortest path (or lowest cost) is extracted by backward search through the graph.

Instead of a full expansion for all fireable actions at each stage, a planning problem can also be treated as a search problem and only the best action is chosen to be fired next. Various planning algorithms based on different search schemes have been developed. HSP [52] and FF [8] are forward chaining based planners. HSP uses the best first search and considers restart after reaching a dead-end to avoid local optimal. FF uses a combination of breadth first and best first search. In both algorithms, the best action is the one with the shortest estimated distance from the resulting states of the action to the goal states. The LPG (Local Search for Planning Graphs with Action Costs) planner [9] is also based on local search, but it is a partial-order planner. A total-order planner fires an action only when its preconditions are fully satisfied. A partial-order planner selects an action for expansion as long as one precondition literal of the action can be satisfied. The best action is evaluated based on the distance of its resulting state to the goal and the number of preconditions of the action that have not yet been satisfied.

In all search based algorithms, it is necessary to estimate the distance from a certain state to the goal state. To enable efficient and relatively accurate evaluation, the relaxed plan graph is used. In the planning domain, a state is represented by conjunctive literals. The effect of an action could cause the addition of new literals or deletion of existing literals. A relaxed plan graph is derived by a graph planner, but ignoring the literal deletions of each action. Relaxed plan graph can be built efficiently because each relevant action only needs to be fired once. Also, it gives the minimal distance from a state to the goal and, hence, provides a good heuristic estimation. It is also used to quickly determine whether there exists a solution for the planning problem.

Hierarchical task network (HTN) planning is quite different from the planning algorithms discussed above. Similar to other planning algorithms, HTN supports the definitions of operators (also called primitive tasks, similar to actions in other planners), which have preconditions and effects. In addition to operators, HTN also supports the definition of methods, which are rules specifying how to decompose a task into a set of subtasks. For a given goal task, a primitive HTN finds the proper methods to recursively decompose it into the composition of a set of primitive tasks. Some HTN planners not only perform decompositions by method matching, but also use other planning techniques to compose operators to satisfy the goal of a task or subtask. Due to the use of decomposition methods, HTN can handle large-scale planning problems. Also, since the definition of methods requires domain knowledge, HTN is considered as a domain-specific planner.

1.9.2 Planning Techniques for Handling Nondeterminism and Uncertainty

A complete workflow should consider conditional branches and loops and these conditions are due to uncertain initial states and nondeterministic services (services producing different effects). Planning research has developed a rich set of techniques for nondeterministic and uncertain planning problem domain. Here we introduce these planning techniques.

WarPlan-C [53] is the first planner that considers conditions. It uses Prolog to reason for the plan with conditional branches. In [54], a conditional non-linear planning is introduced. [55]

suggests that linear planning is more suitable for conditional planning. All these works consider actions with multiple effects and split each into multiple pseudo-actions, one for each effect of the original action. In [54], the effects are prioritized and the pseudo-action with the effect of the highest priority is used first in a nonlinear planner to derive a "weak" plan. The planner is reinvoked to plan for each pseudo-action according to its priority. In [55], effects are not prioritized and all pseudo-actions are considered in the action set to construct a "weak" plan using a GraphPlan like planning algorithm. If a pseudo -action of a nondeterministic action is in the weak plan, then the planner is reinvoked to derive a subplan branch for each pseudo-action. The difference between linear and nonlinear plans is that a linear plan consists of actions that are executed sequentially, while in a nonlinear plan, actions with no causal relations can be executed in parallel. Generally, the partial-order planning techniques such as LGP are used for plan generation, which is the case in [54].

Similar to conditional planning, conformant planning focuses on uncertain initial states, i.e., the initial state consists of disjunctive clauses or may even be unknown. For example, on a cold day, some mountain roads may have heavy snow and require to drive with snow chains. Since the snow condition is uncertain, planning needs to consider both "snow" and "no snow" conditions. In [56], Graphplan is extended to be conformant. The disjunctive initial conditions are handled in a similar way as nondeterministic actions, where each condition is planned as a branch. The paper also discusses adding a sensory node into the plan. The sensor can be treated as a multi-effect action, where the effects are the initial conditions.

Contingent planning is also very similar to conditional planning, but it generally considers partially observable effects or effects that cannot be fully enumerated. Conditional planning discussed in [54] also considers partial observability and is also considered as contingent planning. Cassandra [57] considers contingent planning, but it mostly deals with observable nondeterministic actions, except that it considers interference between nondeterministic effects and attempts to prune the interfering and less important effects. In [58], all nondeterministic actions are translated to observable effects, so contingent planning is solved in the same way as conditional planning. [59] considers partial observability and uses a propositional formula to represent the belief states derived from a sequence of actions. Though every belief state (predicted nondeterministic effect) can have a plan, it is infeasible to do so due to the large number of belief states. This work chooses a lazy approach for belief state planning. It adds a heuristic to predict when a belief state will be reached, and derive the plan for the belief state when it becomes known that it is going to be reached or having a high probability of being reached. [59] builds contingent planning based on Conformant-FF [60].

In [61], the planning solutions are categorized into weak, strong, and strong cyclic plans. A weak plan is a chain of actions without branches. It does not take care of the nondeterminism of multi-effect actions and, hence, may not reach the goal if an unplanned effect has taken place. A strong plan takes care of all possible effects of all nondeterministic actions used in the plan and, hence, can guarantee that the goal can always be achieved no matter which effect of a nondeterministic action has taken place. Conditional planners tackle the strong planning problems and produce conditional, non-iterative plans. However, in some cases, while planning for a dangling effect (the effect of an action in the plan, but has not yet been taken care of), nondeterministic actions may be used and more dangling effects are created. If it happens indefinitely, then no strong plan can be derived. In case during planning for a dangling effect e'

of an action a (and a already has a "success" effect e), the planner may use a again to reach the goal state via e, which means e' becomes a dangling effect again and this may happen indefinitely. But it also means that the execution of the plan will eventually terminate when the effect e of action a eventually takes place. Due to the existence of the loop, such a plan only guarantees eventual success, and it is called the strong cyclic plan. Though a strong cyclic plan is a weaker plan than the strong plan, strong cyclic planning offers a solution when the strong planning cannot. Also, it is more complex to derive a strong cyclic plan because it is necessary to compare the resulting state of each action with the existing states in the plan when planning for dangling effects. Several strong cyclic planning algorithms have been proposed, including the Model Based Planner (MBP) [61], NDP (Nondeterministic Planning) [62] and Fast Incremental planner [10].

MBP [61] uses NUSMV (a new symbolic model verifier), to explore the states of a system and, hence, can provide reachability checking. NUSMV performs state expansion based on the Ordered Binary Decision Diagram (OBDD). To provide the strong cyclic planning, MBP removes the actions which do not have any progress toward the goal according to the BDD. And in its strong cyclic planning algorithm, MBP will iteratively prune the unconnected states to goal and remove all the outgoing states to these unconnected states until there are no changes.

NDP [62] is a strong cyclic planning algorithm that makes use of classic planning techniques, and it can be used with any existing classical planners. It converts a multi-effect action to multiple virtual actions and uses nondeterministic planning algorithms to build a strong plan. To build the strong cyclic plan, NDP iteratively picks one state (effect of state-action pair) from the solution set and performs planning for it until there is no state to pick. The states that cannot achieve the goal are pruned.

FIP [10] uses the same idea in NDP. After obtaining a weak plan, all the effects of the multi-effect actions in the weak plan are considered as intended effects. When planning for a failure effect, a simple heuristic of reaching its corresponding intended effect is considered to achieve the cyclic plan efficiently.

1.9.3 Automated Service Composition Reasoning

Early works on automated service composition mainly apply logic-based reasoning to derive the composition solutions. Sword [3] focuses on adding semantics into WSDL and uses RETE, a many-pattern-many-object-matching logic reasoner, to derive the composition. [4] adapts Golog into a sophisticated service composition model and uses Golog reasoner to perform composition reasoning. Some modeling concepts in this work inspire later models for composition reasoning. First, it incorporates the concept of service patterns and constructs composition constraints based on the service patterns to guide the composition reasoning, which can help reduce the complexity of the reasoning process. Also, it adopts the partial observability concept and introduces a sensing action to produce probabilistic effects based on beliefs for partially observable actions.

Several widely used automated service composition tools, such as SHOP2 [1] and OWLS-Xplan [2], are based on HTN. SHOP [63], an earlier version of SHOP2, uses a simple hierarchical ordered planner, which is a primitive version of HTN. It only considers method definitions and the planner recursively applies the matching methods to decompose the goal task till a plan with only concrete services is obtained. SHOP2 extends SHOP to handle nondeterministic services (creating conditional branches) and nonfunctional properties. OWLS-XPlan uses the planner XPlan, which integrates FF planning and HTN techniques. Whenever possible, XPlan decomposes a goal task using method definitions into subtasks. When no decomposition methods are found for a task or subtask, FF planning is used to compose services into the task/subtask. XPlan is more powerful than SHOP due to the addition of FF planning. But unlike SHOP2, XPlan does not handle nondeterminism. Another contribution of SHOP and OWLS-XPlan is the techniques for systematically translating the formal service specifications into the planning domain. In [64], SHOP2 converts DAML-S and OWL-S service specifications to PDDL. OWLS-Xplan integrates OWL2PDDL converter to translate OWL-S to PDDLXML (PDDL in XML format).

There are also service composition methods using search based planning, similar to the techniques in GraphPlan, FF, LPG, etc. Almost all of them also consider nondeterminism and/or uncertainty. In [5], DAML-S based semantic specification is used to specify services and composition goals. A search algorithm that is similar to LPG is discussed to automatically generate the desired composition. [6] simply considers services with nondeterministic effects and describes how to handle them, which is the same way as the conditional planners discussed in Section 2.5.2. In [17], a semantic markup language is defined based on PDDL. Thus, any PDDL planner can be used for composition reasoning. For nondeterministic actions, it first performs planning with "pseudo-actions" (as defined in Section 2.5.2) to construct the basic composition and identifies the success effect for each nondeterministic action. The additional effects other than the success effect for each nondeterministic action become exceptions and are handled accordingly. In [65], an AND-OR tree structure is used for the composition search process. A node (state) in the search tree is expanded with AND edges after applying a multi-effect service and with OR edges if multiple actions can be applied from the state, indicating potentially multiple paths to achieve the goal. Its search algorithm is similar to FF, but a different cost function is considered. It assigns a cost for each literal in a state expression, which is the distance from the state literal to the goal. The total cost of a state is the sum of the costs of all the literals.

Some automated service composition works consider non-sequential executions. In [66], simple multi-effect services are considered and handled the same way as conditional planning. It also considers asynchronous service execution and introduces new constructs to support it. But it did not consider how to automatically generate a composite service which uses these asynchronous constructs. In [67], sequential and parallel service execution constructs are considered in automated service composition. It uses an AND-OR tree structure similar to that in [65] for composition reasoning, except that each node in the tree represents a composition composed using sequential and parallel constructs. Expansion of the search tree is similar to [65], but results in an expanded composition. The commutative, associative, and distributive rules are defined to eliminate equivalent compositions (nodes) in the tree. Its composition search is based on the A* algorithm and uses distance to goal as the heuristic. In fact, the paper considers a hybrid bidirectional search, including the forward search discussed above together with a decomposition based backward search (similar to the concept in HTN) to hopefully reduce the search and improve search efficiency.

Several automated service composition works discussed earlier also consider using contingency planning techniques to handle unexpected situations [17] [5] [65]. In [17], besides handling nondeterminism as discussed earlier, it further considers services with effects that may not be known a priori (the paper calls this partial observability, which is not correct). The paper suggests to perform dynamic planning, in which planning and execution are interleaved, to resolve the problem. An incomplete composition may be derived and executed and new planning starts after execution since additional information about the effects of the services is known at this point. Similar to [17], [65] also considers that the effects of some services may not have been defined properly, either due to the user omission or due to the involvement of an uncertain domain. The paper suggests to specify the initial and goal states as belief states (probabilistic logic expressions) and use the AND-OR search algorithm introduced earlier to search for a valid composition. The resulting composition may contain states that cannot reach the goals. If such states are reached during execution, the user may provide modified specifications and contingency planning is activated to derive the new plan to resolve the problem. In [5], a new construct, try/catch, is introduced. During planning, multiple paths to achieve the goals are captured, and one of them is the primary composition and the rest are considered as contingency plans.

1.9.4 Remaining Issues in Automated Service Composition

Modern systems are complex and the system may have to be specified by different goals. Current planning techniques do not consider multi-goal planning. Thus, it is necessary to consider the goals separately and apply planning techniques to generate one workflow for each goal. These separate workflows can then be merged together into one integrated workflow to realize the overall system. This one-goal-at-a-time approach may be inefficient for service composition of complex systems. To reduce the composition effort and generate a well-integrated workflow, it is desirable to specify multiple goals for the overall system and compose one workflow with branches for satisfying multiple goals. In this dissertation, we consider the problem of holistic service composition by considering multiple goals at the same time. We also develop multi-goal planning techniques to achieve efficient holistic service compositions.

As discussed in Section 2.4.3, service clustering techniques have been developed to reduce the complexity of composition reasoning caused by the large number of services. It would be of great help if automated service composition can make use of these service clustering results. However, all the existing service clustering algorithms focus on service discovery. They cannot be used for reducing the planning complexity in automated service composition. In this dissertation, we explore the fundamental requirements for service clustering in order to integrate the clustering with planning techniques.

1.10 QoS-driven and Context-aware Service Composition

Sections 2.5 mainly focus on service composition based on the functional aspects. The QoS-driven service composition research considers composing services to satisfy QoS requirements. Generally, the composition problem assumes that the functional composition is already done at the abstract level (abstract services are composed together) and each abstract service can be grounded into a set of concrete services. The goal of the research is to efficiently find the best selection of services such that their composition satisfies the end-to-end system QoS requirements. Various search algorithms, such as linear programming, genetic algorithms, particle swarm optimization (PSO) algorithms, etc., have been used to obtain composition solutions. For example, [68] [69] [70] use linear programming techniques to search for the optimal service composition. [71], [72] and [73] use genetic algorithms to efficiently make the optimal QoS-based service composition decisions. The particle swarm optimization algorithm has also been used for QoS-based service composition, like the works in [74] and [75].

Similar to QoS-driven service composition, context-aware service composition selects the optimal composition solution based on best context matches [76] [77] [78] [79], while QoS-driven service composition selects the optimal composition solution based on QoS tradeoffs. Various context attributes for guiding the composition process include available resources, time constraints, user location, user profile, etc. Many context-aware service composition systems select services based on their locations. For example, when a user wants to have lunch, the composition process will select a restaurant and order a taxi service to bring the user to the selected restaurant. Both services should be selected based on user location and may also be based on user preferences in the user profile. In terms of how these applications are developed, they mostly employ a traditional, monolithic application model. It embeds contextual dependencies as if-then rules, which describe how context aware systems should react to context changes. These rules are encoded by the software engineer. Using this approach whenever the new context types and values are introduced in the system, new rules describing context behavior need to be created by the software engineer. This makes the applications static and inflexible. Furthermore, this may often limit applications to run on a specific device, while offering only predetermined functions to the user. As a result, this model is not suited to accommodate pervasive computing environments, which are characterized by richness of context, by the mobility of users and devices, and by the appearance and disappearance of resources over time.

Also, similar to QoS-driven service composition, the complexity of context-aware service composition comes from the large search space due to the large number of context attributes and their values. Since context is mostly dynamic, the context-aware service composition task needs to be done on the fly and needs to be adaptive to context changes. Most of the context-aware composition works extend existing composition techniques by considering rule-based context constraints. In [77], a six-phase context aware composition process has been discussed, which considers current context data collection, context change detection, select services with context constraint, then perform the context-aware composition task. In [78], context matching is considered under the situation when context information is incomplete and a graph based similarity measurement mechanism is proposed to effectively match context data. In [79], context constraints are specified as rules and services are ranked based on how well they satisfy the constraints. The composition is done dynamically through reasoning. In this dissertation, we only consider functional composition, without considering QoS metrics and contexts. But same as other functional composition based approaches, our solution can be integrated with QoS-driven and context-aware service composition techniques to make the best composition decisions in all aspects.

1.11 Major Gaps in Automated Service Composition

In this dissertation, we consider the major gaps in automated service composition research, including the modeling issues and composition reasoning techniques.

First, in service composition, the derived workflow is to be activated by many users and, hence, should provide multiple functionalities and multiple options to its users. In automated service composition, each functionality or each option corresponds to a set of goal states, which has not been modeled in the literature. Current planning techniques do not consider such multiple options either. Based on current automated service composition techniques, it is necessary to consider each functionality separately and apply planning techniques to generate one workflow for each functionality. These separate workflows can then be merged together into one integrated workflow to realize all the functionalities of the overall system. This one-functionality-at-a-time approach may be inefficient for service composition of complex systems. To reduce the composition effort and generate a well-integrated workflow, it is desirable to specify multiple functionalities for the overall system and compose one workflow with the necessary branches for satisfying these multiple functionalities. Since there is no existing work that covers the multi-functionality aspect in service composition, we consider the problem of holistic service composition in this dissertation. We also develop the corresponding multi-functionality planning techniques to achieve efficient holistic automated service compositions.

Second, the performance of composition reasoning may be an issue if the composition problem size is large. Research in planning algorithms intensively investigates how to improve planning performance. However, in service composition, the number of available services is very large and is still growing, which may have a significant impact on the performance of composition reasoning. Thus, it is desirable to reduce the reasoning complexity due to the number of services. However, no existing research addresses this issue. As discussed in Section 2.4.4, service clustering techniques have been developed to reduce the complexity of composition reasoning caused by the large number of services. It would be of great help if automated service composition can make use of these service clustering results. However, all the existing service clustering algorithms focus on service discovery. They cannot be used for reducing the complexity of the reasoning process. In this dissertation, we explore the fundamental requirements for service clustering in order to integrate clustering results with planning techniques.

With the growing pervasiveness of computing and networking technologies, cyber physical systems and Internet of things have growing importance. Service computing techniques have been used in CPS and IoT system. However, there are some missing links in the service models that are mainly designed for software systems. As discussed in Section 2.3, research in service modeling and composition techniques for CPS and IoT systems still follows the same thoughts for software systems. Thus, in this dissertation, we investigate the required information in automated service composition for CPS and IoT systems and enhance the current service models to incorporate this information.

CHAPTER 3

TWO-LEVEL PLANNING FOR SERVICE COMPOSITION REASONING

Authors – Wei Zhu, I-Ling Yen, Farokh B. Bastani, Jicheng Fu

The Department of Computer Science, EC31

The University of Texas at Dallas

800 West Campbell Road

Richardson, Texas 75080-3021

1.12 Introduction

Due to the capability of dynamic composition and easy reuse, service oriented architecture (SOA) has become a popular framework for rapid software development and deployment in many application domains. However, composing services into an application system still requires significant time and efforts. To cope with this problem, various automated service composition (ASC) techniques have been proposed in the literature.

Early works for ASC tend to use logic-based reasoning, such as Sword [3] and Golog [4], to derive the composition solution. Generally logic-based reasoning techniques cannot scale well to large-scale problems. SHOP2 [1] and OWLS-Xplan [2] are widely used ASC tools. They are based on Hierarchical task network (HTN), which are considered as domain dependent solutions due to the need for knowledge about how to decompose tasks in the specific application domains. There are also ASC methods using search-based planning techniques, similar to those used in GraphPlan [7], Fast Forward (FF) Planner [8], Local Search for Planning Graphs (LPG) [9], etc. All of these works consider nondeterminism and/or uncertainty [5] [6] [17] [65] [66] [67]. [17] and [65] further consider that the effects of some services may not have been well defined, either due to user omission or the involvement of an uncertain domain. [65] uses probabilistic propositional logic to specify the belief states and uses a heuristic search algorithm to search for a valid composition. When state information become clearer during execution or when the user provides modified specifications, a contingency composition can be derived. [5] considers ASC techniques for handling exceptions and introduces a try/catch construct to incorporate exception handling in composition languages. Over the past decades, ASC techniques have achieved significant success.

Recently, with the wide adoption of SOA and SaaS and growing attention in open source, the number of available services is growing very rapidly. With the large number of services, service discovery and composition face challenges. Some research works attempt to reduce the service discovery and composition efforts by clustering services with similar functionalities together [14] [49] [50] [51]. These works extract keywords from WSDL specifications about input, output, and service descriptions and cluster services based on similarities computed from these keyword sets. Common concepts of each cluster are obtained to label each cluster. During service discovery, the search will be selected from the cluster that best matches the discovery query.

The large number of services can significantly impact the feasibility of ASC also, because it results in explosion in the state space of the corresponding planning domain. It would be ideal if service clustering can be used in automated service composition reasoning. Consider a cluster-based two-level planning approach. A virtual parent service can be constructed for each cluster to represent all the services in the cluster. Planning can be performed on the parent services to obtain the Level-1 plan. Since the number of service clusters is relatively small, Level-1 planning can be done efficiently. The Level-1 plan can then be used to eliminate a large number of services. Only the services in the clusters whose parent services are in the Level-1 plan will be considered in Level-2 planning. The plan obtained from Level-2 planning will be the solution for service composition.

Unfortunately, existing service clustering techniques cannot help ASC because the derived service clusters do not provide information that can be used by the composition reasoners. In this paper, we investigate how to make the two-level planning (TLP) approach work and formally derive the TLP solution. As discussed earlier, TLP is realized on a service hierarchy. We first use a semi-supervised co-training algorithm to cluster the services in a similar way as conventional clustering algorithms do. After identifying service clusters, we build the service hierarchy, which includes a virtual root linking to a set of virtual parent services, each links to all the services in its corresponding cluster. We also define the precondition and effect of each virtual parent service according to the preconditions and effects of the services in its cluster.

We explore the fundamental requirements for the service hierarchy in order to facilitate the realization of TLP. We illustrate why an intuitive clustering and virtual service construction would not work. Then we define four criteria that are necessary to support correct two-level planning. The TLP algorithm is sound, but not complete, and we analyze and discuss these properties for TLP.

We build a case study composition problem domain with a large service set to illustrate and validate the TLP approach. We also implement the TLP algorithm by modifying the FF planner for performance evaluation. We compare the TLP solution with the base FF planner. The experimental result show that TLP outperforms its base FF planner significantly.

The rest of this paper is organized as follows. In Section 3.2, we introduce existing definitions for the planning domain and the planning problem and then define the two-level planning problem and the TLP algorithm. The service hierarchy construction mechanism is introduced in Section 3.3, including the service clustering algorithm, the formal requirements for virtual service construction for each service cluster, and the service hierarchy construction algorithm. In Section 3.4, the case study setup and performance evaluation for TLP are discussed. Section 3.5 concludes the paper and discusses future research directions.

1.13 Two Level Planning

In semantic web service models, such as OWL-S [15] and WSMO [31] [32], a web service can be defined by a tuple (I, O, P, E), where I is the input of the service, O is the 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. The OWL-S specifications for a service can be converted to PDDL specifications (for an action) [1] in order to apply AI planning for automated service composition.

The service composition problem can be defined as a planning problem in the planning domain as $\Sigma = (S, A, \gamma)$, where S is the finite set of all possible states, A is the finite set of services (actions in classical planning domain) that can be activated to change the system state, and $\gamma: S \times A \rightarrow S$ specifies the set of transitions caused by a service in A.

For more specific definitions, consider the set L=$\{p_1, p_2, \dots, p_n\}$ which is the set of n propositions used to define the states in S and $S \subseteq 2^L$. Let $Pre(a)$ and $Ef(a)$ denote the precondition and effect of service a, respectively, and $Pre(a), Ef(a)$ are defined by the propositions in L. The effect of a can include positive and negative effect propositions. Let $Ef^+(a)$ and $Ef^-(a)$ denote the positive and negative effect sets of a, respectively. A service $a \in A$ is applicable to $s \in S$ if $\forall p \in Pre(a) \Rightarrow p \in s$. In this case, $\gamma(s, a) = s' = \left(s - Ef^-(a)\right) \cup Ef^+(a)$; otherwise, $\gamma(s, a)$ is undefined. In other words, a proposition in $Pre(a)$ will be carried over to s' if it is not in the negative effect set. The positive effect of action a also exists in s', which means that $\forall p \in Ef^+(a) \Rightarrow p \in s'$. All the propositions in the negative effect set of action a will be removed from s', which means $\forall p \in Ef^-(a) \Rightarrow p \notin s'$.

Note that the input and output of service s can also be interpreted by preconditions and effects [2]. So, we only consider the precondition and effects of a service in the planning system.

The composition problem in a planning domain $\Sigma = (S, \mathrm{A}, \gamma)$ can be defined as $Q = (\Sigma, s_0, g)$, where:

$\Sigma = (S, A, \gamma)$ as discussed above,

$s_0 \in S$ denotes the initial state,

$g \in S$ denotes the goal states.

Both s_0 and g are specified by the propositions in L.

Suppose that the planning problem Q has a solution R = <$a_0, a_1, \dots \dots, a_k$>, which is a sequence of services bringing the system from the initial state s_0 to the goal state g, via intermediate states $s_1, s_2, \dots \dots, s_K$, we have

$\forall p \in Pre(a_0) \Longrightarrow p \in s_0$,

$\forall i, 0 \le i < K, s_{i+1} = (s_i - Ef^-(a_i)) \cup Ef^+(a_i)$,

$\forall p \in g \Longrightarrow p \in s_K$.

With a large number of services in A, the corresponding state space S can be very large and the composition problem Q may have a high complexity. We use the two-level planning (TLP) approach to reduce planning complexity. The first step toward TLP is to build a service hierarchy for all services in A and the hierarchy is defined in Figure 2. A is clustered into M clusters, $A_1, A_2, \dots, A_M$, where $A_i = \{a_{i,1}, a_{i,2}, \dots, a_{i,m_i}\}$, for all i. In the service hierarchy, under root A, there are M virtual parent services $va_i, 1 \le i \le M$. All the services in cluster A_i are children of va_i, and va_i represents all the services in cluster A_i.

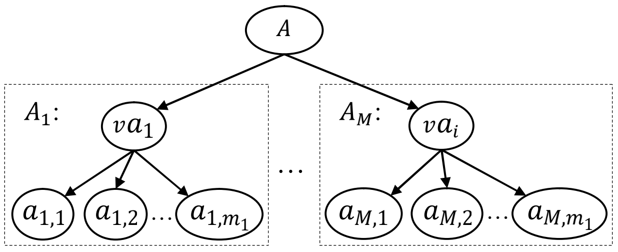

Figure 2. Service Hierarchy

Two-Level Planning (TLP) Algorithm:

Given a composition problem $Q = (\Sigma, s_0, g)$, with the service set A (from Σ) and A's hierarchy is as shown in Figure 2, the TLP algorithm can solve Q and obtain the composition R by going through two levels of planning, Level-1 and Level-2.

The Level-1 planning problem is Q^{l_1}=$(\Sigma^{l_1}, s_0^{l_1}, g^{l_1})$, where $s_0^{l_1}$ and g^{l_1} are refined initial condition and effect, respectively (to be discussed later). The set of services from Σ^{l_1} is $A^{l_1} = \{va_1, va_2, \dots, va_M\}$. An existing planner can be used to derive the composition for Q^{l_1}. Let R^{l_1} = <$va_{i_1}, va_{i_1}, \dots, va_{i_K}$> be the result composition, which is a sequence of K services. Note that for simplicity, we consider a service chain here. A complete workflow is integrated from multiple service chains derived for nondeterministic effects [80] and the same TLP can be used for separate derivation of all the chains.

The Level-2 planning problem is Q^{l_2}=(Σ^{l_2}, s_0, g). The set of services from Σ^{l_2} is $A^{l_2} = \bigcup_j A_j$, for all j, $va_{i_j} \in R^{l_1}$. A strong cyclic planner can be used to solve Q^{l_2} and derive the final composition result R.

To support the applicability of TLP, we need to guarantee its soundness and completeness. As usual, the planning solution should be correct. Also, if there exists a solution for a composition problem Q, TLP should guarantee that Level-1 planning can obtain a solution for Q^{l_1} and Level-2 planning can obtain a solution for Q.

The TLP algorithm is not a divide-and-conquer approach, but is a divide-and-elimination approach. Let $m = \underset{i}{avg}(m_i)$. The original composition problem Q involves $M \times m$ services, while Q^{l_1} involves M services and Q^{l_2} approximately involves $K \times m$ services. If we have a reasonable M/K from clustering, then the performance gain by TLP is significant.

TLP can be easily extended to multi-level planning, but it may not be useful in practice. Thus, we only consider two levels in this paper.

1.14 Service Hierarchy

In order to make TLP work, how should we construct the service hierarchy? Specifically, what property each cluster A_i and its virtual parent service va_i should satisfy?

First, consider an intuitive condition. If $\forall i$, $\forall j$, $Pre(va_i) \Rightarrow Pre(a_{i,j})$ and $Ef(a_{i,j}) \Rightarrow Ef(va_i)$ is satisfied, then as long as Level-1 plan R^{l_1} = $<va_{i_1}, va_{i_2}, \ldots, va_{i_K}>$ is derived, any service from A_{i_j} can be selected to instantiate va_{i_j}, for all j, to obtain the final composition. To precisely derive $Pre(va_i)$ and $Ef(va_i)$, the condition can be rewritten as

$\forall p \in Pre(a_{i,j}) \Rightarrow p \in Pre(va_i)$ and $\forall p \in Ef(va_i) \Rightarrow p \in Ef(a_{i,j})$.

However, there are several problems with this condition. First, to ensure that the service hierarchy does exist, service clustering should ensure that $Pre(va_i)$, for all i, should not have conflicting (mutex) propositions. And, it is likely that such service hierarchy cannot be obtained. Even if the hierarchy does exist for a composition problem Q, it is likely that we cannot obtain a Level-1 composition solution R^{l_1} when a composition solution R does exist.

To assure that a Level-1 composition R^{l_1} can be obtained by TLP as long as a composition solution R exists, we consider another intuitive condition

$\forall p \in Pre(va_i) \Rightarrow p \in Pre(a_{i,j})$, and $\forall p \in Ef(a_{i,j}) \Rightarrow p \in Ef(va_i)$.

Now we can assure that a Level-1 composition R^{l_1} can always be obtained by TLP as long as a composition solution R exists. But we still have the problem that it may not be possible to find a service hierarchy satisfying the condition. Also, even if the hierarchy does exist for a composition problem Q, without further constraint on the clusters, the Level-2 planning of the TLP algorithm cannot guarantee to find a solution when a solution for Q does exist.

As can be seen, how to build the service hierarchy is an important issue. In Section 3.3.1, we discuss the requirements for the service hierarchy to assure the applicability of TLP. In Section 3.3.2, we discuss an algorithm to build the service hierarchy that satisfies the requirements.

1.14.1 Virtual Service Requirement

By using subsume and encompass logic, we cannot obtain the proper requirement for the service hierarchy to assure TLP's completeness. Now we consider the issue from a practical point of view. When we try to simplify a complex composition problem, we can ignore some less important issues and focus on the major planning tasks first. For example, consider planning for a swarm of robots to perform survivor search in a rescue mission. We may focus first on the high level plan, such as assigning the search tasks robustly to the swarm, transporting the robots to near their search regions, planning the search path for each robot, etc., without considering some low level details. After the first level planning, we can then consider issues such as ensuring that the robots are working, are fully charged, are loaded with life detectors and luminance devices, etc. This means that we should prioritize the propositions, delaying some of them till the Level-2 planning.

We formally define the requirements for building the service hierarchy for the idea of proposition prioritization.

Service hierarchy requirement SHR-1:

$\forall i, j, \forall p, p \in Pre(va_i) \Rightarrow p \in Pre(a_{i,j}), \forall p, p \in Ef(va_i) \Rightarrow p \in Ef(a_{i,j}),$

$\forall i, \forall p, p \in Pre(va_i) \vee p \in Ef(va_i) \Rightarrow p \in L^{l_1}.$

Though it may be most effective to use human knowledge to make the priority decisions for propositions in L, we automatically prioritize propositions. We build the service hierarchy first. Then, the common propositions in each cluster are considered as Level-1 propositions L^{l_1}.

SHR-1 intentionally omits the mutual exclusive (mutex) criteria for $Pre(va_i)$ and $Ef(va_i)$, for all i, and we give it separately in SHR-2.

Service hierarchy requirement SHR-2:

$\forall i, \forall x, y, p_x \in Pre(va_i) \wedge p_y \in Pre(va_i) \Rightarrow p_x \wedge p_y \neq \emptyset$,

$p_x \in Ef(va_i) \wedge p_y \in Ef(va_i) \Rightarrow p_x \wedge p_y \neq \emptyset$.

However, with SHR-1 and SHR-2, TLP still may not work. If a proposition appears in $Pre(va_i)$ as well as in the $Pre(a_{j,k})$ of A_j, but got omitted in $Ef(va_j)$, then TLP may fail.

For example, in the Level-1 plan, we have effects $\{p_1, p_2\}$ from the virtual service va_1 and in Level-2 plan, effect of service $a_{1,1}$ is $\{p_1, p_2, p_3\}$. In the Level-2 plan, a virtual action va_2 will not be satisfied with precondition (p_1, p_2, p_3). And service $a_{2,1}$ will not be selected for the Level-2 planning. Then it may cause the service composition to fail.

Here, we define a new requirement to ensure that if a proposition p exists as precondition or effect in L^{l_1}, then it has to be considered in $Ef(va_j)$or $Pre(va_j)$, for all j, as long as p appears in the precondition or effect of at least one service in A_j.

Service hierarchy requirement SHR-3:

$\forall i, j, \forall p, p \in L^{l_1}$ and $p \in Ef\left(a_{i,j}\right) \Rightarrow p \in Ef(va_i)$,

$p \in L^{l_1}$ and $p \in Pre\left(a_{i,j}\right) \Rightarrow p \in Pre(va_i)$.

Here $L^{l_1}.Pre$ means that is precondition proposition set in Level-1 and $L^{l_1}.Ef$ is the effect proposition set in Level-1.

However, with SHR-3, TLP still may not work properly because of involving too many common prepositions. If a proposition appears in $Pre(va_i)$ as well as in the $Pre\left(a_{j,k}\right)$ of A_j and also appears in $Pre\left(va_j\right)$, then TLP may fail because of multiple choice. For example, in the Level-1 plan, we have the effect p after a step. In the Level-2 plan, it supposed to have an action $a_{i,j}$ to be satisfied under the cluster virtual parent service va_i. However there may be another va_j and $Pre(va_j)$ also includes p and can be satisfied. Then, in the Level-1 planning, va_i will not be guaranteed to be selected after the step i and the planning may fail.

Earlier SHRs focus on proper function of TLR. Now we consider an additional factor to improve the quality of service hierarchy. If a proposition appears in the precondition (or effect) of multiple virtual services, then it is not a dominating characteristic for the cluster. Thus, we also require that the proposition sets of different virtual services to be disjoint, which is stated in SHR-4.

Service hierarchy requirement SHR-4:

$$\forall i, j, i \neq j, \forall p, p \in Pre(va_i) \Rightarrow p \notin Pre(va_j)$$
$$\wedge\, p \in Ef(va_i) \Rightarrow p \notin Ef(va_j).$$

1.14.2 Service Hierarchy Construction

Now we discuss an algorithm for constructing a service hierarchy for a service set A, and it should satisfy SHR-1/2/3/4.

Similar to [46], we extract keywords from preconditions, effects, and service descriptions and put keywords from different sources into the corresponding bags. The bags of keyword vectors are fed to two learning algorithms for classification, including naïve Bayes and logistic regression implemented by Weka. We train an IOPE-based classifier and a description-based classifier separately. The classifiers are trained initially by a small size of labeled samples. After services are clustered, we decide the virtual parent service va_i for each cluster A_i. Specifically, we decide the best set of propositions for $Pre(va_i)$ and $Ef(va_i)$. Subsequently, from all clusters, we can decide L^{l_1}. The algorithm for deciding the precondition of va_i, for all i, is given as follows.

```
for each service cluster set $A_i$
  for each proposition $p$ in $A_\mathrm{i}$
    calculate $f(p)$
    // which is the frequency of $p$ as precondition in $A_\mathrm{i}$
    if $f(p) > \Delta$ and $\neg p \notin L_i^{l_1}$
      Add $p$ to $Pre(va_i)$
    else if $f(p) > \Delta$ and $\neg p \in L_i^{l_1}$
      $p_{max} = \max(f(p), f(\neg p))$
      Add $p_{max}$ $Pre(va_i)$
      and remove $\neg p_{max}$
    endif
  endfor
endfor
```

In the algorithm, we use a threshold Δ, to decide whether a proposition has occurred sufficiently often in the preconditions/effects of the services in A_i so as to be included in $Pre(va_i)/Ef(va_i)$, respectively. If two mutually exclusive propositions appear in $Pre(va_i)$ or $Ef(va_i)$ at the same time, then we should choose one of them to include in $Pre(va_i)$ or $Ef(va_i)$.

After $Pre(va_i)$ and $Ef(va_i)$ are determined, we move services out of A_i into A_{others} in case it causes A_i to violate SHR-1/2/3. The algorithm for service cluster adjustment is given as follows.

```
for each service $a_{i,j}$ in $A_i$
  if $\exists p, (p \in Pre(va_i)$ and $p \notin Pre(a_{i,j}))$
        or $(p \in Ef(va_i)$ and $p \notin Ef(a_{i,j}))$
    if $\forall p, p \in Pre(va_o) \Rightarrow p \in Pre(a_{i,j})$
      and $\forall p, p \in Ef(va_o) \Rightarrow p \in Ef(a_{i,j})$
      move $a_{i,j}$ to another cluster $A_o$
    else move $a_{i,j}$ to cluster $A_{others}$
    endif
  endif
endfor

for each proposition $p$ in $L^{l_1}$
  if $p \in Pre(a_{i,j}) \cup Ef(a_{i,j}) \wedge p \notin Pre(va_i) \cup Ef(va_i)$
    add $p$ to $va_i$
  endif
endfor
```

Next, we remove the common propositions in preconditions and merge the same clusters in case it causes A_i to violate SHR-4. The same operation should apply to remove the common propositions in effects also.

```
for each proposition p in $L^{l_1}$
  if $p$ appears in preconditions of multiple virtual services
     or $p$ appears in effects of multiple virtual services
    remove $p$ from $L^{l_1}$ and all the virtual services
 endif
endfor
for each pair of clusters $A_i$ and $A_j$
```

```
  if $Pre(va_i) = Pre(va_j)$ and $Ef(va_i) = Ef(va_j)$
    merge clusters $A_i$ and $A_j$
  endfif
endfor
```

Note that with SHR-3, a service will only appear in one and only one cluster. Consider a service a appearing in two clusters, $a \in A_i \wedge a \in A_j$, $A_i \neq A_j$. From SHR-1, we have $\forall p, p \in Pre(va_i) \vee p \in Pre(va_j) \Rightarrow p \in Pre(a) \wedge p \in L^{l_1}$. Because $a's$ precondition includes propositions from va_i and va_j and all these propositions are in L^{l_1}, from SHR-3, these propositions have to appear in both va_i and va_j because a is in both A_i and A_j, we can derive $Pre(va_i) = Pre(va_j)$. In the same way we can derive $Ef(va_i) = Ef(va_j)$. So, A_i and A_j have to be the same cluster and a only belongs to one cluster. Since the basic clustering algorithm will not place a service in multiple clusters, they are anyway disjoint.

Also, note that if $p_x \in Pre(va_i)$, and $p_y \in Pre(a_{i_j})$, for some a_{i_j} in A_i, and p_x and p_y are mutually exclusive, then definitely $p_x \notin Pre(a_{i_j})$, so a_{i_j} will not be in A_i. Thus, once SHR-2 is satisfied, the mutual exclusion problem will not occur between any service with its virtual parent, and will not occur among services within the same cluster.

1.14.3 Level-1 Planning Problem

The Level-1 planning problem $Q^{l_1}=(\Sigma^{l_1}, s_0^{l_1}, g^{l_1})$ has been discussed earlier, but $s_0^{l_1}$ and g^{l_1} have to be derived after the proposition prioritization. Essentially, we remove the propositions that are not in L^{l_1} from s_0 and g to derive $s_0^{l_1}$ and g^{l_1}. Formally, we have

$s_0^{l_1} = \{p \mid \forall p, p \in s_0 \wedge p \in L^{l_1}\}$ and $g^{l_1} = \{p | \forall p \in g \wedge p \in L^{l_1}\}$.

1.15 Case Study and Evaluation

We use a travel planning case study to illustrate the TLP planning algorithm and evaluate its performance.

We construct a set of services based on the public flight information and city transportation information as well as some service definitions available online. We first selected 7 categories. For each category, we build 10 to 5,000 services, depending on the characteristic of the category. The name of the category and the number of services in the category are given in Table 1. The overall testing service set consists of around 10k services with 125 propositions and 12 object types in their preconditions and effects. The numbers of possible values for the 12 object types are 2552.

Table 1. Service Clustering

Service Category	amount	Service Clustering	amount
Booking	2299	Booking	2299
Payment	10	Payment	10
Pricing	2293	Pricing	2293
Touring	10	Touring	10
Education	10	Education	10
Consulting	10	Consulting	10
Transportation	5251	Bus	102
		Flight-by-AA	1903
		Flight-by-UA	1629
		Flight-by-DL	1617

We use our service hierarchy construction algorithm to build the service hierarchy, including service clustering and derivation of preconditions and effects for the virtual parent services. The resulting service clustering is shown in Table 1. In Level-1 of the hierarchy, there are 10 virtual services, including the 7 categories we had originally and 3 additional clusters.

Since the transportation service cluster is very large, the algorithm further divided it into three clusters as show in the table.

We modified FF planner [8] into our TLP planner and compared the performance of TLP and the original planner for the service composition problem of the case study system. The experimental result is shown in Table 2. As can be seen from the table, our approach can significantly reduce the composition reasoning time for composition problems with a large set of services.

Table 2. Experimental Result

Planning Algorithm	Level-1 Planning	Level-2 Planning	TLP
Service Count	10	~3000	~3000+10
Time (s)	0.004	0.083	0.087

1.16 Conclusion

In this paper, we have proposed the multi-level service composition approach (specifically, we consider two-level planning) to achieve performance improvement for large-scale composition problems. We have developed a service hierarchy construction algorithm and modified the FF planner and converted into TLP planning. Experimental study show that the TLP approach can significantly improve the planning performance for composition problems with a large set of services.

Some potential future research directions include improving the service hierarchy construction method to get a better cluster size distribution with minimized dangling services (services that do not belong to any cluster). Also, we will modify the TLP algorithm to achieve the completeness property. In the current Level-1 planning process, the shortest path is selected as many planners do. Since the Level-2 planning domain is selected based on the virtual parent services along the shortest path, if the services in the corresponding clusters of these virtual services cannot offer a solution, then planning will fail.

There may be other issues in using TLP for service composition. The services in the real world are defined with different standards and follow different conventions. Thus, service hierarchy and virtual parent service construction may have practical issues. We plan to investigate these issues and build a comprehensive composition system for real world services.

In [80] [81], we have developed a model for multiple functionality service composition in which multiple goals composed by if-then-else construct is considered to offer flexible choices to users. We have also developed the planning algorithms to efficiently achieve multi-functionality service composition. We plan to integrate TLP into multi-functionality planning to address the potential performance issues in multi-functionality composition.

CHAPTER 4

AUTOMATED HOLISTIC SERVICE COMPOSITION: MODELING AND COMPOSITION REASONING

Authors – Wei Zhu, I-Ling Yen, Farokh B. Bastani

The Department of Computer Science, EC31

The University of Texas at Dallas

800 West Campbell Road

Richardson, Texas 75080-3021

1.17 Introduction

Service oriented architecture (SOA) has been widely adopted by many industrial companies as well as government agencies because it enables easy and extensive reuse. On the other hand, with the rise of cloud computing, many companies and organizations nowadays provide their software as SaaSs (software as a service) and Apps over the internet. With these technologies and provisioning, new system development can be completed much more rapidly by composing different services into a system.

A lot of research works have investigated the issues on how to compose services into an application system. SHOP2 [1] and OWLS-Xplan [2] are widely used systems for automated service composition. 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.

There are also automated service composition methods using search-based planning techniques [5] [6], similar to GraphPlan [7], Fast Forward (FF) Planner [8], Local Search for Planning Graphs with Action Costs (LPG) [9], Fast Incremental Planner (FIP) [10], etc.

There are planning techniques which can handle the non-deterministic effect and exceptions, and can be used for automated service composition. WarPlan-C [53] is the first planner that considers conditions. It use Prolog to reason for the plan with conditional branches. Similar to conditional planning, conformant planning focuses on uncertain initial states, i.e., the initial state consists of disjunctive clauses or may even be unknown. In [56], Graphplan is extended to be conformant. The disjunctive initial conditions are handled in a similar way as nondeterministic actions, where each condition is planned as a branch. The Planning solutions can be categorized into weak, strong, and strong cyclic plans in [61]. The planner in [53] [56] are all strong planners. Though a strong cyclic plan is a weaker plan than the strong plan, strong cyclic planning offers a solution when the strong planning cannot. Several strong cyclic planning algorithms have been proposed, including the Fast Incremental planner [10], Model Based Planner (MBP) [61], and NDP (Nondeterministic Planning) [62].

However, existing automated service composition techniques have omitted some important issues toward composing services into a holistic system. First, all these techniques consider a single system goal, like a plan does. 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 perform service composition 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 the composition effort and generate a well integrated workflow.

The second issue in automated service composition is the consideration for services with multiple effects. The output of a service may be used as a control parameter to determine what 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 is no customer remaining in the store. Then, a store closing service closes all the 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 to 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 of a system, 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 [18] [19], but some gaps still remain. One gap is that formal service definition models, such as OWL-S and WSMO, do not have a specific mechanism for specifying the multiple effects or exceptions of a service [15] [82]. Without a proper specification model, the techniques for handling them becomes ad hoc. Another important gap in the literature 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 service composition model that supports the specifications of goals for the exceptions.

The third gap in automated service composition is the consideration of alternative paths toward a goal. Most existing planning techniques, especially those used for automated service composition, only derive one path from the initial condition to the goal state. But 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. To facilitate a formal treatment of the holistic composition problem, we extend the classical OWL-S service model [15] with multi-effect specifications 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.

We also develop an 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.

The organization of this paper is as follows: In Section 4.2, our extended model for automated holistic service composition is introduced. The composition reasoning algorithms for achieving holistic service composition are presented in Section 4.3. In Section 4.4, we discuss a case study system to illustrate how to use our model for holistic service composition problem specification and to demonstrate how our composition reasoning algorithms can be used to obtain a holistic workflow for the composition problem. Section 4.5 concludes the paper and outlines some future research directions.

1.18 A Holistic Model for Service Composition

In the OWL-S model, a service can be defined by its (I, O, P, E), where I is the input of the service, O is the 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. This model is generally used to specify a concrete service, which can be an atomic or a composite service. In a service composition problem, the OWL-S service model can be extended to specify an abstract service that has not been grounded. The IOPE of the designated composite service can be specified to derive the automated service composition.

However, there are some shortcomings in the OWL-S model as well as some other existing service composition models. In this section, we develop a holistic service and composition model which extends OWL-S to facilitate automated composition of holistic workflows. Our extension include three directions:

(1) Extend the OWL-S service model to consider multiple effects and exceptions that may be raised in a service. The extended service model is discussed in Section 4.2.1.

(2) Introduce an "exception" model to support the specification of exceptions that may be raised in the system, i.e., may be raised by the services in a workflow. The model is discussed in Section 4.2.2.

(3) Introduce a "system" model to support clearer service composition problem specification. The model considers multiple system functionalities and other issues. Section 4.2.3 discusses this system model. In Section 4.2.4, we further discuss how to incorporate alternative methods for achieving the same functionality in a holistic workflow.

In each subsection, we discuss the reason why existing models are not sufficient and why the extension is needed. We also introduce the extended model formally and justify why the new model can better support the service and the service composition problem specifications.

1.18.1 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 output 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” $\vee$ “item-low-shelf-stock”

$\vee$ “item-spoiled” $\vee$ “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. The situations “item-spoiled” and “item-broken” can be handled by the same sub-workflow which checks the properties and sizes of the items and determines how to dispose them.

Handling multiple effects of a service have been widely considered in automated service composition literature [6] [66] [67]. The technique is to create multiple “virtual services” to represent one concrete service during the 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 predicate “item-misplaced”, InspectS-2 has effect predicate “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 effect specifications.

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 the exceptions of a service. Exceptions are different from effects. When a service raises an exception, the functionality of the service is not fulfilled since the execution is disrupted. For example, the InspectS service may raise exceptions “cart-noGas” due to shortage of gasoline in the inspection cart, “cart-malfunction” due to some malfunction in the inspection cart, or “camera-low-battery” or “camera-malfunction” due to the problems in the inspection camera. In these cases, inspection cannot be completed and some actions should be taken to take care of the exception condition. We will discuss the specific exception specifications at the system level. Here we only consider that each service can be associated with a set of exceptions and extend the OWL-S service model to support such specifications.

A partial view of the extended service model for holistic service specification is given in Figure 3. 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.

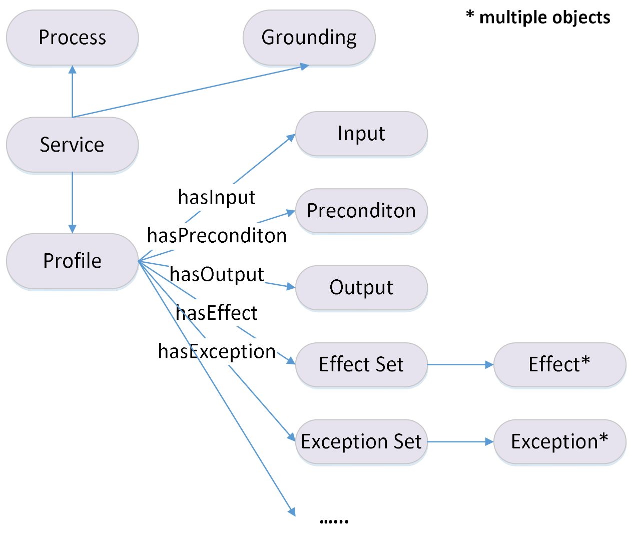

Figure 3. The Extended Service Model

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 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 $XS(s)$. □

1.18.2 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. Here we try to provide a mechanism 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 specification 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 the point of execution at which 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 with 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 discussed above, we define the exception ontology in Figure 4.

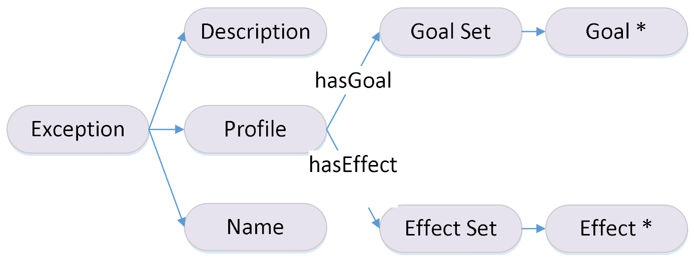

Figure 4. The Exception Model

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 specified in the system.

Based on the model given in Figure 4, 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 for exceptions) that should be achieved after e is raised, no matter which service raises it. □

The exception name and description are omitted in the tuple since they will not be used directly in the composition reasoning process, though they may be useful in other composition processes, such as service clustering to facilitate faster and easier service discovery.

1.18.3 Holistic System Specifications

In the OWL-S model, a composite service to be composed can have its IOPE specifications, which are used as the rules to govern the composition. Such model has the benefit of facilitating the specification of value-added services and facilitates 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 and their specifications are defined separately [82]. 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 some similarities, there are some 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 5.

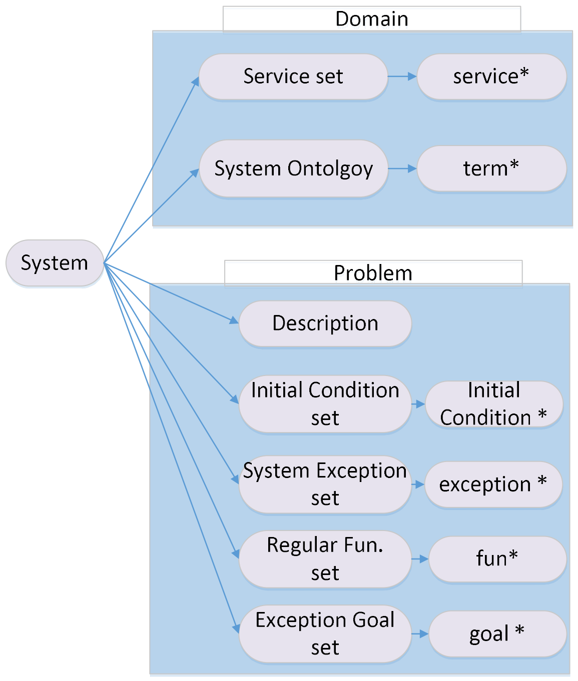

Figure 5. The System Model

We consider the “System” model in two views, namely, 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 “Initial Condition set” class has one or more “Initial Condition” subclasses. The “Initial Condition” is specified by a predicate that represents one of the initial conditions of the system. 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. Each “Goal” is specified by a predicate that represents one goal of the system. The “Workflow” class specifies the workflow that, staring 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. Instead of using the complex OWL-S process specification, we consider using BPMN workflow specification for the “Workflow” class.

In Definition 3, 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 $<IS, FS, SX\ \ , XGS, WF>$, 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. WF_i is the i-th version of the workflow during the composition process and WF_F denotes the final workflow obtained after the overall composition process is completed. □

Note that the literals used in the definition of $IS, XS, FS,\ XGS$ are defined in STS. Services selected to compose WF_i, for all i, are in SS.

In this paper, our discussion only involves a single composition problem under one composition domain and, hence, 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 has 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 the "Initial Condition Set" 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 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 of the system 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.

Also, the system functionalities cannot be specified as $F_1 \land F_2 \land \ldots \land F_N$ 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 $F_1$
elseif <$cond$> then $F_2$
…
else $F_N$
```

where each <$cond$> is independent and can be any condition to be derived in the holistic workflow.

Also, each individual effect 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 concept in the system model. Exceptions may have different scopes. 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 that is 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 with the individual services that have this exception. This is to avoid duplicated derivations of the exception handling workflows. System exceptions should of course be associated with the system, not with the individual services.

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

For all i, $EG_i(e) \in (FS \cup XGS)$, and the semantics for $EGS(e)$ should be

$$EG_1(e) \vee EG_2(e) \vee \ldots$$

Specifically, each goal in the goal set of exception 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.

1.18.4 Multiple Methods via Multiple Functionality Specification

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. These alternative ways are supposed to be incorporated in the final system workflow to offer users desirable choices.

One way to derive a system workflow which covers alternative methods for achieving a functionality is to let the composition reasoning algorithm find all possible paths and incorporate them in the workflow. However, not all alternative paths for achieving a functionality 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 as discussed in Section 4.3 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” $\land$ “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 can include the following goals:

F_1 = “home delivery” $\land$ “items delivered” $\land$ “payment succeed”

F_2 = “store pickup” $\land$ “items delivered” $\land$ “payment succeed”

…

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 one of the methods to achieve the functionality will be selected by the reasoning process. If we need to achieve a functionality with a specific method, then the goal specification for the functionality can include both the method-related predicate and the functionality-related goal predicate.

1.19 The Reasoning Process for Holistic Service Composition

Based on the service and service composition models defined in Section 4.2, we can reason and derive a holistic workflow systematically and automatically. We leverage existing service composition reasoning techniques as well as develop new techniques to achieve the 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 4.3.1, we focus on building the basic workflow for the system with multiple goals. Then in Section 4.3.2, we provide a systematic approach to handle the exceptions (external exceptions). In Section 4.3.3, we discuss the remaining steps to finalize the workflow generated for the composition problem.

1.19.1 Basic Multi-Functionality Reasoning

First, we discuss the algorithm for constructing a workflow WF_1 for the basic multi-functionality problem. In WF_1, multiple functionalities will be reached with branching paths, 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. Determine the initial condition IS of the target system. 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 original IS or the predicate "Initial". FS is a set of multiple goals of the system constructed in Step 1.

5. Use a multi-functionality planner, $MRPMF$ or $SGEMF$, to reason for the multi-functionality planning problem 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 $F_1, F_2, \ldots, F_N$. Note that the multiple 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 $\mathrm{F}_1, F_2, \ldots, F_N$ 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 $W_r = \{w_1, w_2, \dots, w_{m_r}\}$ 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 have 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 $br_{w_i,1}, br_{w_i,2}, \dots$, denote these paths. Let $(w_{i,x,1}, w_{i,x,2}, \dots, w_{i,x,nb_{i,x}})$ 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 $vs_{j,1}$ 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 $W_r = \{w_1, w_2, \dots, w_{m_r}\}$ 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 $E\mathrm{f}(s_j) = Ef(vs_{j,l})$ 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 now been processed. But there may still 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 multi-functionality plan constructed after exiting the loop above. Also, W_B is the set of all the services used in WF_B. It is possible that a service in SS is used more than once in WF_B. In this case, we consider them as different services in W_B because they will be treated differently when handling their exceptions.

1.19.2 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 set of system exceptions SXS and the set of exception goals XGS should have been defined (as discussed in Section 4.2.4). 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 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 $EGS(e_j)$ to them, where $EG\ \ (e_j) \subseteq FS \cup XGS$. 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 $\mathrm{W}F_{e_j}$ 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 4.3.1. First, we derive st_{w_i}, the state before w_i in WF_B is executed, the same way as stated in 6b of Section 4.3.1. Then $IS_{e_j} = (st_i - ES^-(w_i, e_j)) \cup ES^+(w_i, e_j)$.

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

3. Apply Step 6 in Section 4.3.1 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 exception 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 defined in SXS. Note that system exceptions are defined globally and may be raised at any time during the 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

$$P = \left(SS, IS_{e_j}, \mathrm{E}GS(e_j)\right)$$

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 4.3.1. We need to ensure that $EGS(e_j)$ is reached no matter when e_j is raised so we have $IS_{e_j} = \bigvee_{w_i} st_{w_i}$ (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, both from services and at the system level.

1.20 Case Study

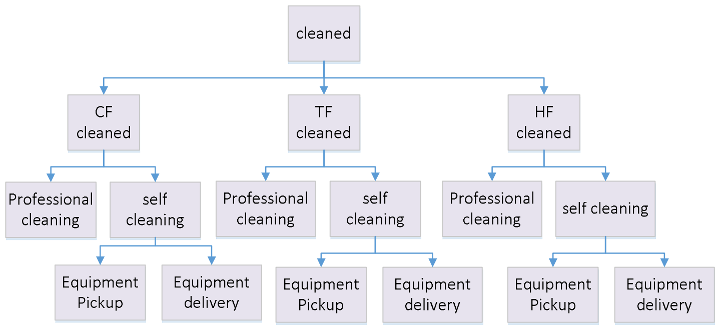

Figure 6. Multiple Functionalities of the System

Consider the online cleaning system discussed earlier. Different choices of functionalities are considered, including “carpet floor cleaning”, “tile floor cleaning” and “hardwood floor cleaning”. For each cleaning target, there are also choices of ordering a full cleaning services or renting the cleaning equipment for do-it-yourself cleaning. Also, if the user chooses to rent the equipment, there are choices of how to pick up the equipment, self-pickup or company delivery. These functionalities for the system (functionality set FS) are specified in Figure 6. The goal definitions for some functionalities in FS are formally specified as follows.

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

G_2 = “professional cleaning” ∧ “carpet cleaning”

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

……

G_6=“ professional cleaning” ∧ “hardwood cleaning”

Next, we apply the service composition reasoning algorithm discussed in Section 4.3.1 to obtain the basic workflow WF_B, which is shown in Figure 7. As can be seen, multiple functionalities are modeled as user choices in conditional branches.

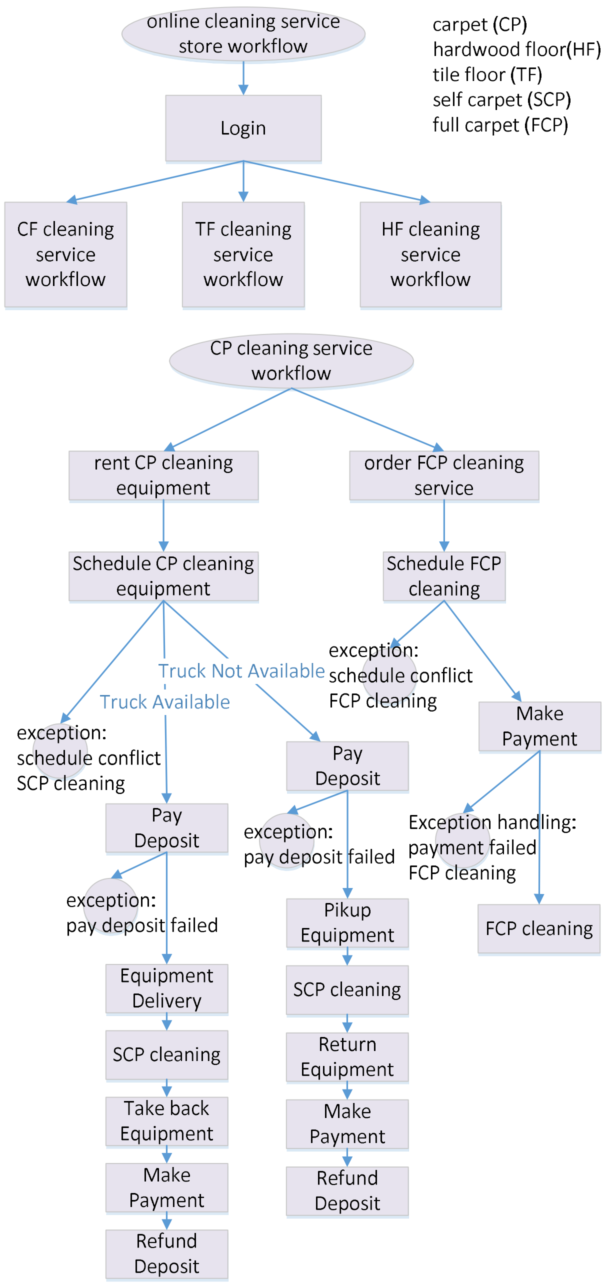
online cleaning service store workflow
carpet (CP)
hardwood floor(HF)
tile floor (TF)
self carpet (SCP)
full carpet (FCP)
Login
CF cleaning service workflow
TF cleaning service workflow
HF cleaning service workflow
CP cleaning service workflow
rent CP cleaning equipment
order FCP cleaning service
Schedule CP cleaning equipment
Schedule FCP cleaning
Truck Not Available
Truck Available
exception: schedule conflict FCP cleaning
exception: schedule conflict SCP cleaning
Make Payment
Pay Deposit
Pay Deposit
exception: pay deposit failed
Exception handling: payment failed FCP cleaning
exception: pay deposit failed
Pikup Equipment
FCP cleaning
Equipment Delivery
SCP cleaning
SCP cleaning
Return Equipment
Take back Equipment
Make Payment
Make Payment
Refund Deposit
Refund Deposit

Figure 7. Cleaning Workflow

Now we can follow the algorithm discussed in Section 4.3.2 to derive the exception handling sub-workflows for the exceptions raised by “Make Payment”, “Schedule FCP cleaning”, “Schedule CP cleaning equipment” and “Pay Deposit” services. Two example exception handling workflows are shown in Figure 8 and they can be integrated into the main workflow in Figure 7.

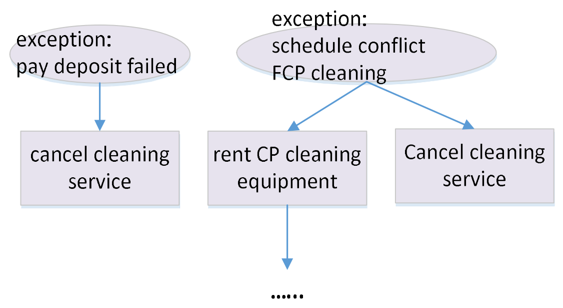

Figure 8. Exception Handling

1.21 Conclusion

In this paper we have proposed the concept of multi-functionality service composition in the attempt of automatically composing services into a complex system with multiple functionalities. Many real world systems have interdependent multiple functionalities and can make use of our solutions. We have developed a model to facilitate the specification of the multi-functionality composition problem and developed algorithms for composition reasoning.

There are several potential future research directions. First, this includes improving the planning techniques for multi-functionality composition reasoning. The improvement should focus on efficiency of the multi-functionality planning as well as optimized output workflow, such as maximal degree of overlapping for multiple functionalities, minimal number of services, etc.

Also, the service composition framework for multi-purpose systems discussed in this paper lays a foundation for the composition of complex systems. However, there may be other issues in service composition of complex systems. For example, the system may have multiple threads of execution and the automatically derived workflow should support user switching among different threads. We plan to investigate a set of real-world applications that are suited for SOA design and investigate additional issues in the automated service composition models and techniques.

CHAPTER 5

A PT-SOA MODEL FOR CPS/IOT SERVICE

Authors – Wei Zhu, Guang Zhou, I-Ling Yen, Farokh B. Bastani

The Department of Computer Science, EC31

The University of Texas at Dallas

800 West Campbell Road

Richardson, Texas 75080-3021

1.22 Introduction

In recent years, internet-of-things (IoT) and cyber-physical system (CPS) have gained a lot of attention. Both IoT and CPS consider a vast number of static and/or mobile physical devices networked together into a system. It is difficult to draw a clear line between CPS and IoT. In general, both deal with the same system configuration, but CPS research focuses more on the control of the huge-scale system, while IoT focuses more on the identification, control, and communication of individual devices.

Due to the increasing varieties and number of physical things (PTs) being integrated into the cyber world, it is important to consider dynamic discovery and composition of them to greatly enhance the capability of IoT and CPS. Most existing CPS/IoT research do not consider these capabilities. Service computing researchers have applied service computing technologies in CPS/IoT to achieve rapid discovery and composition of physical things for new or dynamically arising tasks [20] [24].

The fundamental entity for service discovery, selection, and composition is the service model. Existing service models are mainly designed for software services and are not suitable for services provided by CPS/IoT. One of the major differences is the role of the physical thing (PT) that provides the service. In software services, the PT is the computing and storage hardware. However, due to the sufficient uniformity in the computing facilities for software services and the high speed communication among them, though there are still issues like communication costs and workloads, the PTs for software services do not have a significant role. In CPS/IoT, the PT that provides a service and its properties are very important. In the following, we discuss the major issues in CPS/IoT services:

(1) The physical characteristics of a PT can impact the service it provides. For example, different types of vehicles can be used to transport people from a disaster site to a safe evacuation area. But each type of vehicle has its own characteristics, such as load capacities and number of seats. Also, even for the vehicles that are exactly the same, when grounding the service for transporting people, it is necessary to specifically determine the number of vehicles required and the number of trips each vehicle may have to make.

(2) A PT may be able to provide several different types of services. However, it is frequently not possible for one PT to fulfill multiple services it provides at the same time.

(3) In a rescue mission, some robots may be used for survivor detection. The physical location of the robots must be at the rescue site. If not, additional services are required to bring them to the rescue site.

(4) The side effect of a software service generally can be specified independent of other software services. This may not be true in physical things. For example, a car may transport a robot to a disaster site for a rescue search. In this case, the states of the service provider and the PTs the recipient of the service is may change together. Such impact need to be specified explicitly and existing software service models do not have such a feature.

(5) Granularity is also an issue in CPS/IoT systems. (a) For example, a swarm of robots may provide some services as one unit. But each robot may also be used to provide different services. (b) Many PTs may have components. Generally, they provide services as single units, but require separate maintenance services or control software.

Some device specific specification models have been proposed, e.g., DPWS by OASIS [27]. These models support general description of a device and the services it hosts. But these models do not provide a solution to the issues discussed above. For (1) and (2), it is better to define PTs and services separately and associate them in the upper ontology. In DPWS, etc., services become a secondary entity. Thus, the same (or similar) services by different devices have no association, making service discovery more complex. Defining both PTs and services in the upper ontology also facilitates high level reasoning of services while handling the PT specific issues at a lower level. Another problem with these existing device specific models is that their device specification models are over simplistic. PT specific characteristics that are required for proper service selection and reasoning are not considered in these models. For example, for (3), "context requirements" should be specified in the PT specification. It may be argued that the "context requirements" for the PTs can be specified as a pre-condition for their services. However, these requirements are dynamic (depending on the invocation task), while precondition and effects of software services are generally static. It is better to define the "context related" requirements separately from regular pre-conditions and effects. For (4), for some services, it is necessary to define their recipient PTs (not services) and the effects on them specifically. For (5), (5b) has a static set of components while the constituent PTs in (5a) can be composed dynamically. For both cases, a two-level service definition, including the services by the group (swarm) and by/for the individual constituent components, is necessary. For (5a), the two-level specification also facilitates encapsulation and dynamic PT selection. When a swarm service is selected, the detailed control can be embedded within the service, instead of requiring the specification of complex collaboration between services. The swarm specification should include the selection criteria for the constituent PTs, instead of the static set of constituent PTs, such that PTs can be selected dynamically when the swarm service is selected. Also, separation of service and PT is important such that when a PT is used in a swarm, it will be clear that it should not be selected to provide other individual services.

There have been some service modeling for CPS/IoT systems. SENSEI [20] proposes a resource model to publish the PTs as services. Resource specification is based on OWL-S with keywords. SOCRADES [22] proposes a two-level discovery model. It supports the high level service discovery based on conventional service specifications. From the selected service, the low level devices that satisfy the service description can be selected. The selection can be based on location or QoS attributes. Hydra [23] categorizes common PTs into a class hierarchy and defines semantics based on the hierarchy to facilitate IoT service discovery. In [24], imprecise service specifications and potentially inaccurate sensing are considered. It proposes a probabilistic model to capture the fuzziness due to these problems and supports probabilistic discovery. Though some of the issues and solutions considered in these works are interesting, but none of them directly deal with the issues we identified above. Some other works, such as SATware [83], focus on semantic representation of sensor data, not services provided by PTs. In [84], the concept of contexts is used for service discovery. Also, context-based preconditions and effects are used to assist the composition process. However, these models only address one or two issues above and a more sophisticated model is needed.

Our goal is to build a PT-SOA model to properly model services and PTs in CPS/IoT systems and to facilitate service discovery and composition. We leverage existing models and extend them for CPS/IoT. In Sections 5.2, 5.3, and 5.4, our new PT-SOA model is defined. In Section 5.5, we use a case study to show how our PT-SOA model is useful and essential in CPS/IoT service selection and composition.

1.23 Upper Ontology in the PT-SOA Model

As discussed in the introduction, in CPS/IoT, it is necessary to consider PTs explicitly. Also, the recipient PTs should be explicitly specified in the model. In Figure 7, an upper level ontology for CPS/IoT services is presented. In this model, PTs are considered at the same level as the services. They can be specified independently, but are linked together with several relations. Each PT can "Provide" a set of services and each service can be "ProvidedBy" one or more PTs. A service provided by a PT may be "AppliedTo" some PTs and the state of these PTs may be impacted upon execution of the service.

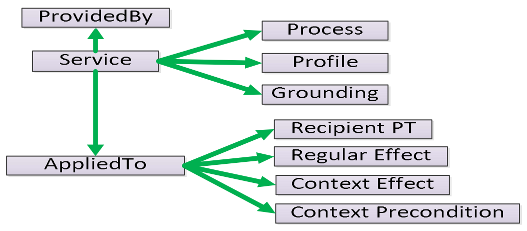

Figure 9. CPS /IoT Upper Ontology Model

"AppliedTo" is an important relation in the composition reasoning process. Consider a transportation service. After its execution, the states of the service provider and the service recipients may change. If the service recipients are local to the service, then their state changes can be captured within the service without any problems. If the service recipient is a PT that is independent of the provider, then it will be harder to define the state change of the PT. Consider another example. A gas station may provide an "AddFuel" service. After its execution, the state of the recipient PT may change from "InsufficientFuel" to "SufficientFuel". In many existing service models, Transport or AddFuel is specified as a local service to the recipient. Such models can be confusing and can make the composition reasoning difficult. One possible solution for the problem is to treat the recipient PT as an input parameter to the service and specify the state changes of the recipient accordingly. This solution may not be possible if the PTs are not modeled as an independent entity in the SOA model. Even if the PT is modeled as a separate entity, the service composition process will have to consider composition through input parameters, making composition more complex. In our PT-SOA model, we introduce a class “AppliedTo” to link a service to one or more recipient PTs. Due to the use of this relation in the upper ontology, the reasoning process can be done clearly by following the “AppliedTo” link.

Separation of services and PTs allows clean specification of the PTs and services and supports encapsulation in the composition process, especially in automated composition. To achieve a desired goal, the services can be composed independently of the PTs first. Then, the specific PTs and their properties and states can be considered afterwards.

Following the upper PT-SOA ontology, we need to define the PT model and the service model. Section 5.3 focuses on the definition of the PT model, i.e., the PT-ontology. The extension for the service model in the PT-SOA model is presented in Section 5.4.

1.24 PT-Ontology in the PT-SOA Model

Figure 8 defines the overall PT-ontology. The properties of a PT are specified by four classes (besides the services), namely “Physical Profile”, “Operation Profile”, “Operation Schedule” and “Context”. In the following subsections, the four classes are discussed in details.

1.24.1 Physical Profile

For each PT, it may be necessary to specify its physical characteristics that impact its operations, PT selection, and service invocation. Also, as discussed in the Introduction, the components and/or constituent PTs of a PT may play some roles in its operation and need to be properly specified. These information are maintained in “Physical Profile”.

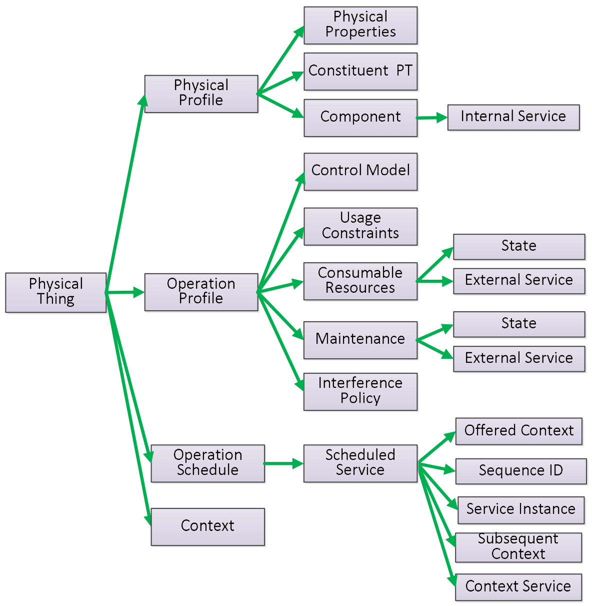

Figure 10. PT Model in PT-SOA Model

Physical Properties: For each PT, it is necessary to specify its “Physical Properties” that may impact its operations or services. For example, a sedan may have 2, 4, or 5 seats, which should be specified to allow proper PT selection and invocation of its service. Other properties that may be helpful in identifying the PT or of potential interest to the service receivers can also be included in this class.

Constituent PT: Sometimes, a group of PTs may work together as a unified entity, and can be viewed as a higher level PT. If the group always works together and individual entities are not of significance, then specifying the single high level PT would be sufficient. If the grouping of PTs may change or the individual entities may provide their own services, then it is necessary to define the individual PTs as well as the grouped entity. Thus, PT ontology should support a hierarchical specification. The “Constituent PT” class supports the specification of the constituent PTs of a PT. For example, in a swarm of robots, each robot may collaborate and provide services on its own. Also, the swarm grouping may change while performing a task. At the same time, the entire swarm can have its own services and the internal operational details can be encapsulated. So the swarm and the constituent robots will form two levels of PTs.

Component: Generally, a PT may have many components, and they can be specified by the "Component" class in the physical profile. Also, components of a PT may provide services which contribute to the realization of an external service of the PT. We use the "Internal Service" class to support the specification of the services provided by the components of a PT. Whether such components and services should be specified with the PT is scenario dependent. If only the external service the PT provides is of interest, then there is no need to expose the internal services. When the focus is on the internal control in order to achieve different variations of the external service, then specifying the internal micro services may be beneficial. Consider the brake and the accelerator of a car which can help achieve "Transportation" service. Generally, the car offers services at macro scale and the "Transportation" service is sufficient without needing to specifically micro-control the acceleration and deceleration of the car. However, when a potential collision is detected, it may be beneficial to expose the brake component and its deceleration service to achieve proper control that is specifically tuned to the capability of the brake component. Note that the brake is a component, not a constituent PT of the car, because its service is only realized through the car. On the other hand, the seats of the car are unlikely to be of external interests and there is no need to expose them. There are yet other components of the car, such as cameras (many modern cars are equipped with them), which can offer their services independently, not interfering with the external services of the car, can simply be treated as separate PTs, even if they are physically inseparable from the car (can just consider the situation as though the camera is not attached to the car but is in the car).

The choice of what to include in the “Physical Profile” is usage dependent and can be designed by the specifier for the PT. The principle is to keep the ontology simple, yet provide detailed specification when they are useful.

1.24.2 Operation Profile

Each PT may need specific control mechanisms, may need some resources during its operation, may need regular maintenances, and may have some physical constraints or limitations when providing services. We define all these operation related properties of the PT under the “Operation Profile” class in the following subsections.

Usage Constraints: Operation profile specifies the properties that may impact the operation of the PT and its services. During service provision, there may be constraints on how these services can be provided. For example, most of the services provided by one PT will have to be provided exclusively, i.e., if the PT is providing one service, then it cannot be used to provide another. Such constraints have to be specified for each PT locally and we use “Usage Constraints” class in PT profile to describe these types of constraints. Also, for each service a PT provides, there may be specific constraints which are not common for the service. For example, both helicopter and ambulance can provide the rescue service for a person injured in a disastrous event, but a helicopter can provide the service in floods while an ambulance cannot. A sedan may provide passenger transportation service for at most 4 passengers and a commercial van may provide the same service for 10 passengers. Such constraints can be specified using the “Usage Constraints” class.

Control Model: As discussed in Section 5.3.1, a PT-Ontology model may include its constituent PTs or components and their services. Thus, it may be necessary to define the control model to specify how the constituent PTs should collaborate or how the components should be controlled to achieve the high level services. In PT-Ontology model, the “Control Model” class describes the internal control for the Constituent PTs and/or components. The “Control Model” can be defined as a control workflow composed of internal services or as a control program that invokes the internal services or simply as a high level description of the control mechanism. As with the components and internal services, the control model only needs to be specified when it is useful in the application scenario.

Interference Policy: When selecting PTs for their services, it is necessary to consider their usage constraints. When a PT is actually in action for service provision, there may be interferences with other PTs and policies can be defined using the “Interference Policy” class to resolve the potential interference problem. Consider a simple example of two buses offering the transport service on the same route. If these two buses meet at some station along the route, the interference policy may require that the second bus waits for some time before continuing its service to increase the time span of service provision by the two buses.

Consumable Resources: Generally, the operation of a PT may consume some resources. For example, a car needs to consume gas to offer its “Transportation” service. A life detector needs to consume battery to provides “life detecting” service. A printer needs to consume electricity as well as ink and paper to provide its “printing” service. These resources are provided externally by other services and, hence, need to be exposed specifically. We include

"Consumable Resources" class in the operational profile to specify the resources required for proper operation of the PT.

Maintenance: Similar to consumable resources, external maintenance services are required to ensure the proper operation of a PT. The "Maintenance" class is included in the PT ontology for specifying required maintenance services and maintenance schedules.

Both "Consumable Resources" and "Maintenance" classes, if specified, should include state variables to describe the status of the consumable resources (such as the level of the gas tank of a car) and the status of each maintenance service (such as the date the maintenance service has been most recently performed and the time period during which it has to be performed next) and the services that can be used to maintain the sufficiency of the consumable resources and the services that perform the maintenance. The state variables may be used in preconditions of some of the services of the PT. It may also be updated after the associated external services are performed. Due to their importance in reasoning, we include the "State" variable and "External Service" classes in the PT-Ontology model.

Consumable resource and Maintenance classes can also be skipped in a PT-Ontology model if they are not of interest in the application scenarios. For example, when selecting a car to perform a service, it is essential to confirm that it has enough gas or can easily access a gas station service to keep it running. But maintenance of the car is generally of less concern since it is not time-critical. Also, when selecting a printer for its printing service, it may not be interesting to confirm whether it has its power source since most likely it is statically connected to its power source.

1.24.3 Context

Generally, a PT has some “constant” properties and some dynamic properties that may change all the time. The “Physical Properties” class specifies the constant properties of the PT, while the “Context” class defines the specific state of the PT in terms of those dynamic properties.

In the physical world, the context of the PT is very critical in service provision. For example, it is not realistic to expect to use a PT in Florida to fulfill a service that is needed in California within an hour. Consider that a service consumer requests for a CPS/IoT service s at location A. A PT y that can provide service s is at location B, and y is not mobile. Then, a transportation service which can bring y from B to A should be composed with y.s (service s provided by PT y) in order to fulfill the request. This scenario generally does not need to be considered in software services, but is required in a PT-SOA model. Here, we have defined the “Context” class to specify the current context of a PT. Later, we define “Offered Context” to define future contexts that the PT should satisfy for its service provisions.

1.24.4 Scheduled Service

Unlike software services, in which a service can be provided simultaneously to multiple requesters who may invoke the service from any geographical locations, services provided by a PT may have to be provided exclusively and have time, location, and other potential context requirements (context preconditions, will be defined later). Thus, scheduling has a significant role in a PT-Ontology model. A PT, if required to offer its service(s) to multiple requesters and the service(s) should be offered mutual exclusively, then the PT has to be scheduled properly to serve those requests.

We define the “Scheduled Service” class in our PT-Ontology model. It has subclasses “Service Instance”, “Sequence ID”, “Offered Context”, “Subsequent Context”, and “Context Service”. “Service Instance” is the reference to the specific service of the PT that the PT is scheduled to satisfy. Sequence ID is used to clearly order the service offerings, and can be realized, from the implementation point of view, by linked list or other data structures for convenient retrieval. “Offered Context” is the context that the PT can offer to satisfy the requested context and “Subsequent Context” is the context after the scheduled service completes. The “Context Service” class specifies the service that can transfer the PT from the “Subsequent Context” of the previous “Scheduled Service” (by Sequence ID) to the “Offered Context” of this scheduled service.

For example, a technician (PT) provides a concrete elevator-repair service. An office may request for the service on a specific day at a specific address. The technician, after scheduling, may offer to provide the service at a certain time range, say 2-4pm, of the day. The location of the service will, of course, be at the given address. The technician may have many such “Scheduled Services”. The “Context Service” can transfer the technician from one location to another. Consider a PT “Hotel”, which has several meeting rooms and offers a concrete meeting service. A person requesting for the meeting services has a room size and time-period requirement. In this case, the hotel can schedule a specific room for the specified time-range to satisfy this meeting service request. As can be seen, the “Offered Context” should always imply the requested context. The “Offered Context” may be a specialization in location or in time or in both of the requested context.

1.25 Extended Service Model for CPS/IoT

In our PT-SOA model, we keep the “Process”, “Profile” and “IOPE” from the conventional OWL-S model, and extend them with new CPS/IoT specific information. We extend service ontology with “Context Precondition” and “Context Effect” classes (shown in Figure 9) to facilitate the specification of dynamic requirements and subsequent effects for each service of the PT (discussed in Sections 5.4.1 and 5.4.2). To specify the effects of a service on its recipients, we define the “AppliedTo” class under the service ontology, which is discussed in Section 4.4.3.

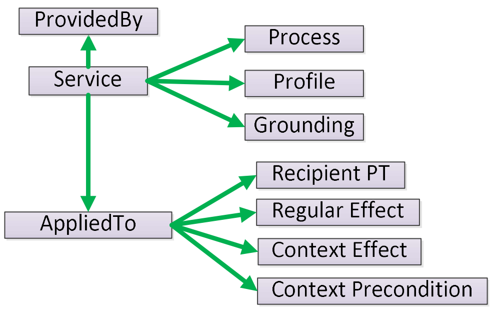

Figure 11. Service Extension for Context and Effect

1.25.1 Context Precondition

As discussed in Section 5.3, when a service is invoked, some context requirement may have to be satisfied in order to proceed with the service provision. We define the “Context Precondition” class to specify such requirements. Note that context preconditions cannot be specified as conventional preconditions. Generally, preconditions of a service are fixed conditions that stay the same for all service invocations. But context preconditions are dynamic, probably different in each invocation.

In the service paradigm, a request for a service can be specified as an abstract service with IOPE being the requirements for match making. Similarly, in service composition, an abstract service may be decomposed into multiple abstract services with a process model specifying their control flow relations. Each abstract service can go through the same match making process as above. The extension in the PT-SOA model is that the invocation-specific context preconditions should also be specified in the abstract service model. Consider the match making process where a concrete service or a "more specialized" abstract service, say sc, is to instantiate the abstract service, say sa.

- In conventional service paradigm, sc can instantiate sa if sc's precondition subsumes sa's precondition and sc's effect implies sa's effect.
- In PT-SOA model, sc can instantiate sa if sc's precondition subsumes sa's precondition and sc's effect implies sa's effect and let pt be the provider of sa, either of the following conditions should hold:
- If pt and sc are abstract, then sa's context precondition should subsume sc's context precondition; or
- The current context of a concrete pt satisfies sa's context precondition; or
- There exists a "Scheduled Service" in a concrete pt whose "Offered Context" satisfies sa's context precondition, or
- A "Scheduled Service" y with an "Offered Context" that can satisfy sa's context precondition can be added to a concrete pt, where y's prior "Scheduled Service"

is x, and there exists another service sb, such that sb can transfer pt from x's "Subsequent Context" to y's "Offered Context".

Though context preconditions need to be satisfied by the PT's context, they are the requirements for the individual service instances to be provided by the PT. Hence, they should be specified under service definitions, not directly associated with the PT.

1.25.2 Context Effect

Effect is the condition that will hold after the execution of a service, if the precondition is satisfied. Effect specification has been widely investigated for software services (post-conditions). Formal effect specification needs to be rigorous and can be highly complex for real systems. Semantic based effect specification has been used for services. Descriptive logical terms that are not directly related to the program itself can be defined to capture the concept of the effect of a service. Such specification can be used in PT-SOA as well. However, there are issues for effect specification in the model that need to be specially addressed.

As analyzed earlier, precondition alone is not sufficient for specifying all requirements that have to be satisfied before service execution. We have defined context precondition to capture the dynamic context requirements that are only known upon service invocation. Correspondingly, we define the Context Effect to capture the corresponding effect on context. However, once context precondition is known, the context effect can be defined correspondingly. We can express context precondition as an unknown variable, and define the context effect correspondingly. For example, consider a service s provided by pt. We can define:

Context Precondition: At (s.pt, \$x) $\wedge$ ServeAt (s, \$t) $\wedge$ Within (\$x, \$area) $\wedge$ Within (\$t, \$T)
Context Effect: At (s.pt, \$x) $\wedge$ EndTime (s, \$t + e) $\wedge$ Within (\$e, [2..3]seconds)

Here At specifies the location the pt should be at and ServeAt specifies the time at which s should be activated. Within specifies a range for any variable (of polymorphic type). Note that context effects can be specified just like regular effects. We still keep them separate from regular effects to facilitate potential separation in reasoning.

1.25.3 "AppliedTo" Concept

A software service has effect on the state of the system, such as database updates and parameter passing among services. One service may change the state of another service, but only through passing input parameters. In the physical world, a service may change the state of the provider, the external environment, as well as other physical things. In a simple model, a service may only be associated with its provider. In our PT-SOA model, the recipient PTs of a service are also specified through the "AppliedTo" class.

With the "AppliedTo" list, the "Precondition", "Context Precondition", "Effect", and "Context Effect" can be defined for the system state as well as for the PTs in the "AppliedTo" class. For example, a transportation service provided by a vehicle not only changes the context of this vehicle but also other PTs that are loaded on this vehicle. All the PTs loaded on the vehicle as well as the vehicle itself are all considered as the receivers of the service. Thus, the effect on all the PTs can be clearly specified based on the "AppliedTo" class. Consider a bus providing a transport service from Washington DC to New York City. The service is applied to all the riders, who are specified in the "AppliedTo" class. The "Context Precondition" is that all the riders should be in the Washington DC bus station and the "Context Effect" is that all the riders will be in the New York City bus station. The precondition and effect can be specified to include individual PTs or all PTs in the "AppliedTo" class.

1.26 Case Study and Model Validation

We use emergency rescue as a case study system. The first 24 hours after an earthquake is the most critical time for a rescue mission. Planning on using the physical resources for timely discovery of the survivors is highly critical. We illustrate how our PT-SOA model can be used for such planning and demonstrate its necessity and effectiveness in CPS/IoT service composition.

Figure 10 shows the map of the earthquake site (disaster region DR) and the neighboring facilities related to the rescue mission, including the hospital (H), robot storage facility (R), and truck station (T). The distances between the sites are marked in the map. We consider a highly simplified scenario. The rescue mission starts at 10:00am and is expected to be completed by midnight of the same day. There are 20 robots in R equipped with life detectors, 1 ambulance from H which has no other scheduled service, and 1 truck in T that has 2 other already scheduled services. The goal is to find all the injured survivors in DR and take them by ambulance to the hospital.

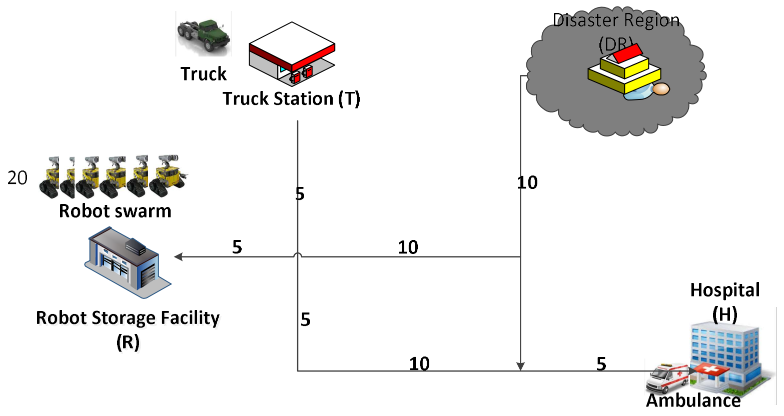

Figure 12. Earthquake Scenario

1.26.1 Specification of the PTs and CPS/IoT Services

We follow the PT-Ontology to define the robots, the truck, and the ambulances. The robots can provide individual services and can also be grouped into a swarm and provide services. The specification of "*swarm*" and "*robot*" PTs are given as follows. Some details are omitted due to space limitation. We use the notations discussed in Sections 5.3 and 5.4 and borrow PDDL [85] [86] like notation to express the predicates. The first identifier of a predicate is the predicate name and it is followed by its parameters. An identifier preceded with a question mark represents a variable; otherwise, a value or an instance.

```
swarm swarm.Provide: search-survivor
swarm.Physical Profile.
  Constitute PT: robot and number(robot) = 20 and
      ∀robot in swarm, robot.schedule = null
swarm.Context: At(R)

robot.Provide:
  remove_debri, detect_survivor, report_injured, move
robot.Physical Profile.
  Operation Schedule.Scheduled Service: null
robot.Context: At(R)
robot.Operation Profile
          UsageConstraints: move.distance < 1 mile

truck.Provide: transport, load
truck.Physical Profile:
          Operation Schedule.ScheduledService:
                    9:30am, transport(S1,D1);
                    1:00pm, transport(S2,D2);
fuel-adding
truck.Context: At(T)

ambulance.Provide: move-patient, load-patient
ambulance.Operation Profile:
          Usage Contraints: move-patient.number(patient) ≤ 2
ambulance.Physical Profile:
          Patient Capability: 2
          Operation Schedule.Scheduled Service: null
ambulance.Context: At(H)
```

The specifications of some services given above are defined as follow. We only give the detailed specification for some services and the rest are just mentioned.

```
swarm.search-for-life:
        Input: region, num-robot, start-time
        Precondition: need-search-for-life
        Context Precondition:
                start-time < 2pm
                At ?swarm ?region
 Effect: when (At ?human ?region)
                        (human-discovered ?region ?human)

robot.scan:
  Precondition: human-discovered ?region ?human
  Effect: human-visibility-determined ?human
                 human-visible ?human  /human-invisible ?human
  Context Precondition: At ?robot ?region
  Context Effect: At ?human ?region

robot.remove-debri:
        Precondition: human-discovered ?region ?human
     human-visibility-determined ?human
     human-invisible ?human
        Effect: human-visible ?human
        Context Precondition: At ?robot ?region

robot.report:
  Output: human-status
        Precondition: human-discovered ?region ?human
     human-visibility-determined ?human
     human-visible ?human
 Effect: human-status-determined ?human
                 human-injured ?human
                /human-not-injured ?human
Context Precondition: At ?robot ?region

truck.transport:
 Input: from_location, to_location
 Context Precondition: at ?from_location
 AppliedTo: on-vehilce ?load ?vehicle
 Context Effect: At ?vehicle ?to_location
                        At ?load ?to_location

        truck.load:
 Context Precondition: At ?vehicle ?location
                        At ?load ?location
 Context Effect: on-vehicle ?load ?vehicle

ambulance.load-patient:
 Context Precondition: At ?ambulance ?location
                        At ?human ?location
  Context Effect: on-ambulance ?human ?vehicle

ambulance.move-patient
        Input: human-status
        Precondition: human-status-determined ?human
                      human-injured ?human
  AppliedTo:  on-ambulance ?human ?ambulance
        Effect: at-hospital ?human
        Context Precondition: At ?human ?loc
                              At ?ambulance ?loc
                              on-ambulance ?human ? ambulance
        Context Effect: At ?human  ?loc_hospital
                         At ?ambulance ?loc_hospital
```

1.26.2 Regular Workflow Composition

The initial state of the system can be defined as:

Initial State: need-search-for-life

The goal discussed earlier can be formally defined as:

```
Goal: forall (?human - human)
        ( when  ( and (human-discovered ?region ?human)
                (human-injured ?human))
        ( and   (at-hospital ?human)  )
```

Based on the initial condition and the goal, we first use the regular pre-conditions and effects of the services defined in 5.5.1 (note that the context preconditions and effects are not considered yet) to construct the basic workflow. The resulting workflow is shown in Figure 11.

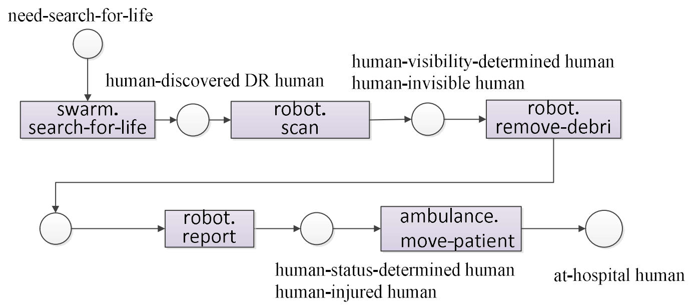

Figure 13. Basic Rescue Workflow

The initial state "*need-search-for-life*" can be matched with the precondition of "*swarm.search-for-life*". After search, the predicate "*human-discovered*" will match the precondition of service "*robot.scan*". Then, the effect of robot.scan is "*human-visibility-determined*" and it will trigger the service "*robot.visible*". Its result "*human-visibility-determined*" is the precondition of "*robot.report*". After robot.report, the state will be changed to "human-injured". Then the precondition of "*ambulance.move-patient*" is satisfied and the patient will be "at-hospital".

1.26.3 Composition Using the PT-SOA Model

In this section, we illustrate the use of our extended PT-SOA service model for complete workflow construction and why the PT-SOA features are useful. Note that in the workflow in Figure 12, the service "*swarm.search-for-life*" links to PT swarm, which are required to be grounded. Since there are 20 robots, they will be selected and grounded to the "Constitute PT" of the swarm. Also, the Context Precondition of the service "*swarm.search-for-life*", (At swarm DR), is not satisfied. Since the robots of the swarm have physical constraints (*move.distance* < 1 mile) and the distance from DR to Hospital H is 15 miles, the *robot.move* service cannot be used, and *truck.transport* service should be used to satisfy this context precondition. However, the truck has been scheduled for 2 other services (given in 5.5.1).

The goal is to schedule the *swarm.search-for-life* service between 10am to 2pm. From truck.Schedule, this service can be added in between, and the resulting schedule is as follows.

```
truck.Physical Profile:
        Operation Schedule.ScheduledService:
                9:30am, transport(S1,D1);
   10:20am, transport(R,DR);
                1:00pm, transport(S2,D2);
```

Based on the schedule, the workflow for the truck that are relevant to the rescue mission can be constructed as follows:

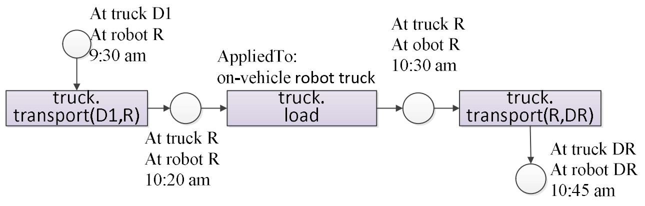

Figure 14. Truck Transportation Workflow

Note that the ContextPrecondition for truck.load requires that truck.Context: (At truck R). To satisfy it, *truck.transport*(D1,R) needs to be used to bring the truck to location R. At R, truck.load service loads robots to truck. Next, the *truck.transport*(R,DR) service will have its AppliedTo link linked to all the robots in the swarm. Finally, *truck.transport*(R,DR) delivers the robots from R to DR to satisfy the precondition of "*swarm.search-for-life*". Thus, after executing the workflow, the context of the truck and the robot will both be (At robot DR) and (At truck DR).

The composition using ambulances is similar to that for the truck. However, the Operation Constraint of the ambulances need to be considered in the composition. Assume that there are 6 survivors detected. Then, the *ambulance.move-patient* in the workflow in Figure 13 needs to be modified to include 3 rounds of the *ambulance.move-patient* service. The modified workflow from the single *ambulance.move-patient* will be as follows:

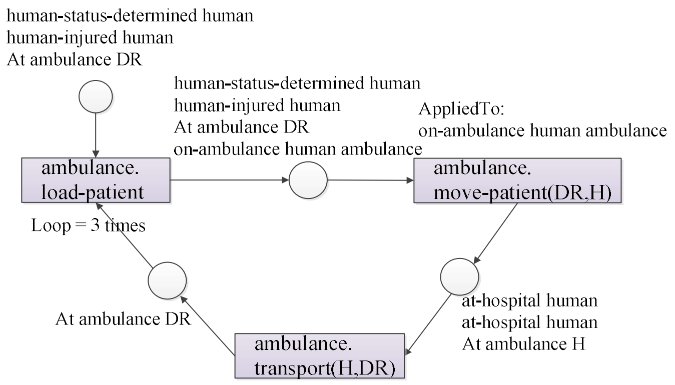

Figure 15. Update Rescue Workflow

1.26.4 Remarks

Due to space limit, we omitted some detailed composition reasoning steps using our PT-SOA model. Here, we discuss some of the omitted steps at a high level: (1) In the truck PT-ontology, there is a “Consumable Resource” = fuel. In case the truck does not have enough fuel, some external services will be used to add fuel to the truck to make its “Consumable Resource” condition be satisfied. This will trigger the requirement for composing additional external services to make the PT usable. (2) The “Maintenance” for ambulance will be linked to two maintenance services provided by two different providers, one maintenance service for the engine provided by a car technician and the other for checking the medical facilities on the ambulance, which is provided by the hospital. These services will be specified in the “ Maintenance” classes. (3) When using the *swarm.search-for-life* service, we did not consider how the constituent PTs are selected, which should be done in a complete composition process.

1.27 Conclusion

We have identified the issues in the modeling technology for discovery and composition of services and physical things to achieve new and dynamically arising tasks. Then, we extended OWL-S and other service models to build the PT-SOA model for CPS/IoT systems to address these issues. Finally, we presented a simple case study system to illustrate the necessity and effectiveness of the PT-SOA model in CPS/IoT service composition.

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 Fall Symposium on Semantic Web and Agents*, 2005.

[3] S. R. Ponnekanti and A. Fox, ""Sword: A developer toolkit for web service composition," in *International World Wide Web Conference*, 2002.

[4] S. McIlraith and T. C. Son, "Adapting golog for composition of semantic web services," in *KR*, 2002.

[5] L. A. da Costa, P. F. Pires and M. Mattoso, "Automatic composition of web services with contingency plans," in *IEEE International Conference on Web Services*, 2004.

[6] D. Berardi, D. Calvanese, G. D. Giacomo and M. Mecella, "Composition of services with nondeterministic observable behavior," in *International Conference on Service-Oriented Computing*, 2005.

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

[8] J. Hoffmann, "FF: The Fast-Forward Planning System," *AI magezine,* p. 57, 2001.

[9] A. Gerevini and I. Serina, "LPG: A Planner Based on Local Search for Planning Graphs with Action Costs," in *the Sixth International Conference on Artificial Intelligence Planning and Scheduling Systems*, 2002.

[10] 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 *International Joint Conference on Artificial Intelligence*, 2011.

[11] C. Fellbaum, WordNet., Blackwell Publishing Ltd, 1998.

[12] A. V. Paliwal, B. Shafiq, J. Vaidya, H. Xiong and N. Adam, "Semantics-based automated service discovery," *IEEE Transactions on Services Computing,* pp. 260-275, 2012.

[13] J. Sangers, F. Frasincar, F. Hogenboom and V. Chepegin, "Semantic Web service discovery using natural language processing techniques," *Expert Systems with Applications,* pp. 4660-4671, 2013.

[14] Z. Cong and A. F. Gil, "Efficient web service discovery using hierarchical clustering," *Agreement Technologies,* pp. 63-74, 2013.

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

[16] J. De Bruijn, C. Bussler, J. Domingue, D. Fensel, M. Hepp, M. Kifer and B. K.-R. e. al., "Web service modeling ontology (wsmo)," 2005.

[17] J. Peer, "A PDDL based tool for automatic web service composition," in *International Workshop on Principles and Practice of Semantic Web Reasoning*, 2004.

[18] H. Yang, X. Zhao, C. Chao and Z. Qiu, "Exploring the connection of choreography and orchestration with exception handling and finalization/compensation," in *International Conference on Formal Techniques for Networked and Distributed Systems*, 2007.

[19] K. Christos, V. Costas and G. Panayiotis, "Enhancing BPEL scenarios with dynamic relevance-based exception handling," in *IEEE International Conference on Web Services*, 2007.

[20] M. Presser, P. M. Barnaghi, M. Eurich and C. Villalonga, "The SENSEI project: Integrating the physical world with the digital world of the network of the future," *IEEE Communications Magazine,* pp. 1-4, 2009.

[21] D. Guinard, V. Trifa, S. Karnouskos, P. Spiess and D. Savio, "Interacting with the SOA-based Internet of Things: Discovery, query, selection, and on-demand provisioning of web services.," *Transaction on Service Computing,* p. 223, 2010.

[22] A. Cannata, M. Gerosa and M. Taisch, "SOCRADES: A framework for developing intelligent systems in manufacturing," in *IEEE International Conference on Industrial Engineering and Engineering Management* , 2008.

[23] M. Eisenhauer, P. Rosengren and P. Antolin, "Hydra: A development platform for integrating wireless devices and sensors into ambient intelligence systems," *The Internet of Things,* pp. 367-373, 2010.

[24] T. Teixeira, S. Hachem, V. Issarny and N. Georgantas, "Service oriented middleware for the Internet of Things: A perspective," *Towards a Service-Based Internet,* pp. 220-229, 2011.

[25] Y. Zhang, L. Duan and J. L. Chen, "Event-driven soa for iot services," in *IEEE International Conference on Service Computing*, 2014.

[26] S. Alam, M. M. Chowdhury and J. Noll, "Senaas: An event-driven sensor virtualization approach for internet of things cloud," in *IEEE International Conference on Networked Embedded Systems for Enterprise Applications*, 2010 .

[27] T. Nixon, "OASIS Devices Profile for Web Services (DPWS) Version 1.1," 2009. [Online].

[28] H. J. La and S. D. Kim, "A service-based approach to designing cyber physical systems," in *2010 IEEE/ACIS 9th International Conference In Computer and Information Science (ICIS)*, 2010.

[29] H.-C. Hsieh, C. Kai-Di, W. Ling-Feng, C. Jiann-Liang and C. Han-Chieh, "ScriptIoT: A Script Framework for Internet-of-Things Applications.," *IEEE Internet of Things Journal,* pp. 628 - 636, 2015.

[30] E. Christensen, F. Curbera, G. Meredith and W. Sanjiva, "Web services description language (WSDL) 1.1," 2001. [Online].

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

[32] C. Feier and J. Domingue, "WSMO Primer," 2005.

[33] Pahl, Claus and Y. Zhu, "A semantical framework for the orchestration and choreography of web services," *Electronic Notes in Theoretical Computer Science ,* pp. 3-18, 2006.

[34] A. Alves, A. Arkin, B. B. Sid Askary, F. Curbera, Y. Goland and N. K. e. al., " Web services business process execution language version 2.0," 2006.

[35] Z. Song, A. A. Cárdenas and R. Masuoka, "Semantic middleware for the internet of things," in *Internet of Things* , 2010.

[36] G. Denker, D. Elenius and D. Martin, " OWL-S Editor," in *International Semantic Web Conference*, 2004.

[37] A. Haller, E. Cimpian, A. Mocan, E. Oren and C. Bussler., "Wsmx-a semantic service-oriented architecture," in *International Conference on Web Services*, 2005.

[38] C.-H. Lee, S.-Y. Hwang, I.-L. Yen and T.-K. Yu, "A service pattern model for service composition with flexible functionality," *Information Systems and e-Business Management,* pp. 235-265, 2015.

[39] R. Tang and Y. Zou, "An approach for mining web service composition patterns from execution logs," in *IEEE International Symposium on Web Systems Evolution (WSE)*, 2010.

[40] Q. A. Liang, J.-Y. Chung, S. Miller and Y. Ouyang., "Service pattern discovery of web service mining in web service registry-repository," in *IEEE International Conference on e-Business Engineering*, 2006.

[41] J. Fu, F. B. Bastani, I.-L. Yen and W. Hao, "Using service patterns to achieve Web service composition," in *IEEE International Conference on Semantic Computing*, 2009.

[42] B. Orriëns, J. Yang and M. P. Papazoglou, "Model driven service composition," in *International Conference on Service Oriented Computing*, 2003.

[43] J. B. Warmer and A. G. Kleppe, The Object Constraint Language: Precise Modeling With Uml (Addison-Wesley Object Technology Series), 1998.

[44] R. Grønmo and M. C. Jaeger, "Model-driven semantic web service composition," in *12th Asia-Pacific Software Engineering Conference*, 2005.

[45] A. Sabraoui, A. Ennouaary, I. Khriss and M. E. Koutbi, "An MDA-Based Approach for WS Composition Using UML Scenarios," in *International Conference on Information Technology: New Generations (ITNG),*, 2012.

[46] A. Heb and N. Kushmerick, "Automatically attaching semantic metadata to Web Services.," 2003.

[47] R. Nayak and B. Lee, "Web service discovery with additional semantics and clustering," in *International Conference on Web Intelligence*, 2007.

[48] R. Gunasri and R. Kanagaraj, "Natural Language Processing And Clustering Based Service Discovery," *International Journal of Technology Enhancements and Emerging Engineering Research,* pp. 28-31, 2014.

[49] T. Wen, G. Sheng, Y. Li and Q. Guo, "Research on Web service discovery with semantics and clustering," in *IEEE Joint International on Information Technology and Artificial Intelligence Conference*, 2011.

[50] K. Elgazzar, A. E. Hassan and P. Martin, "Clustering WSDL Documents to Bootstrap the Discovery of Web Services," in *ICWS*, 2010.

[51] Y. Y. Du, Y. J. Zhang and X. L. Zhang, "A Semantic Approach of Service Clustering and Web Service Discovery," *Information Technology Journal ,* p. 967, 2013.

[52] B. Bonnet and H. Geffner, "HSP: Heuristic search planner," in *AIPS*, 1998.

[53] D. H. Warren, "Generating conditional plans and programs," in *2nd Summer Conference on Artificial Intelligence and Simulation of Behaviour*, 1976.

[54] M. Peot and D. Smith, "Conditional nonlinear planning," in *Proc. First International Conference on Artificial Intelligence Planning Systems*, 1992.

[55] R. P. Goldman and M. S. Boddy, "Conditional Linear Planning," in *AIPS*, 1994.

[56] D. E. Smith and D. S. Weld, "Conformant graphplan," in *AAAI/IAAI*, 1998.

[57] L. Pryor and G. Collins, "Planning for contingencies: A decision-based approach," *Journal of Artificial Intelligence Research,* pp. 287-339., 1996.

[58] R. I. Brafman and G. Shani, "A Multi-Path Compilation Approach to Contingent Planning," in *AAAI*, 2012.

[59] J. Hoffmann and R. Brafman, "Contingent planning via heuristic forward search with implicit belief states," in *International Conference n Automated Planning and Scheduling*, 2005.

[60] J. Hoffmann and R. I. Brafman, "Conformant planning via heuristic forward search: A new approach," *Artificial Intelligence,* pp. 507-541, 2006.

[61] 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.

[62] U. Kuter, D. Nau, E. Reisner and R. P. Goldman., "Using classical planners to solve nondeterministic planning problems.," in *The International Conference on Automated Planning and Scheduling*, 2008.

[63] D. Nau, Y. Cao, A. Lotem and H. Munoz-Avila, "SHOP: Simple hierarchical ordered planner," in *International Joint Conference on Artificial intelligence*, 1999.

[64] E. Sirin, B. Parsia, D. Wu, J. Hendler and D. Nau, "HTN planning for web service composition using SHOP2," *Web Semantics: Science, Services and Agents on the World Wide Web,* pp. 377-392, 2004.

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

[66] M. Pistore, P. Traverso and P. Bertoli, "Automated Composition of Web Services by Planning in Asynchronous Domains," in *ICAPS*, 2005.

[67] N. Milanovic and M. Malek, "Search strategies for automatic web service composition," in *International Journal of Web Services Research (IJWSR)*, 2006.

[68] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam and Q. Z. Sheng, "Quality driven web services composition," in *International Conference on World Wide Web*, 2003.

[69] L. Zeng, B. Benatallah, A. H. Ngu, M. Dumas, J. Kalagnanam and H. Chang, "Qos-aware middleware for web services composition," *IEEE Transactions on Software Engineering,* pp. 311-32, 2004.

[70] T. Yu, Y. Zhang and K.-J. Lin, "Efficient algorithms for Web services selection with end-to-end QoS constraints," *ACM Transactions on the Web ,* p. 6, 2007.

[71] G. Canfora, M. D. Penta, R. Esposito and M. L. Villani, "An approach for QoS-aware service composition based on genetic algorithms," in *7th Annual Conference on Genetic and Evolutionary Computation*, 2005.

[72] T. Gao, H. Ma, I.-L. Yen, F. Bastani and W.-T. Tsai, "Toward QoS analysis of adaptive service-oriented architecture," in *International Workshop on Service-Oriented System Engineering*, 2005.

[73] H. Ma, F. Bastani, I.-L. Yen and H. Mei, "QoS-Driven Service Composition with Reconfigurable Services," *IEEE Transactions on Services Computing,* pp. 20-34, 2013.

[74] W. Wang, Q. Sun, X. Zhao and F. Yang, "An improved particle swarm optimization algorithm for QoS-aware web service selection in service oriented communica-tion," in *International Journal of Computational Intelligence Systems*, 2010.

[75] G. Kang, J. Liu, M. Tang and Y. Xu, "An effective dynamic web service selection strategy with global optimal QoS based on particle swarm optimization algorithm," in *Parallel and Distributed Processing Symposium Workshops & PhD Forum* , 2012.

[76] M. Vukovic and P. Robinson, "Context aware service composition," University of Cambridge, Computer Laboratory, 2007.

[77] S. N. Han, G. M. Lee and N. Crespi, "Semantic context-aware service composition for building automation system," *IEEE Transactions on Industrial Informatics,* pp. 752-761, 2014.

[78] M. Kirsch-Pinheiro, Y. Vanrompay and Y. Berbers, "Context-aware service selection using graph matching," in *Non Functional Properties and Service Level Agreements in Service Oriented Computing Workshop*, 2008.

[79] K. Fujii and T. Suda, "Semantics-based context-aware dynamic service composition," *ACM Transactions on Autonomous and Adaptive Systems (TAAS),* p. 12, 2009.

[80] W. Zhu, J. Huang, F. B. Bastani and I.-L. Yen, "Multi-purpose Planning for Practical Web Service Composition Problems," in *2013 Fifth International Conference on Service Science and Innovation*, 2013.

[81] W. Zhu, F. B. Bastani, I.-L. Yen and J. Fu, "Automated Holistic Service Composition: Modeling and Planning Techniques," in *University of Texas at Dallas*, 2017.

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

[83] L. M. Pryor and G. Collins, "Cassandra: Planning for contingencies," Northwestern University, 1993.

[84] J. Huang, F. Bastani, I.-L. Yen and J.-J. Jeng, "Toward a smart cyber-physical space: a context-sensitive resource-explicit service model," in *COMPSAC* , 2009.

[85] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld and D. Wilkins, "PDDL-the planning domain definition language.," 1998.

[86] D. Pellier, "PDDL4j," 2011. [Online]. Available: http://sourceforge. net/projects/pdd4j.

BIOGRAPHICAL SKETCH

Wei Zhu was born in XuYi, Jiangsu, China. After completing his schoolwork at XuYi Senior High School in XuYi in 2002, Wei entered Nanjing University. He received Bachelor of Science with a major in computer science and a Master of Engineering with a major in software engineering from Nanjing University. During the following two years, he was employed as a software developer at Lucent Technologies in Nanjing. In 2010, he entered the graduate school at the State University of New York at Buffalo. In 2011, he transferred to graduate school at The University of Texas at Dallas.

CURRICULUM VITAE

Address: 800 W Campbell Rd, Richardson, TX, 75080
Email: wxz094120@utdallas.edu

EDUCATION
PhD Candidate in Computer Science, Aug.2011 - present, The University of Texas at Dallas
PhD Candidate in Computer Science, Aug.2010 - May.2011, State University of New York at Buffalo
Master Degree in Software Engineering, Sept.2002 - Jun.2006 Nanjing University, China
Bachelor Degree in Computer Science and Technology, Sept.2002 - Jun.2008 Nanjing University, China

TEACHING ASSISTANT EXPERIENCE
Cloud Computing, Spring 2013 & Fall 2014,
Advanced Operating System, Spring 2012
Compiler Construction, Fall 2011
University of Texas at Dallas

Advanced Computer Networking, Spring 2011
State University of New York at Buffalo

Programming Language (Java), Fall 2006
Computer Organization, Spring 2007
Software Institute, Nanjing University

PUBLICATION
[1] Nidhiben Solanki, Wei Zhu, I-Ling Yen, Farokh Bastani, Elham Rezvani: Multi-Tenant Access and Information Flow Control for SaaS. ICWS 2016
[2] I-Ling Yen, Wei Zhu, Farokh Bastani, Yongtao Huang, Guang Zhou: Rapid Service Composition Reasoning for Agile Cyber Physical Systems. SOSE 2016
[3] Wei Zhu, Guang Zhou, I-Ling Yen, Farokh B. Bastani: A PT-SOA Model for CPS/IoT Services. ICWS 2015
[4] I-Ling Yen, Guang Zhou, Wei Zhu, Farokh B. Bastani, San-Yih Hwang: A Smart Physical World Based on Service Technologies, Big Data, and Game-Based Crowd Sourcing. ICWS 2015
[5] Wei She, Wei Zhu, I-Ling Yen, Farokh Bastani, Bhavani Thuraisingham: Role-Based Integrated Access Control and Data Provenance for SOA Based Net-Centric Systems, Transaction on Service Computing
[6] Wei Zhu, Guang Zhou, I-Ling Yen, San-Yih Hwang: A CFL-Ontology Model for Carbon Footprint Reasoning. ICSC 2015
[7] Wei Zhu, Jian Huang, Farokh B Bastani, I-Ling Yen: Multi-Purpose Planning for Practical Web Service Composition Problems. ICSSI 2013
[8] Wei Zhu, Guang Zhou, I-Ling Yen, San-Yih Hwang: Toward Ontology and Service Paradigm for Enhanced Carbon Footprint Management and Labeling. ICWS 2013
[9] Jian Huang, Wei Zhu, Farokh B. Bastani, I-Ling Yen, Jicheng Fu: Automated Exception Handling in Service Composition Using Holistic Planning. CSE 2012