A Framework for Efficient Service Composition in Cyber-Physical Systems

Jian Huang, Farokh B. Bastani, I-Ling Yen, Wenke Zhang
Department of Computer Science
University of Texas at Dallas
{jxh072000, bastani, ilyen, wxz072000}@utdallas.edu

Abstract—Service-oriented architecture (SOA) provides the concept of packaging available functionalities as interoperable services within the context of various domains that use it. With rapid advances in SOA technologies and the growing availability of web services, the problem of composing a set of web services to achieve complex systems is becoming more practical. In the context of cyber-physical systems where hardware and software are coupled together to realize integrated systems, there are special characteristics and requirements. One of these is that most service providers are physical entities with their own states and properties. The constraint that follows is that a given entity might not be able to perform all the services it can provide at the same time. In fact, “multi-threading” for physical entities is rarely possible. This requires specific service modeling techniques to enable the use of SOA methods for this domain. Another characteristic is that due to the dynamic nature of cyber-physical worlds, the service composition procedure must be dynamically adaptive. In terms of AI planning, which is one of the fundamental techniques for service composition, not only the initial state and the goal are dynamic, but the planning domain also needs to be generated dynamically to provide the complete input to the underlying planner.

Taking all these characteristics and requirements into consideration, we develop an ontology model for physical entity specification. Based on this model, another widely used service ontology model, OWL-S is extended to accurately model the characteristics of service providers in the context of cyber-physical worlds. Further, a technique for generating planning domain based on task requirements is developed.

Keywords—Cyber-physical systems; service-oriented architecture; service composition; AI planning techniques.

I. INTRODUCTION

A cyber-physical system (CPS) is a system featuring the integration of a variety of physical entities with diverse functionalities. The economic and societal potential of CPS is believed to be tremendously greater than what has been achieved by existing systems in terms of autonomy, adaptability, efficiency, flexibility, and versatility. However, many technical challenges need to be addressed before we can take full advantages of CPS. One of the major challenges is just-in-time assembly of networked physical entities into desired capabilities.

To facilitate rapid integration of physical entities to achieve desired tasks, it is necessary to model these entities and support discovery and composition of their capabilities. Existing service-oriented architecture (SOA) can be used for the modeling, management, and integration of physical entities in cyber-physical systems. The capabilities of the physical entities can be wrapped as services and SOA technologies, such as service discovery and composition, can be applied. However, SOA models do not provide full solution and some new concepts and extensions are required. Also, the SOA model for CPS should include a (unified) modeling approach to facilitate automated composition. Specifically, we need to consider

- Specification of the physical entity (PE) and the services it provides. In existing models, the service provider (the PE) is not well defined. The providers of software services may not be important. But the provider of physical services has critical physical meanings. For example, a PE can only be at one location at one time and cannot provide its services out of its context.
- Specification of tasks. Task specification is critical in automated composition. The specification model should allow easy correlation of tasks and services.

Another important technique to facilitate the integration of the capabilities of the PEs is automated composition. AI planning techniques have been widely considered in automated service composition. In [1, 2] the Golog language is adapted for automatic composition of Web services. PDDL based Web service composition has been studied in [3]. Waldinger presents an approach for composition based service synthesis by theorem proving [4]. With the advent of web 3.0 [5], semantic specifications have been defined for web services and used to assist automated service composition. In [6], OWLS-XPlan has been developed to compose semantic web services. It converts the given initial and goal ontologies in OWL and services in OWL-S [7] into equivalent domain and problem descriptions in PDDL [8], and returns a composition plan sequence representing the composition. The logic involved in precondition, postcondition, and goal condition specifications are called PDDXML, which is essentially an XML [18] version of the PDDL language. In [9], the hierarchical task network (HTN) planning technique has been developed to perform composition of services automatically.

All the planning based service composition techniques consider simple cases, and have faced challenges in composing software services for real-world problems. This may be attributed to the fact that some software services are highly complex and their preconditions and effects are difficult to specify. But when considering domain-specific physical systems, there have been some success in
automated composition of real-world cases. For example, in the MER 2003 project, a planner is used to dynamically compose the rover’s operations to perform various scientific tasks under resource and environment constraints [10]. Generally, the services provided by physical entities are more specific and their effects can be clearly specified based on the changes they cause in the physical environment. Thus, automated composition in CPS is promising. However, directly applying planning techniques on physical services still will not work. In fact, naïve usage of planning techniques for automated CPS composition may result in a more complex problem space than for software systems. This is due to the large number of physical entities in general cyber-physical systems. Thus, some advanced filtering process to select the relevant PEs for a give task is needed to reduce the problem space and make planning feasible.

In this paper, we address the challenges of automated service composition in cyber-physical systems. First, we consider the modeling and specification of CPS. We extend existing service paradigms to build a provider explicit model for PE specification. For each PE, some extensions are introduced to specify their properties, especially the representation of the state of the PE instance. After providing services, the state properties of the service provider PE as well as other PEs in the system may change, which is specified as the effect of the service. Correspondingly, the goal of a task is specified using a set of PE states. When a task arises, the goal of the task will be matched against the effects of certain services provided by some PEs.

With a proper specification model for the PEs and the tasks, the automated composition can be performed using existing planning techniques. However, as discussed earlier, there will be a huge number of physical entities in the system and each may provide different services. Thus, the search space for the planner is prohibitive. We address this problem by performing pre-selection to eliminate irrelevant PEs. To facilitate the pre-selection process, we define a relevancy metric, which is defined recursively. For a given task, those PEs that can be used directly to achieve the task are given a unit of relevant metric. The relevant PEs are placed in a “selected” set. Other PEs that can affect the state of the “selected” PEs are considered related to the task indirectly and are placed in the “selected” set. This process is repeated recursively to obtain all relevant PEs. Then, the services of the relevant PEs are given to the AI planner to derive a composition solution.

The rest of this paper is organized as follows. In Section 2, we present our provider-centric specification model, which extends the OWL-S model for service, PE, and task specifications. In Section 3, we describe our service composition architecture and define the relevance metric. In Section 4, we evaluate our approach using a scenario where a solution is generated in case of a fire outbreak. Finally, Section 5 summarizes the paper and outlines some future research directions.

II. SERVICE MODEL

Our service model contains four components and their relationships are depicted in Figure 1.

At the bottom of Figure 1, there is a PE specification. Based on the specification, existing service class is extended with provider and receiver, both of which are PEs. By the side of the PE specification there is a task specification. The task model is new and needed because, again, existing service models do not have well defined ways for goal specification. In our goal model, the PE definition and status can be referred.

Based on the PE and goal specification, we develop a logic model to provide theoretical foundation for relevant PE selection. The logic model supports the specification of direct links between PEs and tasks, including the direct links from a PE to the services it provides and the direct links from a task to the services that can be used to achieve the task. The logic model also supports the inference of indirect links (and the corresponding distances) between PEs and tasks.

Finally, in the CPS domain, quite frequently the major steps for achieving a task is known at a high level. Such knowledge can be used in the planning process to direct planning of the full workflow and speed up the process. We express these high level steps as workflow patterns. It is inspired by an existing “macro” in AI planning field [7]. A macro is a set of consecutively performed concrete actions. For example, in the transportation domain, an ‘unload’ action is used to unload some cargo from a truck and a ‘drop’ action is used to put the cargo to the right place. So an ‘unload-drop’ macro is a useful composite service that can be reused in higher level compositions. The workflow pattern we define is more powerful than the “macro” technique. Non-consecutively invoked services can also be modeled in workflow patterns. For instance, when transporting cargo from one location to another, three useful services can be identified: load, move, and unload. But these services are not necessarily invoked one after another. For example, the gas of the truck might need refilling after loading but before moving. The workflow pattern, thus, represents the backbone of the flow.

The service composition architecture in the next section is developed based on this specification model. Note that, in our model, we assume full observability of the world. Our
knowledge base is a comprehensive ontology model that contains accurate information about a domain.

A. Upper Ontologies for Entity Specification

Before presenting our model, we first review an existing service model, namely OWL-S [7]. OWL-S is an ontology model built on top of OWL that is aimed at enabling automatic web service discovery, invocation, composition, and monitoring. It consists of three main parts: ServiceProfile specifies what the service provides for prospective clients; ServiceModel provides information regarding how this service should be used at a process level; ServiceGrounding specifies how to interact with it. The OWL-S model has been widely adopted in the Semantic Web Service composition community.

However, this model has some disadvantages when it comes to the cyber-physical application domain. First, each individual (physical) service provider has its own state such as location, battery, fuel level, etc., which is different from software services whose service provider is usually web server and the state is not important. Second, the OWL-S model binds the ServiceModel tightly with ServiceProfile and vice versa. This prevents an intelligent agent from planning at a “higher level”, that is, treating services that provide the same functionalities as one class of services. In case of a cyber-physical world, there might be a large number of services that provide the same function. It would be helpful if we can ignore the lower level differences at an early decision making phase. Taking all these disadvantages into consideration, we present a new upper ontology model, which is shown in Figure 2. We have a concept of Entity at the very beginning which represents any existing object inside the environment. The state of the entity should be specified using instances of EntityProperty. Two subclasses of Entity are Phenomenon and PhysicalEntity. Phenomenon is used to describe an observable fact or event that exists in the environment. PhysicalEntity (PE) is used to describe the object that can provide services which change the world. The ProvideService relation connects PE to Service concept which is defined in OWL-S.

Here we have added two more properties to Service. Specifically each Service can have a provider and a receiver. The range of provider is PE and the range of receiver is Entity. The rationale behind this is that the purpose of service is to change the state of other entities. Meanwhile, the state of the service provider may also change.

We use the following notations in this paper. First, we want to declare that the Vehicle is a subclass of PhysicalEntity. This can be done in OWL using standard Notation 3 (or N3 notation)[15]:

```
:Vehicle rdfs:subClassOf :PhysicalEntity .
```

As a physical-entity, all vehicles have states of their own. This is expressed using properties.

```
:Vehicle
  :hasLocation :Location ;
  :hasFuelLevel xsd:integer ;
  :hasLoadCapacity xsd:integer ;
  :hasLoad xsd:integer ;
  :hasPassengerCapacity xsd:integer ;
  :hasPassenger xsd:integer .
```

Note that the above definition is an adaptation of the N3 notation. We choose this representation because it is more compact than a regular N3 definition while the meaning is clearly conveyed. For example, all vehicles should have a property called fuel level, and the value of fuel level is of type integer.

After having the PE definitions, we can model physical services. Continuing along the way of OWL-S, services are still modeled as first-class objects here. Our extension to the original model is to add service provider(s) and service receiver(s) here. Also, the state of provider(s) and receiver(s) will be taken into consideration in the precondition/effect specification of services automatically.

The following is an example of load passenger service provided by Jeep which is a subclass of Vehicle:

```
:Jeep
  a owl:Class ,:PhysicalEntity ;
  rdfs:subClassOf :Vehicle .
```

```
:JeepLoadPassenger
  a Service:Service ;
  :hasProvider (jeep rdf:type :Jeep);
  :hasReceiver (human rdf:type :Human);
  :hasInput(location rdf:type :Environment);
  :hasPrecondition {
    jeep :hasLocation location;
    human :hasLocation location;}
  :hasEffect (human :hasLocation jeep)
```

B. Workflow Pattern

There are approaches for synthesizing a workflow using templates [11-13]. The "template" concept that is employed in these approaches are usually action-oriented. For example, in [14], there is a discovery phase which takes the abstract composite workflow, and finds suitable atomic services for each task in the workflow by querying service repositories. However, there is a chance that no such atomic
service exists at all, which will make the whole composite flow infeasible.

To avoid such unnecessary risk, we develop a state-oriented template which we call workflow pattern. By state-oriented template we mean that there is no explicit indication of atomic or composite service in the pattern definition. The explicit indication is a sequence of states which should be achieved in a specific order. Our workflow pattern definition is shown in Figure 3. It is similar to the Process definition from OWL-S in that it has Parameter which can be further subcategorized as input and output. It has provider which is of type PE. It has receiver which is of type Entity. The main property is called WorkflowSteps, which for now is modeled as a subclass of List but can be enhanced to contain more complex control constructs. The elements in the WorkflowSteps are of type Expression. Each instance of Expression represents a possible state of the world. Since the list is ordered, these states are ordered intermediate states that will be passed through in order to achieve the goal.

Workflow patterns are abstract and they are not executable because they do not contain any service information directly. They might refer to some instances of entities as a service provider or receiver. But there is no information regarding which service will change what state of the entity. Thus, it acts as a bridge between the actual executable workflow or plan and the whole world state.

For example, the transportation service provided by a certain vehicle could be: First move to a certain location to pick up passengers, then transport them to the destination, and finally drop off the passengers there. Such a pattern can be defined as follows:

```plaintext
:VehicleTransportation
  a :Pattern ;
  :hasProvider vehicle (rdfs:type :Ambulance);
  :hasReceiver human (rdfs:type :Human) ;
  :hasInput srcLoc, destLoc;
  :hasSteps:
    1. vehicle.location = srcLoc
    2. human.location = vehicle
    3. vehicle.location = destLoc
    4. human.location = destLoc.
```

C. Upper Ontologies for Task/Goal Specification

Figure 4 shows the structure of a task. A task contains a name, input, participants, and a set of goals. The name is a string that provides a way for us to refer to the task. The input is the parameters to the specific task, e.g. a destination location of a transferring task. The participants are entities whose status needs to be changed in this specific task. Finally, the goal specifies the desired status of the entity. The actual achievement of a specific task might depend on the current situation and, hence, involves more participants than specified in the definition.

A task can be interpreted by a physical-entity as a planning problem. The interpretation is a clear formalization of the planning task and involves initial state and goal state specification. Different physical-entities can have different interpretations about the same task.

D. A Logic Model

In this section, we present a logic model of service specification. The reason for doing this is that in the previous sections, our ontology model captured relatively comprehensive information regarding PEs and services. For composition purpose, we are more interested in the following two aspects of information than others:

- Which PE provides what services?
- What task can a specific service be used to achieve?

Let \( S, e, P, c \) be the symbols that denote service, PE, any object, and any condition such as the precondition and postcondition of a service. Then, the following statement asserts that PE \( e \) can provide a set of services \( S_1, S_2, ..., S_n \):

\[
e \xrightarrow{\text{provides}} \{S_1, S_2, ..., S_n\}
\]

For each service, the provider, receiver, precondition, and effects can be asserted as follows:

\[
\begin{align*}
S \xrightarrow{\text{hasProvider}} \{e_{p1}, e_{p2}, ..., e_{pk}\} \\
S \xrightarrow{\text{hasReceiver}} \{e_{r1}, e_{r2}, ..., e_{rk}\} \\
S \xrightarrow{\text{hasPrecondition}} \{c_{p1}, c_{p2}, ..., c_{pk}\} \\
S \xrightarrow{\text{hasEffect}} \{c_{e1}, c_{e2}, ..., c_{em}\}
\end{align*}
\]

We treat each condition \( c \) as an RDF [17] triple that contains three components, namely, the subject, the predicate, and the object, denoted as sub(c), pre(c), and obj(c), respectively.

Let \( t \) be the symbol that denotes a task. Then a task (Section II.C) can be specified using the following form:
The service composition architecture is as follows: When a task with an appropriate name, description and goal arises, service definitions are searched to identify relevant service providers. Given the goal requirement and service provider/receiver information, the set of workflow patterns is searched and a possible workflow pattern is extracted. The next step is to instantiate the abstract workflow pattern. Since the workflow pattern does not contain any service information, relevant service provider/receiver and services must be identified. This is done by searching for plan segments that connect with intermediate states, and initial states with the first intermediate state, and the last intermediate state with the goal state. Intermediate states do not divide the whole planning problem into independent segments. The situation can be illustrated using a scenario. Consider a task of transporting something from one location to another using a vehicle. There is an intermediate state that specifies that the gas of the vehicle should be filled up before it sets off. If there are two gas stations A and B available, then there are two paths from the initial state to the intermediate state. The successor path from the intermediate state to the final goal state depends on the choice that leads to the intermediate state.

A. Generating Planning Problem from Ontological Descriptions

AI planning is the technique for finding a sequence of actions to move a system to a desired state given an initial state, a goal description, and a set of possible actions. The generated plan can be used as a workflow. To take advantage of existing planning techniques, we have to find a method to formulate planning problems out of our ontology model. The key elements of a planning problem include:

- World description (objects and predicates)
- Initial state
- Goal description
- Actions

Due to the introduction of PEs, we take the view that each physical-entity (together with appropriate conditions) is the “driving-force” behind services within the CPS. This view enables us to break the problem of generating those necessary planning elements into two smaller ones:

- What entities are involved in accomplishing a specific task?
- How are they involved?

The first question is answered by identifying the set of useful PEs that need to be involved. More precisely, we are tentatively eliminating PEs that seems not so relevant. In answering the second question, a specific plan needs to be generated. Before we develop a technique to address the first question, let us take a closer look at the relationships between different PEs.

1) Relevancy Function

In this section, we define a relevancy function that reveals the causalities. There are three versions of causality relation, that is: causalities between entities, causalities between task and entities, and causalities between conditions.

Two PEs interact with each other via the provision/receipt of a service. For example, consider a jeep providing a loading passenger service to people. Upon receiving the service, the human instance changes its status from “not inside a vehicle” to “inside a vehicle”. Using notations from Section II.D this can be asserted as:

\[ \text{Jeep} \xrightarrow{\text{provides}} \{ \text{LoadPassenger}, \ldots \} \]

\[ \text{LoadPassenger} \xrightarrow{\text{hasReceiver}} \{ \text{Human} \} \]

A short-hand relation for specifying this is as follows:

\[ e_1 \rightarrow \left\{ \{ c_{p1}, c_{p2}, \ldots, c_{pm} \} \big| \{ c_{e1}, c_{e2}, \ldots, c_{em} \} \right\} e_2 \]  

This notation specifies that PE Jeep can affect PE Human via some service. \( c_{p1}, c_{p2}, \ldots, c_{pm} \) and \( c_{e1}, c_{e2}, \ldots, c_{em} \) are preconditions and effects of that service. In this particular case, \( e_1 \) and \( e_2 \) can be \textit{Jeep} and \textit{Human} respectively.

Thus for each PE \( e \), we can identify all PEs that can potentially interact with \( e \) via one invocation of a service. Such PEs are defined as a set:

\[ L^0(e) := \{ e \} \]

\[ L^1(e) := L^0(e) \cup \left\{ e + e' \rightarrow_{-f} \right\} e \]

Similarly, considering multiple invocations of services will help identify additional PEs that can impact a given PE:

\[ L^n(e) := \left\{ \{ e \} | n = 0 \right\} \cup L^{n-1}(e) \cup \left\{ e + e' \rightarrow_{-f} \right\} e \wedge e' \in L^{n-1}(e) \]}

In the following subsections, we present a mechanism to build these elements from ontological descriptions. The output should be in the form of PDDL with typed objects.

As mentioned in Section II.C and Section II.D, a task \( t \) has a set of participants whose state changes are the ultimate goals of the task. Thus, we can discover the entities that can be used to achieve the task, with a configurable vision setting (the superscript \( \mathbb{N} \)).

\[ L^n(t) := \left\{ e | e \xrightarrow{\text{hasParticipant}} (e) \right\} | n = 0 \]

\[ \{ L^n(e) | e \in L^{n-1}(t) \} | n \geq 1 \]

This definition can be interpreted in the following way. If we refuse to look at entities other than those explicitly mentioned in the task definition, then only those entities are considered as relevant. If we look further one more step,
then those entities that can interact with those mentioned in the task definition are considered. We can look even further by considering indirect entities, more indirect entities, etc. The parameter \( n \) in definitions (2) and (3) represents the number of indirections.

The relevancy functions defined in (2) and (3) are coarse-grained in that they detect the causality either between entities or between entity and task. A complementary approach is to look at the causality between conditions which are descriptions of the world.

According to (1), we define the relevancy between conditions as follows (again, it is quantified by a number \( n \)):

\[
L^n(c) := \begin{cases} \exists e_2, s.t. e_1 \rightarrow (c_3, \ldots, c_{n-1}) e_2(n = 0) \\
L^{n-1}(c) \cup \{ L^1(e) | e \in L^{n-1}(t) \}(n \geq 1)
\end{cases}
\] (4)

The base case of (4) specifies that for a condition \( c \), if there is a service that has \( c \) as an effect and \( c' \) as a precondition, then \( c' \) is one of the causes of \( c \). This relation is generalized in the second half of (4).

2) World Description

For a task \( t \), assume that we have identified the relevant PEs, \( L^n(t) \) for \( n \geq 0 \). All instances of those PEs are converted into objects and constants of the generated planning problem.

Also, as mentioned above, the EntityProperty class is designed to represent the state of each entity. Hence, all the instances of EntityProperty class will naturally become predicates. The first parameter of these predicates will always be of type Entity, and the second parameter will be of type Parameter. As a consequence, any subclass of Entity or Parameter is converted to a type definition in PDDL. Again, not every instance of EntityProperty is relevant here. We consider it relevant if only if

- the domain of the property is of a relevant PE type, and
- the range of the property is of a relevant PE type or is of a built-in type such as string or integer.

3) State and Goal Descriptions

In PDDL, a state is described using a conjunction of grounded predicates. In our ontology model, the state is a conjunction of all entity states. An entity state is described using a set of EntityProperty instances.

These instances are already captured in the last step. In this step, the state of all entities of interest need to be converted into a format that can be understood by the planner. Since goal can be considered as a set of desired states, the processing of goal is similar.

4) Action Description

[8] presents an approach for converting OWL-S 1.1 service description into PDDL 2.1 action descriptions. The basic idea is to convert hasPrecondition and hasInput relation to precondition predicate, hasEffect and hasOutput relation to effect predicate. The idea can be reused in our situation. Also, since we have an extended PE-SOA model [4], extra relations must be processed. Specifically, the provideService relation is converted to its inverse relation, hasProvider relation. Then, the range value of hasProvider and hasReceiver are appended to the input argument list of the service.

Using the mapping procedures defined in sections 1) to 4), the JeepLoadPassenger service mentioned in Section II will be translated as follows:

\[
\text{(:action JeepLoadPassenger}
\text{ :parameters (?jeep - Jeep ?location - Environment ?human - Human)}
\text{ :precondition and}
\text{ (hasLocation ?jeep ?location)}
\text{ (hasLocation ?human ?location))}
\text{ :effect (hasLocation ?human ?jeep))}
\]

This is a standard PDDL definition with three parameters, two preconditions, and one effect.

IV. CASE STUDY

In this section, we apply the planning problem generation technique to a fire rescue scenario. Assume that there is a fire in a certain location of a town and that several people are trapped around that area and must be rescued.

Figure 5 Part of PE-ontology and Environment

A. PE-Ontology

In our previous work [16], we used PE-ontology to describe physical entities. The PE-ontology that will be used in this case is described in Figure 5. The main physical-entities are Vehicle, Ambulance, Fire Truck, and Human. They share common properties such as location information. They can have their own properties, such as fuel level, patient information, and level of fire retardant. The inheritance relation is obvious. Note that properties are also inherited by using the subclass mechanism. Therefore, each instance of Ambulance and FireTruck has three entity properties, namely, location, fuel level, and patient for an ambulance and location, fuel level, and fire retardant level for a fire truck.

B. Service and Workflow Pattern

Vehicle as a general PE class offers a service called MoveTo that can change the location property of the vehicle. As the subclass of vehicle, Ambulance and FireTruck class inherit the service. Also, the fire truck class provides firefighting related services such as CreateFireBreak, PutOutFire, and RescueTrappedPeople. Ambulance, on the other hand, provides rescue related services such as LoadPatient, UnloadPatient. A particular location offers certain service. For example, WareHouse
offers the loading of fire retardant to FireTruck. GasStation offers FillGas service to all vehicles. AutoWash has the service of WashCar and it is provided to all vehicles.

Due to space constraint, we give only two formal service definitions here, namely, the MoveTo service and the FillGas service:

**Service: MoveTo (?loc - Location)**

**HasProvider:** ?v - Vehicle

**Precondition:** ?v hasLocation ?src and ?src != ?loc and

?v hasFuelLevel ?f1 and ?f1 > 0 and

?loc hasState NORM.

**Effect:** ?v hasLocation ?loc and

?v hasFuelLevel ?f2 and ?f2 = ?f1 - 1

**Service: FillGas(?v - Vehicle, ?loc - Location, ?f - integer)**

**HasProvider:** ?g - GasStation

**Precondition:** ?v hasLocation ?loc and

?g hasLocation ?loc and

?v hasFuelLevel ?f

All services can be summarized in Figure 6 with its provider.

As for workflow pattern, we use a similar definition as the one mentioned in Section II.B

**C. Service Selection and Composition**

The problem we are dealing with here is to come up with a comprehensive solution in case of a fire outbreak. In real life, once a fire breaks out in certain area, the tasks that need to be performed usually include, but are not confined to, the following: 1. to create firebreaks in certain areas; 2. to putout fire; 3. to rescue injured people; 4. to transport injured people to hospitals.

Assume that our goal here is to rescue trapped people and transfer them to a hospital. The task can be formally specified as follows.

```sparql
:Transportation1
  a :Task ;
  :hasName Transportation;
  :hasParticipant human (rdfs:type :Human) ;
  :hasInput srcLoc destLoc (rdfs:type :Location);
  :hasGoal:
    human.location = destLoc and
    not (destLoc.status = onFire) and
destLoc rdfs:type Hospital.

It means that initially there are certain individuals who are trapped at a location. The task is to bring those people to a hospital that is not on fire.
```

Let \( t \) denote the task. Now let us apply the relevancy function defined in (3) in Section III.A.1.

\[
L^0(t) = \{ \text{Human, Hospital} \}
\]

The result is also denoted in Figure 5 and Figure 6 by gray shadings.

If we continue to generate a plan based on this result, namely \( L^0(t) \), the only possibility would be that the human instance moves to hospital by himself. In case he or she is trapped and potentially injured, it is not feasible.

We can increase the number \( n \) in (3) which denotes the number of indirections. Then we obtain:

\[
L^1(t) = \{ \text{Human, Hospital, Ambulance, FireTruck} \}
\]

Given \( L^1(t) \), a plan that is more feasible can be generated, that is, the FireTruck would rescue the trapped person at first and then he or she is transferred to hospital by an ambulance.

If we further increase the number \( n \) in (3), we obtain:

\[
L^2(t) = \{ \text{Human, Hospital, Ambulance, FireTruck}, \text{GasStation, WareHouse, AutoWash} \}
\]

\( L^2(t) \) is useful when initially the fuel level of certain vehicle is low or the fire retardant of fire truck needs to be refilled.

Continuing with the result of \( L^1(t) \), our workflow template can help us generate a plan. Assume that the template in Section II.B is adopted and, according to the template, the procedure for transferring injured people to a hospital includes three steps: first, the vehicle moves to the location where the injured people are located. Second, load the injured people into the vehicle. Finally, proceed to a hospital.

In each phase, a specific planning problem can be identified. In the first and last phases which are basically transportation oriented, factors that need to be considered include fuel level and road condition. If the gas level is low, the refilling is needed first. If certain location is on fire, it can block the rescue path and a putout fire service is needed in order to proceed forward. In the second phase, the injured people cannot be loaded directly if they are trapped in a fire zone. Hence, the rescue service offered by fire trucks is needed.

Consider the following two individual vehicle instances. If Vehicle1 is chosen to perform the task, it has to fill up some gas as the first step. On the other hand, if Vehicle2 is chosen, then it can start off to the location where people are trapped right away. These two plans are different in execution time.

```sparql
:Vehicle1
  a Vehicle
  :hasEnvironment :FireDepartment ;
  :hasFuelLevel 2 ;
  :hasPassengerCapacity 13 ;
  :hasPassenger 0.

:Vehicle2
  a Vehicle
  :hasEnvironment :ServiceRoad ;
  :hasFuelLevel 40 ;
```

![Figure 6 PEs and Services](image-url)
As mentioned in Section II, the state of entities is mainly described by instances of EntityProperty. It is shown above that different values of the same property can affect the generated plan. Also, there are other factors that can affect the quality of the generated plan. For example, consider the same Vehicle2 mentioned above with exactly the same value for each property. Vehicle2 can start performing the designated task only if it is not in the middle of performing other tasks. If it is, then the ongoing task must be completed first. This will lead to a different "initial state" for the next task that is considered.

V. CONCLUSION

In this paper, we have addressed the problem of modeling physical services as well as the states of service providers in CPS using an ontology model of our previous work of PE-ontology model and the well-known OWL-S model. Based on these extensions, we have developed a technique for generating an "AI planning domain" according to a specific task description. The generated domain can be converted to PDDL and then provided to modern planners to find an appropriate task-achieving solution. The challenge of generating a valid planning domain is to clearly and explicitly identify basic elements of a domain which includes: constants and objects, the set of predicates being used, and the set of available operators (actions).

Compared with previous work of converting OWL-S to PDDL [6], our approach is selective in that not all possible available ontology classes and services are converted, but only the actually relevant ones are chosen. We provide a formal logic model to determine the relevancy of a physical entity with respect to a particular task. Whenever a planning task is generated, we use a workflow pattern (if available) to guide the process of finding a valid plan. With the help of workflow patterns, we solve a set of smaller planning problems instead of a large complex problem. The benefit of doing so is to find a plan efficiently in spite of a large search space with numerous physical entity service providers.

The relevant PE identification technique mentioned in this paper is relatively coarse grained. Finer grained identification technique is desired and this is a future research direction. Also, AI planning is used as the core search technique in this paper, which requires PDDL as the input language. To meet the demand of combining Semantic Web and AI planning technique together, it would be very useful if planning can be carried out based on ontology specification directly.

REFERENCES