Building toward Capability Specifications of Web Services Based on an Environment Ontology

Puwei Wang, Zhi Jin, Member, IEEE, Lin Liu, Member, IEEE, and Guangjun Cai

Abstract—Automated Web service discovery requires Web service capability specifications of a high precision. Semantic-based approaches are inherently more precise than conventional keyword-based approaches. This paper proposes to build capability specifications of Web services based on an environment ontology, the main concepts of which are the environment entities in a particular application domain and their interactions. For each environment entity, there is a tree-like hierarchical state machine modeling the effects that are to be achieved by the Web services on this environment entity. The proposed approach is based on the assumption that the Web service capability specifications, built on the effects of the Web services on the environment entities, are more accessible and observable. Algorithms for constructing the domain environment ontology and the matchmaking between the Web service capability specifications are presented to show how Web service discovery is supported. An example on Travel Service is given to illustrate this proposed approach.

Index Terms—Web services, capability specification, environment ontology, hierarchical state machine.

1 INTRODUCTION

WEB services, as a new effort toward a seamless integration of distributed components on the Web, have gained expanding attention and interest. A series of standards (WSDL, UDDI, SOAP, etc.) have been proposed to support essential activities such as service description, service discovery, and service invocation. In the meantime, the semantic Web service technology [1] aims to describe the Web services in an unambiguous and machine-processable form for enabling the intelligent interoperation of Web services. Ontology, which provides a formal and explicit specification of a shared conceptualization [2], has been expected to enrich Web services with semantics. For example, OWL-S (formerly DAML-S) [3], WSMO [4], and WSDL-S [5] are ontology-based approaches providing a worldwide standard for semantic Web services, which have made successful steps.

Semantic Web services have brought up several research issues [1], including automated Web service discovery, composition, and execution. Among them, automated Web service discovery is key because it is the prerequisite for the automated composition and execution. Web service discovery refers to the process that a service requester identifies candidate services to fulfill the requester’s requirements. In this process, service providers advertise their services using capability specifications, and service requesters submit their queries to find services that can potentially fulfill their needs. Therefore, the central problem to be solved is how to specify the capabilities of Web services, as well as the requested capabilities, semantically.

Earlier efforts on the semantic capability specifications of Web services assumed that the service capabilities were modeled as the service interfaces. In OWL-S, the service capabilities are modeled as the inputs, the outputs, the preconditions, and the results of Web services (henceforth referred to as IOPR). An example scenario in real life could be that a regular traveler, who wants to have a good budget travel, requests that “I need a travel agency service that provides flight ticket sales and hotel room reservations, whose service fees are charged by credit card.” Obviously, the Web service that satisfies this request can be described by the following IOPR schema:

```
Capability
    cap-id BudgetTravel
    input (?creditCardNo ?start ?end ?time ?hotelLocation)
    output (!flightTicketReceipt !hotelroomReceipt)
    precondition (is-valid-creditcard)
    result (is-charged-creditcard)
```

However, more specific user requests may be raised. For example, the traveler may want to put the flight ticket on hold without being charged. He/She has the specific request that “I want this travel agency service to provide flight ticket ordering, that is, I can order a flight ticket, and if there is an emergency, I can still cancel the ticket on hold.” When considering the service capability model as the service interface, this kind of service capability cannot be described

---

*P. Wang and G. Cai are with the Institute of Computing Technology, Chinese Academy of Sciences, No. 6 Kexueyuan South Road Zhongguancun, Beijing 100080, China. E-mail: {wangpw, caiguang}@ict.ac.cn.

Z. Jin is with the Academy of Mathematics and System Science, Chinese Academy of Science, No. 55 Zhongguancun East Road, Beijing 100080, China. E-mail: zhijin@amss.ac.cn.

L. Liu is with the School of Software, Tsinghua University, Beijing 100084, China. E-mail: llinliu@tsinghua.edu.cn.

Manuscript received 10 Apr. 2006; revised 31 Jan. 2007; accepted 18 Oct. 2007; published online 31 Oct. 2007.

For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-0167-0406.

Digital Object Identifier no. 10.1109/TKDE.2007.190719.
with precision, because the intermediate state “flight ticket is pending without being charged” cannot be suitably expressed. As a result, these queries cannot be handled by the current service discovery mechanisms. Hence, a more elaborate capability model is needed.

To make the capability specifications more expressive and machine processable, this paper proposes an approach that specifies the capabilities of Web services based on the effects that are to be achieved by Web services. These effects are modeled as the state transitions of “real-world” resource entities on which Web services can operate, such as the tickets, the credit cards, the hotel rooms, etc. These “real-world” resource entities are the environment entities to be modeled here. They are domain relevant and independent of the existence of any particular Web service.

The environment-based view has its origin in the research area of requirements engineering. In this area, the requirements of a software system act as the meeting point between an inner world and an outer world, and this has been well recognized. Here, the inner world refers to the “machine” (the software’s inner construction) and the outer world refers to the “world” (the environment on which the software will operate). In fact, this viewpoint can be traced back to Parnas’s famous Four-Variable Model [6]. In this model, two environment variables are used, that is, the monitored and the controlled variables, to describe the system behaviors. This model has been successfully used in modeling control system requirements. Furthermore, Jackson, in his foundation papers [7], [8], has pointed out that the semantics of a software system concerns the environment on which the system will operate instead of the system itself. In the Problem Frame approach [9], a context diagram is used to model the requirements of software systems, which includes domains in the “problem world,” and interactions between the domains and systems. The works in [10] also demonstrate that the meaning of the requirements of a desired software system can be depicted by the “to-be” environment of this system and the operative interactions between them.

It is necessary to isolate the capability specification from the internal structure of the Web services. Hence, the capability specification refers to an “outer world” that corresponds to the environment of Web services (“environment,” for short). This paper assumes that the environment contains the domain environment entities that the Web services operate on. The environment is different from the concept of the “context of Web services.” From a Web service perspective, a context defines a set of common data about the current status of a Web service and its collaboration with peer Web services [11]. Although the context exists depending on a Web service, the environment is relevant to an application domain, and it is independent of the existence of any particular Web service. Based on our previous works [12], [13], [14], [15], this paper proposes an environment-based capability specification for automated Web service discovery. Our research is motivated by the following considerations:

The environment exhibits the capabilities of Web services. The meaning of the requirements can be depicted by the operative interactions of the software system with its environment, as well as the causal relationships among these interactions [10]. Hence, the capabilities of Web services can be expressed by the effects of the Web services on the environment. The characteristics of and interconnections with the environment are what we need to know when specifying the capabilities of Web services.

Environment entities can be shared among loosely coupled Web services. Web services are loosely coupled, and they have sharable descriptions to support the service discovery. In earlier efforts [16], service interfaces were modeled to contain sharable descriptions. A service is discovered when the service’s interface description matches the request. In contrast to the service interface description, the environment entities can play the role to obtain a more expressive sharable description. The environment entities are outside the boundary of Web services. Web services provided by different service providers can interact with some shared environment entities, and the expressive capability descriptions are obtained based on these environment entities.

It is very feasible, using an environment-based perspective, to specify the evolving capabilities of open Web services. In the systems theory, an open system is defined as a system that interacts with its environment [17]. The motivation for change in open Web services is caused by the dynamic changing requests. In other words, changes in the Web services’ desired effects on the environment cause the evolvement of service capabilities. To a particular Web service, although its effects on the environment evolve with changing requests, the environment itself is comparatively stable.

The existing semantic Web service infrastructures describe various aspects related to semantic Web services. In these infrastructures, service capabilities are usually described as the service interfaces. The Environment-ontology-based Capability specification Framework for Web Services (ECF4WS) is proposed as a way to obtain more elaborate service capability specification than that found in an interface-based description. It can be used as a part of the existing infrastructures. For example, in WSMO [4], four building blocks are introduced: Goals, Ontologies, Web services, and Mediators. The proposed environment approach includes both the ontology and Web service capability description. The domain environment ontology corresponds to the WSMO ontology that gives the sharable terminology of the application domain. The environment-based capability specification corresponds to the WSMO Web service capability description.

The rest of this paper is structured as follows: Section 2 introduces the structure of the environment ontology and a procedure of its construction. A running example is given for illustration. Section 3 describes the ECF4WS. Based on the ECF4WS, state-machine-based capability specifications of Web services are generated. Moreover, the matchmaking details between the capability specifications are presented to support Web service discovery. Section 4 analyzes related works and compares them with the proposed approach. Finally, Section 5 concludes this paper and discusses future work.
2 ENVIRONMENT ONTOLOGY

The environment of a Web service is assumed to be a set of stateful domain environment entities surrounding the Web service. This paper shows that the capability specification of a Web service can be obtained by modeling the environment. The proposed environment ontology serves as such a basis. To model the effects on the environment, the environment ontology needs the following features: 1) it captures the state transitions of the environment entities, and 2) it satisfies the demands of different abstraction levels.

The current general ontological structure [18] includes the concept declarations and the relations between them. We extend the general ontological structure in the following aspects.

First, we extend the general ontological structure with state machines to include specifying state transitions of environment entities. Second, we use the hierarchical state machine [19] to support the different conceptualization granularities. Each state in a hierarchical state machine may be an atomic state or a superstate that can be further subdivided (refined). Its skeleton makes the hierarchical state machine support different state granularities. To reduce the computational complexity, the hierarchical state machine is organized into a tree-like structure.

2.1 Skeleton of the Environment Ontology

Fig. 1 depicts the skeleton of the environment ontology, in which the gray part consists of tree-like hierarchical state machines (THSMs) and the binary dependencies between the THSMs. This part is the extension to the general ontological structure. The white part contains the environment entities, as well as the relations among them. For each environment entity, it has a THSM. The environment ontology is constructed from the existing general domain ontology under the guidance of the domain experts.

This section gives the definition of the environment ontology and its related concepts.

Definition 1. Environment Ontology is defined as a 6-tuple $\mathcal{E}_{\text{env}} \overset{\text{def}}{=} \{\text{Ent}, \mathcal{R}, \text{rel}, \mathcal{HSM}, \text{hsm}, \} \}$, in which we have the following:

- $\text{Ent}$ is a set of environment entities. An environment entity is described as $(\text{id}, \text{Attrs}, \text{Rans}, \text{value})$, in which id is the name, Attrs $= \{\alpha_1, \ldots, \alpha_n\}$ is a set of attributes, Rans $= \{v_1, \ldots, v_m\}$ is a set of values, and value : Attrs $\rightarrow 2^{\text{Rans}}$ is a function mapping attributes to value ranges.

- $\mathcal{R}$ is a set of relations.

- $\text{rel} : \mathcal{R} \rightarrow \text{Ent} \times \text{Ent}$ is a function that relates environment entities. For $e_1, e_2 \in \text{Ent}$, and $r \in \mathcal{R}$, if $\text{rel}(r) = \langle e_1, e_2 \rangle$, then $r(e_1, e_2)$ or $e_1 \text{rel} e_2$ holds.

- $\mathcal{HSM}$ is a set of THSMs.

- $\text{hsm} : \text{Ent} \rightarrow \mathcal{HSM}$ is a bijective function. For each environment entity $e \in \text{Ent}$, there is one and only one $\text{hsm}_1 \in \mathcal{HSM}$ such that $\text{hsm}(e) = \text{hsm}_1$. It means that $\text{hsm}_1$ is the THSM of $e$.

- $s \uparrow t$ is a dependence, in which $s$ is a state in a THSM $\text{hsm}_1 \in \mathcal{HSM}$ and $t$ is a state transition in another THSM $\text{hsm}_2 \in \mathcal{HSM}$ ($\text{hsm}_1 \neq \text{hsm}_2$). $s \uparrow t$ means that state transition $t$ can be triggered by the output of state $s$. There is a dependence $s \uparrow t$ between $\text{hsm}_1$ and $\text{hsm}_2$.

For each environment entity, its attributes are classified into two groups: 1) characteristic attributes (and their values, which describe the states of this environment entity) and 2) noncharacteristic attributes. In the environment ontology, a THSM of an environment entity is described as the hierarchical state transition diagram of this entity in a particular domain. This THSM is called the domain THSM of this environment entity. Then, the effects of Web services can be modeled, based on the domain THSMs. Before defining the THSMs of environment entities, the definitions of two basic concepts are given.

Definition 2. For an environment entity $e \in \text{Ent}$, $\text{CAattrs}(e) \subseteq \text{Attrs}(e)$ is the set of characteristic attributes of $e$. The set of states of $e$ is defined as

$$\text{States}(e) \overset{\text{def}}{=} \{\langle \alpha_i, v_j \rangle | \alpha_i \in \text{CAattrs}(e), v_j \in \text{value}(\alpha_i)\}.$$  

This definition shows that the state of an environment entity is a pair of characteristic attributes of this environment entity and a certain value within its finite value range. For example, the environment entity ticket has a characteristic attribute titled salecond, and available is a value of salecond. Thus, (salecond, available) is modeled to be a state of ticket.

Now, a basic state machine (BSM) of an environment entity is defined as follows: Let $e \in \text{Ent}$ be an environment entity and $\alpha \in \text{CAattrs}(e)$ be a characteristic attribute of $e$.

Definition 3. A Basic State Machine (BSM) of $e$ is defined as $\mathcal{N} \overset{\text{def}}{=} \{\mathcal{S}, \mathcal{Σ}, \mathcal{T}, f, \lambda_0\}$, in which

$$\mathcal{S} = \{\langle \alpha, v_j \rangle | v_j \in \text{value}(\alpha)\} \subseteq \text{States}(e)$$

is a finite set of states of $e$; $\mathcal{Σ} = \{\mathcal{Σ}^\text{in}, \mathcal{Σ}^\text{out}\}$, where $\mathcal{Σ}^\text{in}$ is the set of inputs to this BSM, and $\mathcal{Σ}^\text{out}$ is the set of outputs from this BSM; $\mathcal{T} \subseteq \mathcal{S} \times \mathcal{Σ}^\text{in} \times \mathcal{S}$ is a set of state transitions; $\mathcal{T} : \mathcal{S} \rightarrow \mathcal{Σ}^\text{out}$ is an output function; and $\lambda_0 \in \mathcal{S}$ is the initial state.

Then, hierarchy is added to the BSM. $\preceq$ is a tree-like partial-order relation with a topmost point [19]. This relation defines the hierarchy on the states within $\text{States}(e)$ ($x \preceq y$ means that state $x$ is a descendant of state $y$ ($x < y$) or state $x$ and state $y$ are equal ($x = y$)). “Tree-like” means that $\preceq$ has the property $\neg(a \preceq b \vee b \preceq a) \Rightarrow \neg 3z : (a \preceq z \wedge c \preceq b)$. If $x$ is a descendant of $y$ ($x < y$) and there is no state $z$ such that $x < z < y$, the state $x$ is a child of state $y$ ($x$ child $y$).
According to the ≤ on States(e), the sub-division from a state s of e to a BSM N' of e can be defined. If s \notin S(N), ∀s' \in S(N) (S(N) denotes the set of states in BSM N) such that s'child s, state s is a super-state of BSM N' (or BSM N has a super-state s), and (s, N') is a sub-division. The initial state λ₀ in sub-BSM N' is called the "default child" of super-state s. Since ≤ is tree-like, the sub-BSM has no more than one super-state.

The definition of a THSM of an environment entity is presented as follows: Let e ∈ Ent be an environment entity, BSM(e) = {N₁, ..., N_n} (n ≥ 1) be the set of BSMs of e, and D be the set of sub-divisions.

**Definition 4.** A Tree-like Hierarchical State Machine (THSM) of e is defined as hsm(e) = {BSM(e), D}, in which we have the following:

- There is only one special BSM in BSM(e) that called the root (denoted by N_root ∈ BSM(e)) of the THSM.
- Other BSMs are partitioned into m > 0 disjoint sets B₁, ..., B_m, where each of them also can constitute a THSM hsm_i = {B_i, D_i}, D_i ⊆ D, i ∈ [1, m]. If N_root in B_i is the root of hsm_i, then there exists s \in S(N_root) such that (s, N_root) ∈ D. hsm_i is also called a sub-THSM of s.

For example, Fig. 2 depicts a THSM of e (that is, hsm(e)), which consists of 10 BSMs and the sub-divisions D (formally, hsm(e) = {N₁, ..., N₁₀, D}). Among these BSMs, N₁ is the root of hsm(e). Other BSMs are partitioned into three disjointed sets: B₁ = {N₂, N₅, N₆}, B₂ = {N₃, N₇, N₈}, and B₃ = {N₄, N₉, N₁₀}. B₁, B₂, and B₃ also constitute the THSMs of e. Let us focus on the THSM hsm₁ (hsm₁ = {B₁, D₁}). Its root is N₂, and ∃s₁ ∈ S(N₁) is such that (s₁, N₂) ∈ D. The other BSMs in B₁ are further partitioned into two disjointed sets: B₁₁ = {N₃} and B₁₂ = {N₆}. Both B₁₁ and B₁₂ also constitute the THSMs of e. For the THSM hsm₁₁ (hsm₁₁ = {B₁₁, D₁₁}), its root is N₅, and ∃s₁₁ ∈ S(N₁) is such that (s₁₁, N₅) ∈ D₁₁ ⊆ D. There are no other BSMs in B₁₁. Therefore, there are no further sub-THSMs in B₁₁. For hsm₁₂ = {B₁₂, D₁₂}, its root is N₆, and ∃s₁₂ ∈ S(N₁) is such that (s₁₂, N₆) ∈ D₁₂ ⊆ D. There are no other BSMs in B₁₂, and there also are no further sub-THSMs in B₁₂. B₂ and B₃ can be further partitioned into sub-THSMs in the same way. Finally, the sub-divisions D are

\[
\{(s₁, N₂), (s₄, N₅), (s₅, N₆), (s₂, N₃), (s₆, N₇), (s₇, N₈), (s₃, N₉), (s₈, N₆), (s₉, N₁₀)\}.
\]

In the environment ontology, a unique environment entity object is the top entity. Other environment entities are sub-types of object. The domain THSM of object is defined as follows:

**Definition 5.** The domain THSM of the top environment entity object is defined as hsm(object) = {N_obj, φ}. N_obj is the only BSM in hsm(object). It is constructed in terms of its characteristic attribute initcond and value range {noninstantiated, instantiated} of this attribute. Fig. 3 shows the domain THSM hsm(object).

In the environment ontology, other environment entities are sub-types of the top environment entity object. Hence, they have the inherited BSM that is constructed in terms of the characteristic attribute initcond and its value range {noninstantiated, instantiated}. Consequently, the domain THSM of each environment entity has a root BSM, which is inherited from the domain THSM hsm(object).

### 2.2 Construction of the Environment Ontology

The environment ontology is derived from the existing general domain ontology under the guidance of domain experts. Concretely, environment entities can be identified from the existing general domain ontology, as are the relations among these environment entities. Therefore, the first step for constructing an environment ontology is to refine the environment entities and the relations among them from the existing general domain ontology. The second step is to construct the domain THSM of each environment entity, in consideration of the component and the inheritance relations. We consider the following relations among environment entities as derived from the general domain ontology for constructing the domain THSMs of the environment entities:

- These relations represent the characteristics of concepts (called "characteristic relation"). For example, Fig. 4 is a general domain ontology Ticket, in which the dark gray ellipse denotes the environment entity, and the dashed line with an arrowhead denotes its characteristic relation. The relations initcond, deliverycond, salecond, and discountcond associated with the concept ticket are characteristic relations. Ticket is an environment entity, and these characteristic relations will be treated as its characteristic attributes. They are fundamental to the construction of the domain THSM of ticket.

- These relations are the components or the inheritance relations between concepts. For example, isPartOf is a component relation from itinerary to
Algorithm 1 Constructing BSMs

Input: $O$ is a general domain ontology, and $c$ is a concept in $O$, which is identified to be an environment entity (denoted as $e \in \text{Ent}$).

Output: set of BSMs of $c$, that is, $\text{BSM}(e)$.

1: $\text{BSM}(e) = \emptyset$ //Initialization
2: $\text{CAttr}(e)$ is the set of characteristic attributes of $e$ that are derived from $O$;
3: for all $\alpha \in \text{CAttr}(e)$ do
4: Get a value range $\text{value}(\alpha)$;
5: Create a set of states $S = \{ (\alpha, v_j) | v_j \in \text{value}(\alpha) \}$;
6: Create a set of state transitions $T \subseteq S \times \Sigma^m \times S$;
7: Create an output function $f$ such that for each state $s \in S$, $f(s) = m_{out}$, $m_{out} \in \Sigma^m$ is an output;
8: Set an initial state $s_0 \in S$;
9: $\text{BSM}(e) = \text{BSM}(e) \cup \{ s, f, s_0 \}$;
10: end for
11: return $\text{BSM}(e)$

After the BSMs of environment entities are constructed, we shift our focus to the component relation and the inheritance relation between environment entities. If there is an inheritance relation from one environment entity $e_1 \in \text{Ent}$ to another environment entity $e_2 \in \text{Ent}$, it means that $e_1$ inherits the characteristic attributes from $e_2$. For example, the environment entity "merchandise" has two characteristic attributes \{initcond, salecond\}. Therefore, the environment entity "ticket", which has an inheritance relation to merchandise, has its own characteristic attributes \{deliverycond\} and the inherited characteristic attributes \{initcond, salecond\}. During the procedure of constructing $\text{hsm}(\text{merchandise})$, the BSMs and the subdivisions in $\text{hsm}(\text{merchandise})$ will be inherited.

Similarly, if there is a component relation from $e_2$ to $e_1$, $e_1$ owns characteristic attributes of $e_2$, and the identifiers of these characteristic attributes are added with the identifier of $e_2$ as a prefix. For example, "itinerary" has a component relation to "ticket", and preparecond is a characteristic attribute of itinerary. Then, "ticket" has the characteristic attribute itinerary:preparecond. The BSMs and the subdivisions in $\text{hsm}(\text{itinerary})$ will also be included in $\text{hsm}(\text{ticket})$.

Algorithm 2 constructs the domain THSM of an environment entity $e \in \text{Ent}$ in terms of the inheritance relation. Let $e_1, \ldots, e_n \in \text{Ent}$ be $n$ environment entities. The domain THSMs of $e_1, \ldots, e_n$ have been constructed, and $e$ has the inheritance relation to $e_1, \ldots, e_n$. The idea is that $e$ inherits characteristic attributes of $e_1, \ldots, e_n$, because $e$ inherits $e_1, \ldots, e_n$. Therefore, the domain THSMs ($\text{hsm}(e_1), \ldots, \text{hsm}(e_n)$) are inherited during the construction of $\text{hsm}(e)$. The operation sematics of inheritance in the algorithm are in order to create inheritance threads from the BSMs of $e$ to its inherited BSMs from $e_1, \ldots, e_n$.

Algorithm 2 Constructing domain THSM in terms of the inheritance relation.

Input: Domain THSMs $\text{hsm}(e_1), \ldots, \text{hsm}(e_n)$ and $e$ has the inheritance relation to $e_1, \ldots, e_n$

Output: Domain THSMs $\text{hsm}(e_1), \ldots, \text{hsm}(e_n)$ and $e$ has the inheritance relation to $e_1, \ldots, e_n$

1: $\text{BSM}_e = \emptyset$;
2: $\mathcal{D}_e = \emptyset$ //Initialization
3: $\text{BSM}_{\text{hsm}} = \text{ConstructingBSMs}(e, O)$ //Creating BSMs of $e$ according to Algorithm 1
4: $\text{BSM}_e = \text{BSM}_{\text{hsm}}$;
5: for all $e' \in \{ e_1, \ldots, e_n \}$ do
6: $\text{hsm}(e') = \{ \text{BSM}_{e'}, \mathcal{D}_{e'} \}$ is the domain THSM of $e'$;
7: $\text{BSM}_e = \text{BSM}_e \cup \text{BSM}_{e'}$;
8: $\mathcal{D}_e = \mathcal{D}_e \cup \mathcal{D}_{e'}$ //Inheriting domain THSMs of $e_1, \ldots, e_n$
9: end for
10: for all $N_i \in \text{BSM}_{\text{hsm}}$ do
11: while $\exists N_j \in \text{BSM}_e - \text{BSM}_{\text{hsm}}$, $N_i$ and $N_j$ are constructed in terms of a same characteristic attribute do
12: $\text{Overload}(N_i, N_j)$;
13: end while
14: if $\exists s$ is a state in $\text{BSM}_e$, $\langle s, N_i \rangle$ is a subdivision then
15: $\mathcal{D}_e = \mathcal{D}_e \cup \{ \langle s, N_i \rangle \}$ //Adding a subdivision
16: end if
17: end for
18: return $\text{hsm}(e)$

Algorithm 3 constructs the domain THSM of an environment entity $e \in \text{Ent}$ in terms of the component relation. Let $e_1, \ldots, e_n \in \text{Ent}$ be $n$ environment entities. The domain THSMs of $e_1, \ldots, e_n$ have been constructed, and $e_1, \ldots, e_n$ have the component relation to $e$. The algorithm is similar to Algorithm 2. The domain THSMs ($\text{hsm}(e_1), \ldots, \text{hsm}(e_n)$) are included during the construction of $\text{hsm}(e)$. However, it does not need to deal with the problem of overloading. During the procedure, the root BSMs that are constructed in terms of the characteristic attribute initcond of $e$ and $e_1, \ldots, e_n$ are combined.

Fig. 4. Fragment of a general domain ontology Ticket.
Algorithm 3 Constructing domain THSM in terms of the component relation.

**Input:** Domain THSMs $hsm(e_1), \ldots, hsm(e_n)$ and $e_1, \ldots, e_n$ have the component relation to $e$

**Output:** $hsm(e) = \{BSM_e, D_e\}$

1. $BSM_e = \emptyset$;
2. $D_e = \emptyset$; //Initialization
3. $BSM_{\text{orm}} = \text{ConstructingBSMs}(e, O)$; //Creating BSMs of $e$ according to Algorithm 1
4. $BSM_e = BSM_{\text{orm}}$;
5. for all $e' \in \{e_1, \ldots, e_n\}$ do
6. $hsm(e') = \{BSM_{e'}, D_{e'}\}$ is the domain THSM of $e'$;
7. $BSM_e = BSM_e \cup BSM_{e'}$;
8. $D_e = D_e \cup D_{e'}$; //Including domain THSMs of $e_1, \ldots, e_n$
9. end for
10. for all $N_i \in BSM_{\text{orm}}$ do
11. while $\exists N_j \in BSM_e, BSM_{\text{orm}} = BSM_{\text{orm}} \cup \{s, N_i\}$ and $N_j$ is constructed in terms of the characteristic attribute
12. $\text{incond}_{\text{characteristic attribute}}$ do
13. $\text{Combine}(N_i, N_j)$;
14. end while
15. if $\exists s$ is a state in $BSM_e, (s, N_i)$ is a subdivision then
16. $D_e = D_e \cup \{(s, N_i)\}$; //Adding a subdivision
17. end if
18. end for
19. return $hsm(e)$

### 2.3 An Example Environment Ontology

Fig. 5 shows the fragment of an environment ontology on budget traveling (BTO) represented using the Object-Role Modeling (ORM) schema [20], [21]. ORM has a nice graphical notation that can be verbalized into pseudonatural language. In this figure, creditcard, visacard, mastercard, cardholder, person, merchandise, ticket, flight-ticket, train-ticket, boat-ticket, itinerary, and hotelroom are the environment entities. There are inheritance relations among them: visacard and mastercard are subtypes of creditcard; ticket and hotelroom are subtypes of merchandise; flight-ticket, train-ticket, and boat-ticket are subtypes of ticket; and cardholder is a subtype of person. There are also component relations among them: itinerary is a component of ticket. There are also other relations among them: creditcard is owned by a cardholder, and creditcard is a payment for merchandise. For each environment entity, its characteristic attributes are to construct its domain THSM. Moreover, each environment entity has its own noncharacteristic attributes.

The rest of this section details the procedure of constructing the domain THSM of environment entity ticket, that is, $hsm(ticket)$, in BTO. Other domain THSMs in BTO can be constructed in the same way. According to Algorithm 3, which constructs domain THSM in terms of the component relation, the precondition of constructing $hsm(ticket)$ is that $hsm(itinerary)$ has been constructed. For itinerary, the construction of its domain THSM under the guidance of domain experts is given as follows:

#### 2.3.1 Construction of $hsm(itinerary)$

- **Step 1:** Identifying the characteristic attributes. Four relations are associated with itinerary in Fig. 4. Among them, initcond and preparecond are characteristic relations, that is, they are characteristic attributes of itinerary.
- **Step 2:** Constructing BSMs. Extract value ranges of the above characteristic attributes, respectively:

  - $\{\text{noninstantiated}, \text{instantiated}\}$ and $\{\text{planned}, \text{designed}, \text{formulated}\}$.

According to Algorithm 1, BSMs $N_1$ and $N_2$ are constructed from these characteristic attributes and their values.

- **Step 3:** Constructing the subdivisions.

  - **Instantiated** is the superstate of $N_2$.

  $\langle\text{instantiated}, N_2\rangle \in D_{hi}$. Consequently, $hsm(itinerary) = \{\langle N_1, N_2 \rangle, D_{hi}\}$ is constructed as shown in Fig. 6.

#### 2.3.2 Construction of $hsm(ticket)$

- **Step 1:** Identifying the characteristic attributes. Seven relations are associated with ticket in Fig. 4. Among them, initcond, salecond, discountcond, and deliverycond are characteristic relations, that is, they are characteristic attributes of ticket. Moreover, itinerary has a component relation to ticket.
- **Step 2:** Constructing BSMs. Extract value ranges of the above characteristic attributes, respectively:
According to Algorithm 1, BSMs $N_3$, $N_4$, $N_5$, and $N_6$ are constructed from these characteristic attributes and their values.

- **Step 3: Including the domain THSM hsm(itinerary).** Because itinerary has a component relation to ticket, according to Algorithm 3, the BSMs $N_1$ and $N_2$ of itinerary and the subdivisions $D_{iti} = \{\text{instantiated}, N_2\}$ are included into hsm(ticket). $N_1$ is combined with $N_3$ because they are constructed in terms of the characteristic attribute initcond.

- **Step 4: Constructing the subdivisions.**
  - formulated is the superstate of $N_4$. $\langle\text{formulated}, N_4\rangle \in D_{tic}$.
  - available is the superstate of $N_5$. $\langle\text{available}, N_5\rangle \in D_{tic}$.
  - sold is the superstate of $N_6$. $\langle\text{sold}, N_6\rangle \in D_{tic}$.

Consequently,

$$hsm(ticket) = \{N_3, N_2, N_4, N_5, N_6\}, D_{tic}$$

is constructed. Fig. 7 shows the result.

After the domain THSMs of environment entities are constructed, we identify the dependencies between these domain THSMs. In the environment ontology BTO, there are three dependencies between hsm(creditcard) and hsm(ticket). Similarly, there are three dependencies between hsm(creditcard) and hsm(hotelroom), as shown in Fig. 7.

### 3 The Environment-Ontology-Based Capability Specification Framework for Web Services

Once the environment ontology is explicitly represented, we can then specify the capability of a Web service using the state transitions of those environment entities surrounding this Web service. For example, the capability of a simple ticket-selling service, through which customers order tickets before they purchase them, can be specified as the state transitions of the environment entity ticket. Concretely, with the effect of the ticket-selling service, hsm(ticket) starts in the

![Fig. 6. The domain THSM hsm(itinerary).](image)

![Fig. 7. The domain THSMs hsm(ticket), hsm(hotelroom), and hsm(creditcard).](image)
state of available, the start state, and ends at sold, the target state, and the state trace traverses some intermediate states, for example, ordered. Thus, the state trace of $hsm(ticket)$ is denoted as available $\rightarrow$ ordered $\rightarrow$ sold.

On the basis of this environment-based view of the capability specifications, we propose the ECF4WS in this section. Within the ECF4WS, the state-machine-based capability specification of a Web service is grounded onto the effects of the Web service on its environment entities, and the capability specification can be generated based on the environment ontology.

### 3.1 Skeleton of the Environment-Ontology-Based Capability Specification Framework for Web Services

The ECF4WS mainly consists of the following elements:

- **Environment ontology.** The environment ontology is the fundamental layer for specifying service capabilities, as shown in Fig. 8. As stated in Section 2, it is a sharable knowledge base, constructed from the existing general domain ontology under the guidance of domain experts, and it is also maintained by them. In Web service discovery, the requesters and the providers may have variations in their environment entities, that is, these environment entities belong to different environment ontologies. Ontology mapping and integration can provide a common layer between them. Currently, various approaches of ontology mapping [22] and integration [23] have been investigated, such as the WSMO Mediators [24]. These ontology mediators allow the management of the variation in the environment ontology between the requesters and the providers. In this paper, we assume that the requesters and the providers have the same environment ontology.

- **Capability profile.** The capability profile is designed to advertise the capabilities of Web services. To define the capability profile of a Web service, we first define the effects that this Web service can impose on its environment entities. Based on the effect-based view mentioned above, the effects on an environment entity can be described as a triplet, which contains a start state, a target state, and a set of intermediate states.

#### Definition 6. Let $ws$ be a Web service, $e$ be one of its environment entities, $s_i, s_t \in States(e)$, and $S_m \subseteq States(e)$. If $ws$ can make the state of $e$ change from $s_i$ to $s_t$ and all the states in $S_m$ are in the state transition trace, $ws$ imposes an effect on $e$, and the effect can be defined as $effect_{ws}(e) \overset{def}{=} e : (s_i, S_m, s_t)$.

The traces from $s_i$ to $s_t$ by passing $S_m$ may consist of 1) the state transitions in a BSM of $e$, 2) the transition from a state to its default child in the THSM of $e$, or 3) the transition from a state to its superstate in the THSM. Thus, the capability profile, in order to advertise the capability of the Web service $ws$, can be described with the following structure:

**Definition 7.** The capability profile of a Web service $ws$ is defined as $cap_{ws} \overset{def}{=} \{ Ent_{ws}, M_{ws}, effect_{ws} \}$, in which we have the following:

- $Ent_{ws} = \{ e_1, \ldots, e_n \}$ is a set of environment entities of $ws$. The noncharacteristic attributes and corresponding values of each environment entity are provided. An environment entity is denoted as $e := id[s_{attr1} = v_1, \ldots, s_{attrn} = v_n]$

where $id$ is the name, $NAtr(e)$ is the set of noncharacteristic attributes, $s_{attri}$ is a noncharacteristic attribute in $NAtr(e)$, and $v_i$ is a value of $s_{attri}$.

- $M_{ws} = \{ M_{ws}(e_1), \ldots, M_{ws}(e_n) \}$. Each $M_{ws}(e_i) (i \in [1, n])$ is partitioned into two subsets, $M^{in}_{ws}(e_i)$ and $M^{out}_{ws}(e_i)$ for containing the inputs that $ws$ needs and the outputs $ws$ produces, respectively.

- $effect_{ws} = \{ effect_{ws}(e_1), \ldots, effect_{ws}(e_n) \}$ is a set of effects (called “effect set”) that $ws$ can impose on $e_1, \ldots, e_n$.

The capability profile is a lightweight profile because the providers of Web services are able to describe, in a simple manner, a set of environment entities and the effects of Web services on these environment entities. A capability specification is generated by assigning the semantics (that is, state transitions) automatically to the capability profile in the terms of the environment ontology.

**Forest-structured Communicating Hierarchical State Machine (FCHM).** FCHM is a capability specification of a Web service, which is obtained first by producing specific THSMs. By traversing the domain THSM of an environment entity, traces from the start state to the target state can be generated by passing a set of intermediate states that are triggered by a series of inputs. These traces constitute a THSM. Normally, the THSM, which is reduced from the domain THSM, is called a “specific THSM.” Each specific THSM corresponds to an effect. The specific THSMs and the dependencies between them constitute an FCHM.

**Definition 8.** An FCHM is defined as $chm \overset{def}{=} \{ K, inter_K \}$, in which

- $K = \{ k_1, \ldots, k_n \}$ is a set of specific THSMs corresponding to an effect set and

- $inter_K$ is a set of dependencies between $K$.

Fig. 8 shows the skeleton of the ECF4WS. In the ECF4WS, a service provider/requester, who describes its own/desired capabilities, follows these steps:

1. it points out its environment entities,
2. it describes the effects on these environment entities,
3. according to these effects, it publishes a capability profile to a broker or anything equivalent,
4. according to the capability profile, it generates an FCHM (semi)automatically, based on the environment ontology, and
5. it obtains the FCHM-based capability specification.

3.2 An Example Capability Profile

Web service “Budget Traveling Agency” (BTA) is used to illustrate the capability profile, which provides a budget traveling arrangement online for customers. BTA is supposed to have the following basic capability for customers: ticket purchases and hotel room reservations. The environment in which BTA operates is described as BTO (shown in Fig. 5). Three environment entities (ticket, hotelroom, and creditcard) in BTO are operated on by BTA, and their domain THSMs $hsm(ticket)$, $hsm(hotelroom)$, and $hsm(creditcard)$ have been depicted in Fig. 7. The capability profile of BTA, as grounded on the ECF4WS, is presented as follows:

Since itinerary is a component of ticket, ticket owns the noncharacteristic attributes of itinerary, and the identifiers of these noncharacteristic attributes are added with the identifier itinerary as a prefix, that is, itinerary:start. This capability profile can be translated into natural language statements as follows: BTA provides a ticket sales and delivery service in China for travelers. Before purchasing tickets, an ordering service is provided. If the ordered tickets are not appropriate, the order can be canceled by travelers. Moreover, BTA provides a hotel room reservation service in China. Similarly, if the ordered rooms are not appropriate, the order can be canceled.

3.3 Capability Specification Generation

The algorithm for generating an FCHM from a service capability profile, based on an environment ontology, is described as follows:

Algorithm 4 FCHM-Generation

Input: Environment Ontology \( \{ \text{Ent, R, rel, HSM, hsm, } \} \)

Capability Profile \( \{ \text{Ent}_u, \text{M}_u, \text{eff}_u \} \), in which, \( \text{Ent}_u = \{ e_1, \ldots, e_n \} \subseteq \text{Ent} \), \( \text{M}_u = \{ \text{M}_u(e_1), \ldots, \text{M}_u(e_n) \} \), \( \text{eff}_u = \{ \text{eff}_u(e_1), \ldots, \text{eff}_u(e_n) \} \).

Output: FCHM \( \text{chm} = \{ K, \text{inter}_K \} \)

1. Get \( \text{hsm}(e_1), \ldots, \text{hsm}(e_n) \) which are domain THSMs of \( \text{Ent}_u = \{ e_1, \ldots, e_n \} \) from HSM;
2. Let \( \text{runhsms} = \{ \text{hsm}(e_1), \ldots, \text{hsm}(e_n) \} \); //For depositing the original THSMs to be processed
3. Let \( \text{readyhsms} = \varnothing \);
4. while \( \text{runhsms}! = \varnothing \) do
5. \( \text{hsm}(e) = \text{GetFrom}(\text{runhsms}) \);
6. \( \text{runhsms} = \text{runhsms} \setminus \{ \text{hsm}(e) \} \);
7. Let \( k(e) = \text{buildSpecificTHSM}(\text{hsm}(e), \text{factors}(e)) \);
   //buildSpecificTHSM is a procedure to build a specific THSM from an original THSM according to the factors. Originally,
   \( \text{factors}(e) = \{ \text{effect}_u(e), \text{M}_u(e) \} \)
8. \( \text{readyhsms} = \text{readyhsms} \cup \{ k(e) \} \);
9. Let \( \text{depIn}(k(e)) \) keep the set of dependencies which can trigger state transitions in the specific THSM \( k(e) \);
10. for all \( s \uparrow t \in \text{depIn}(k(e)) \) do
11. \( \text{Adjust}(\text{factors}(e'), \{ s \uparrow t \}) \); //Output of \( s \) of another environment entity \( e' \in \text{Ent}_u - \{ e \} \) can trigger the state transition \( t \) in \( k(e) \). Adjusting \( \text{factors}(e') \) to assure that state \( s \) of environment entity \( e' \) can be reached
12. if \( \exists k(e') \in \text{readyhsms} \) then
13. \( \text{readyhsms} = \text{readyhsms} \setminus \{ k(e') \} \);
14. \( \text{runhsms} = \text{runhsms} \cup \{ k(e') \} \); //\( k(e') \) needs to be processed again according to \( \text{factors}(e') \).
15. Hence, it is got out from \( \text{readyhsms} \) and put into \( \text{runhsms} \)
16. end while

//When \( \text{runhsms} \) is empty, the THSMs in \( \text{readyhsms} \) are the specific THSMs, and dependencies between these specific THSMs are also created
17. FCHM \( \text{chm} = \{ K, \text{inter}_K \} \) then is builded as:
   \( K = \text{readyhsms} = \{ k(e_1), \ldots, k(e_n) \} \);
   \( \text{inter}_K = \{ \text{depIn}(k(e_1)), \ldots, \text{depIn}(k(e_n)) \} \)
18. return \( \text{chm} \)

In this algorithm, buildSpecificTHSM is executed first to generate the specific THSMs from the domain THSMs in the environment ontology, and the factors are the effects and inputs regardless of the dependencies. Second, the factors are adjusted in terms of the dependencies between the specific THSMs, and then, buildSpecificTHSM is executed again to generate the target THSMs from the original specific THSMs. Finally, the target THSMs and their corresponding dependencies constitute an FCHM.

Fig. 9 is the screenshot showing the FCHM \( \text{chm}_{\text{chmbta}} \) which is generated based on the capability profile of BTA and the environment ontology BTO. The ws_ticket depicts the specific THSM \( k(\text{ticket}) \), which is generated corresponding to the effect that BTA imposes on ticket. Similarly, ws_hotelroom and ws_creditcard depict the specific THSMs \( k(\text{hotelroom}) \) and \( k(\text{creditcard}) \). The asterisk (*) in Fig. 9 denotes a special trigger in terms of the dependencies between THSMs, rather than the inputs, in the capability profile. For example, the trigger \( \text{accountInfo} \) to \( k(\text{ticket}) \) is from \( k(\text{creditcard}) \) in terms of the dependence

\( \text{creditcard-valid} \uparrow (\text{ticket-ordered} \rightarrow \text{ticket-sold}) \).

3.4 Web Service Discovery

The structured effects on the sharable environment entities make the Web service capability descriptions more expressive and understandable to each other. The scenario that a
user wants to have a good budget travel becomes our focus now. Again, the well-defined environment ontology $BTO$ and the capability profile of a Web service $BTA$ are given. A similar service request as that given in our introduction can be formulated as follows: “I want to find a service that provides a flight ticket from Beijing to Shanghai on 9 July 2007 and I want to order a hotel room in Shanghai from 9 July 2007 to 18 July 2007. If the tickets and hotel rooms are not available, I want to be sure that the orders can still be canceled.” Based on the ECF4WS, the request is described as an effect-based goal profile.

<table>
<thead>
<tr>
<th>Goal Profile Request Budget Traveling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Environment Entities</td>
</tr>
</tbody>
</table>

Inputs
- flight-ticket: [orderInfo, orderCancelInfo, reAvailableInfo, accountInfo]
- hotel-room: [orderInfo, orderCancelInfo, reVacancyInfo]

Outputs
- flight-ticket: [soldInfo]
- hotel-room: [orderInfo]

Effects
- flight-ticket: [available, {ordered, cancelled}, soldInfo]
- hotel-room: [vacancy, {cancelled}, ordered]

Discovery is “the act of locating a machine-processable description of a Web service that may have been previously unknown and that meets certain functional criteria” [25].

We use the environment ontology to represent the functional criteria (that is, the goal) provided by the service requestor, as well as the machine-processable capability descriptions of the Web services that are to be provided by the service providers.

In the existing semantic Web service discovery infrastructures, the requestor searches the capability profiles of the candidate services to find which one can meet the requestor’s goal profile [26], [27]. The ECF4WS complements the existing infrastructures by making requests and service capabilities expressed with intermediate steps of state transition possible.

Let

$$\text{Cap}_{\text{req}} = \{\text{Ent}_{\text{req}}^w, \text{Ms}_{\text{req}}, \text{eff}_{\text{req}}\}$$

and

$$\text{Cap}_{\text{ unsustainable}} = \{\text{Ent}_{\text{ unsustainable}}^w, \text{Ms}_{\text{ unsustainable}}, \text{eff}_{\text{ unsustainable}}\}$$

be a goal profile and a capability profile of the candidate service, respectively. The matchmaking process for Web service discovery includes the following two-step process:

**Step 1.** We first do the matchmaking between the sets of environment entities, that is, $\text{Ent}_{\text{req}}^w$ and $\text{Ent}_{\text{ unsustainable}}^w$. For two environment entities

$$e_1 = id_1[s_{\text{attr1}}, \ldots, s_{\text{attrn}}, v_{\text{1n}}]$$

$$(s_{\text{attr1}}, \in N\text{Attrs}(e_1), i \in [1, n])$$

and

$$e_2 = id_2[s_{\text{attr2}}, \ldots, s_{\text{attr2n}}, v_{\text{2n}}]$$

$$(s_{\text{attr2}}, \in N\text{Attrs}(e_2), j \in [1, m])$$

$s_{\text{attr1}} = s_{\text{attr2}} \land v_{\text{1i}} \subseteq v_{\text{2j}}$ means that $v_{\text{1i}}$ and $v_{\text{2j}}$ are values of the same noncharacteristic attribute, and $v_{\text{1i}}$ is a subtype of $v_{\text{2j}}$. When the values of the noncharacteristic attributes of environment entities are not designated here, the values are universal and denoted as $\top$ (for any value $v \subseteq \top$). If $e_1$ and $e_2$ are the same environment entity, $id_1 = id_2$, or if $e_1$ is a subtype of $e_2$, $id_1 \subseteq id_2$. Now, we can introduce three kinds of relations between the two environment entities $e_1$ and $e_2$ as follows:

1. $e_1 \subseteq e_2 \Leftrightarrow (id_1 \subseteq id_2 \land id_1 \equiv id_2) \land \forall s_{\text{attri}}$,

$$(s_{\text{attri}} = s_{\text{attrj}} \land (v_{\text{1i}} \subseteq v_{\text{2j}} \lor v_{\text{1i}} \not\subseteq v_{\text{2j}})).$$

$e_1$ is a subtype of $e_2$, or they are the same environment entity, and their noncharacteristic attributes and values match inclusively.

2. $e_1 = e_2 \Leftrightarrow e_1 \subseteq e_2 \land e_1 \supseteq e_2$.

$e_1$ and $e_2$ are the same environment entity, and their noncharacteristic attributes and values match exactly.

3. $e_1 \cap e_2 \Leftrightarrow (id_1 \cap id_2 \lor id_1 \supseteq id_2 \lor id_1 \equiv id_2)$

$\land \exists s_{\text{attr1}} \in N\text{Attrs}(e_1) \exists s_{\text{attr2}}$,

$$(s_{\text{attr1}} = s_{\text{attr2}} \land (v_{\text{1i}} \subseteq v_{\text{2j}} \lor v_{\text{1i}} \not\subseteq v_{\text{2j}} \lor v_{\text{1i}} \equiv v_{\text{2j}})).$$

$e_1$ is subtype of $e_2$, $e_2$ is subtype of $e_1$, or they are the same environment entity, and they have some equal values of noncharacteristic attributes.
attributes, and corresponding values, is given using the set-based approach in [27]:

- \(\forall e^{req} \in E^{req}_{ws}, \exists e^{ava} \in E^{ava}_{ws} \Rightarrow e^{req} \subseteq e^{ava} \lor e^{req} \cap e^{ava} \neq \phi\),
  - the general environment ontology of the environment entity.
- \(\forall e^{req} \in E^{req}_{ws}, \exists e^{ava} \in E^{ava}_{ws} \Rightarrow e^{req} \subseteq e^{ava},\) there is the Non-Match \((E^{req}_{ws} \cap E^{ava}_{ws} = \phi)\).

The environment entities in the goal profile and the environment entities that the capability profile refers to are irrelevant. In this situation, the matchmaking process terminates.

- \(\forall e^{req} \in E^{req}_{ws}, \exists e^{ava} \in E^{ava}_{ws} \Rightarrow e^{req} \subseteq e^{ava},\) there is the Plug-In Match \((E^{req}_{ws} \subseteq E^{ava}_{ws})\).

The environment entities in the goal profile form a subset of the environment entities’ set that the capability profile refers to.

- \(\forall e^{req} \in E^{req}_{ws}, \exists e^{ava} \in E^{ava}_{ws} \Rightarrow e^{req} \varsubsetneq e^{ava},\) there is the Subsume Match \((E^{req}_{ws} \varsubsetneq E^{ava}_{ws})\).

The environment entities in the goal profile form a superset of the environment entities’ set that the capability profile refers to.

- \(\forall e^{req} \in E^{req}_{ws}, \exists e^{ava} \in E^{ava}_{ws} \Rightarrow e^{req} = e^{ava},\) there is the Equivalence Match \((E^{req}_{ws} = E^{ava}_{ws})\).

The environment entities in the goal profile and the set of environment entities that the capability profile refers to match perfectly.

- \(\exists e^{ava} \in E^{ava}_{ws} \Rightarrow E^{req} \subseteq e^{ava} \lor e^{req} \varsubsetneq e^{ava},\) there is the Intersection Match \((E^{req}_{ws} \cap E^{ava}_{ws} \neq \phi)\).

The environment entities in the goal profile and the set of environment entities that the capability profile refers to have a “nonempty” intersection.

The matchmaking between the goal profile Request BudgetTraveling and the capability profile BudgetTraveling Agency of BTA (“Req” and “Ava” for short) is given. In the environment ontology BTO, we have, as in Fig. 5

flight-ticket \(\subseteq\) ticket
Beijing \(\subseteq\) ChineseCity, Shanghai \(\subseteq\) ChineseCity
July 9 2007 \(\subseteq\) T., July 18 2007 \(\subseteq\) T.

In conclusion, \(\text{ Req} \cap \text{flight-ticket} \subseteq \text{ Ava} \cap \text{ticket}, \) and \(\text{ Req} \cap \text{hotelroom} \oplus \text{Ava} \cap \text{hotelroom} \). Hence, there is the Plug-In Match between \(\text{ Req} \cap \text{Ent}_{ws}^{req} \) and \(\text{ Ava} \cap \text{Ent}_{ws}^{ava}\) as

\[
\text{Req} : \{\text{flight-ticket}, \text{hotelroom}\} \subseteq \text{Ava} : \{\text{ticket}, \text{hotelroom}, \text{creditcard}\}.
\]

It means that Req and Ava are related.

Step 2. If there is a relevancy (no Non-Match) between \(\text{Ent}_{ws}^{req}\) and \(\text{Ent}_{ws}^{ava}\), they have some common or relevant environment entities. This step is to perform a matchmaking of their effects on the common or relevant environment entities. Because the FCHM is generated according to an effect set grounded on the environment ontology (see Algorithm 4), the matchmaking between the effects is regarded as the matchmaking between the FCHMs that are generated from the effects.

![Fig. 10. Screenshot of the FCHM of the user’s request.](image)

The relevant environment entities of Req and Ava are \{flight-ticket/ticket, hotelroom\}, in which flight-ticket is a supertype of ticket. Fig. 9 shows Ava’s FCHM, which contains \{k-ticket, k-hotelroom, k-creditcard\}. We only focus on \{k-ticket\} and \{k-hotelroom\}, which also constitute an FCHM for describing Ava’s effects on \{ticket, hotelroom\}. Fig. 10 shows Req’s FCHM, which describes the desired Req’s effects on \{flight-ticket, hotelroom\}.

At this point, the problem is how to match the two FCHMs. We do the matchmaking in terms of the equivalence, the inclusion, or the intersection relations between the FCHMs [28]. The problem of matching the general communicating hierarchical state machines (CHMs) consumes extra exponential space. This is due to the arbitrary nesting of the concurrency and hierarchy constructs [29]. The FCHM is a well-structured subclass of the general CHM, in which the arbitrary nesting of the concurrency and hierarchy constructs is not allowed.

Before describing the matchmaking of the FCHMs, some basic definitions are given [19]. Let \(\text{chm} = \{K, \text{ inter}_K\}\) be an FCHM, \(S(k_i)\) be the set of states in \(k_i \in K\), and \(S(\text{chm})\) be the set of all states in \(\text{chm}\) \((S(k_i) \subseteq S(\text{chm}))\).

**Definition 9.** States \(a, b \in S(\text{chm})\) are simultaneous (a|b) if and only if (iff) \(a \in S(k_i), b \in S(k_j), k_i, k_j \in K, \) and \(k_i \neq k_j\).

The states \(a\) and \(b\) are simultaneous iff they are in different specific THSMs. For example, in the FCHM (Fig. 10), states flight-ticket-sold and hotelroom-ordered are simultaneously pairwise.

A set \(A \in 2^S(\text{chm})\) is said to be consistent iff \(\forall a, b \in A : a|b\). It means that all elements in \(A\) are simultaneously pairwise. The set \{flight-ticket-sold, hotelroom-ordered\} is an example consistent set. A global state of an FCHM is defined as a maximal consistent set of the FCHM.

**Definition 10.** A global state of FCHM chm is defined as a maximal consistent set \(A \in 2^S(\text{chm})\):

\[
\forall x \in S(\text{chm}) - A : \neg \text{consistent}(A \cup \{x\}).
\]

For example, \{flight-ticket-sold, hotelroom-ordered\} is the global state of the FCHM (Fig. 10). If we add an arbitrary state, for example, hotelroom-cancelled, into the set, the set will become inconsistent due to the fact that hotelroom-ordered and hotelroom-cancelled are not simultaneous. After defining the global state, we then can define a next-state relation, which is referred to as the global state transitions under the triggers of the FCHM.

**Definition 11.** The next-state relation of an FCHM is defined as \(F \subseteq G \times 2^G \times \mathbb{G}\), in which \(G\) is the set of global states of the FCHM and \(\Sigma^m\) is the set of inputs to the FCHM.
Matching Degrees between $chm_{req}$ and $chm_{ava}$

<table>
<thead>
<tr>
<th>Degree</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equivalence Match</td>
<td>$chm_{req} \equiv \varphi chm_{ava}$</td>
</tr>
<tr>
<td>Plug-in Match</td>
<td>$chm_{req} \subseteq^\varphi chm_{ava}$</td>
</tr>
<tr>
<td>Subsume Match</td>
<td>$chm_{req} \supseteq^\varphi chm_{ava}$</td>
</tr>
<tr>
<td>Intersection Match</td>
<td>$chm_{req} \cap^\varphi chm_{ava}$</td>
</tr>
<tr>
<td>Non-Match</td>
<td>$chm_{req} \not\equiv \varphi chm_{ava}$</td>
</tr>
</tbody>
</table>

In the FCHM (Fig. 10),

\[ g_1 = \{\text{flight-ticket-ordered, hotelroom-vacancy}\} \]
\[ g_2 = \{\text{flight-ticket-sold, hotelroom-ordered}\} \]

are two global states. $(g_1, \{\text{?accountInfo, ?orderInfo}\}, g_2) \in F$ is a next-state relation, in which flight-ticket-ordered transits to flight-ticket-sold under input ?accountInfo and hotelroom-vacancy transits to hotelroom-ordered under input ?orderInfo simultaneously.

Given two FCHMs $chm_1$ and $chm_2$, and a binary relation $\varphi \subseteq G_1 \times G_2$ ($G_1$ is the set of global states of $chm_1$, and $G_2$ is the set of global states of $chm_2$). Let $F_1$ and $F_2$ be the next-state relations of $chm_1$ and $chm_2$, and $G \subseteq G(g)$ be the set of inputs to a global state $g$. We give the following four conditions:

1. $(\sigma_1, \sigma_2) \in \varphi$, where $\sigma_1$ and $\sigma_2$ are the initial global states of $chm_1$ and $chm_2$, respectively.

2. 
\[
\langle g_1, g_2 \rangle \in \varphi \Rightarrow \forall i \in 2^{\Sigma^*(g)} \exists g'_0(\langle g_1, i, g'_0 \rangle \in F_1 \Rightarrow \exists g'_2(\langle g_2, i, g'_2 \rangle \in F_2 \land \langle g'_1, g'_2 \rangle \in \varphi)
\]

3. 
\[
\langle g_1, g_2 \rangle \in \varphi \Rightarrow \forall i \in 2^{\Sigma^*(g)} \exists g'_0(\langle g_1, i, g'_0 \rangle \in F_1 \land \langle g'_0, g'_2 \rangle \in \varphi)
\]

4. 
\[
\langle g_1, g_2 \rangle \in \varphi \Rightarrow \exists i \in 2^{\Sigma^*(g)} \exists g'_0(\langle g_1, i, g'_0 \rangle \in F_1 \land \exists g'_2(\langle g_2, i, g'_2 \rangle \in F_2 \land \langle g'_1, g'_2 \rangle \in \varphi)
\]

If $\varphi$ satisfies conditions 1 and 2 and does not satisfy condition 3, then $\varphi$ is an inclusion relation, and we say that $chm_1$ is included by $chm_2$ under $\varphi$, $chm_1 \subseteq^\varphi chm_2$. Symmetrically, if $\varphi$ satisfies conditions 1 and 3 and does not satisfy condition 2, then we say that $chm_1$ includes $chm_2$ under $\varphi$, $chm_1 \supseteq^\varphi chm_2$. If $\varphi$ satisfies conditions 1, 2, and 3, then $\varphi$ is an equivalence relation; $chm_1$ and $chm_2$ are equal under $\varphi$, $chm_1 =^\varphi chm_2$. If $\varphi$ satisfies condition 4 and does not satisfy conditions 2 and 3, then $\varphi$ is a transitive relation, and we say that $chm_1$ intersects with $chm_2$ under $\varphi$, $chm_1 \cap^\varphi chm_2$. If $\varphi$ does not satisfy any conditions above, then we say that $chm_1$ is irrelevant to $chm_2$ under $\varphi$, $chm_1 \not\equiv^\varphi chm_2$. In summary, the matching degrees between two FCHMs $chm_{req}$ and $chm_{ava}$ are listed in Table 1.

For the matchmaking between the goal profile $Req$ and the capability profile $Ava$, in the first step, there is the Plug-In Match between $Req:Ent_{us}$ and $Ava:Ent_{us}$. $Req$ and $Ava$ have the related environment entities \{flight-ticket/ticket, hotelroom\}. In the second step, there is the Plug-In Match between the FCHMs generated from $Req$’s and $Ava$’s effects on \{flight-ticket/ticket, hotelroom\}. Finally, there is the Plug-In Match between the goal profile $Req$ and the capability profile $Ava$ of the candidate service (BTA). Therefore, it is concluded that the candidate service BTA can satisfy the request.

In this case, the goal profile $Req$ and the capability profile $Ava$ are first inherently related due to their common environment entities. Second, their effects on these common entities are modeled as FCHMs based on the environment ontology. Then, the matchmaking between these FCHMs checks whether $Ava$ can fulfill the requirements of $Req$ or not. Therefore, the shared environment ontology plays the role of a common knowledge base that supports communication among Web services and facilitates interoperability. Moreover, the state transitions triggered by the service on the sharable environment entities are structured to form FCHMs automatically. It constitutes a Web service capability specification with richer semantics so that queries at different levels of abstraction are supported. At the same time, the capability specifications of Web services emphasize its external manifestation without exposing their realization details.

4 Related Work

Web service discovery is a research topic attracting attention daily. Capability specifications of Web services are necessary to publish or request a service. Earlier efforts are mainly XML-based standards such as the Web Service Description Language (WSDL1.1) [30]. It is built on a Universal Description Discovery Integration (UDDI) [31], which provides a registry of Web services. Then, the keyword-based search is conducted against service profiles in WSDL1.1, which describes the Web services in terms of their physical attributes such as name, address, and service interface. Thus, service capabilities in WSDL1.1 are barely represented semantically.

Ontologically-based Web service descriptions have made considerable progress on describing service capability from multiple perspectives. Major initiatives to be mentioned are OWL-S [3], WSMO [4], and WSDL-S [5]. The OWL-S capability model is based on the service interface, that is, the IOPR schema. However, IOPR-based approaches fall short of logical relationships for the underlying inputs and outputs [32]. The OWL-S process model and LARKS [33] are proposed to overcome such obstacles in order to achieve automated and intelligent service discovery. The works in [34] propose execution-related features that can be combined with OWL-S to provide richer semantic features. WSMO [4], [35] differentiates from OWL-S, by specifying internal and external choreography using abstract state machines. The external choreography also represents the service capabilities to some extent. Also, the WSMO Mediator [24] handles heterogeneity among the requesters and providers.
There are also other lightweight approaches annotating WSDL documents with semantic descriptions, such as WSDL-S [5]. WSDL-S uses the semantic concepts analogous to those in OWL-S. However, similar to OWL-S, WSDL-S does not overcome the obstacles of the IOPR schema mentioned above either. The works in [36] propose to annotate WSDL documents by using pi-calculus to solve these problems.

Behavior-based service description [37], [38] aims to discover Web services by matching their behavior descriptions. It is more expressive than the service-interface-based description. Various behavioral models including extended finite-state automata, process algebraic notations, graph formalism, and logic formalisms are studied. Web services are developed by different teams and are described in different conceptual frameworks without prior agreement. Therefore, the behaviors of Web services are described with semantic annotations, such the works in [39].

Service contextual information has also been modeled with the purposes of coordinating and collaborating Web services in their actual implementation. WS-Resource [40] and WS-Context [41] are proposed to promote interoperability among Web services and their stateful interactions. The works in [42] also propose an agent-based and context-oriented approach that supports the composition of Web services. Generally, these efforts emphasize the information on the actual execution of Web services. Descriptions of service capabilities at a higher-level of abstraction are not considered either.

Our approach is different from the above-mentioned approaches. We follow the environment-based “requirements engineering” to specify the service capabilities. Instead of focusing on the philosophy of the Web services, we give high regard to the effects that are to be achieved by the Web services on their environment entities. The Web services’ capabilities are expressed by the services’ effects on the environment entities. Thus, we state that all the capabilities are based on the environment entities, whose characteristics and interconnections are observable and applicable during service discovery and composition.

Existing generic modeling frameworks such as the ORM mechanism [20], [21], can be used to represent the proposed environment ontology. It has a nice graphical notation, and it can be verbalized into pseudonatural language. It is also useful in describing service workflows. We consider the two lines of work complementary to each other.

Finally, Table 2 shows the resulting comparisons of various approaches on the capability specification for Web service discovery. We follow the ontologically-based approaches. The state-based mechanism, which is more expressive than the IOPR schema, is adopted in our capability model. Therefore, the set-based modeling [27] and matching of state machines [28] are combined to realize the automated Web service discovery.

5 CONCLUSION

This paper proposes an approach for specifying Web service capability based on an environment ontology. The main standpoint is that the capabilities of a Web service can be grounded on the effects of the Web service’s environment entities. We propose to build the environment ontology to supply the sharable understanding of those possible effects. The structure of the environment ontology has been presented. It is an extension of the general domain ontology, which associates THSM with each environment entity. The algorithm for constructing this kind of environment ontology from a general domain ontology has also been described so that with the guidance of the domain experts, the environment ontology is constructed. After that, when a Web service provider publishes a Web service or a Web service requester needs a Web service, they can just provide the effects that the Web service can impose on its environment entities. The algorithm has also been developed to transfer the effect-based profile into a capability specification (semi)automatically.

<table>
<thead>
<tr>
<th>Approaches</th>
<th>Model</th>
<th>Ontology</th>
<th>Web Service Discovery</th>
</tr>
</thead>
<tbody>
<tr>
<td>WSDL [30]</td>
<td>message, ports, binding</td>
<td>/</td>
<td>keyword-based service interface matching</td>
</tr>
<tr>
<td>WSDL-S [5]</td>
<td>WSDL with semantic annotations</td>
<td>ontology</td>
<td>ontology-based service interface matching</td>
</tr>
<tr>
<td>WSMO [4]</td>
<td>Capability; service interface; Choreography; service’s abstract state machine</td>
<td>ontology, first order logic</td>
<td>WSMO Capability: ontology-based service interface matching; WSMO Choreography: matching of service’s abstract state machine</td>
</tr>
<tr>
<td>Automata with ontology [37]</td>
<td>service’s automata</td>
<td>/</td>
<td>matching of service’s automata</td>
</tr>
<tr>
<td>Automata with ontology [39]</td>
<td>service’s automata with semantic annotations</td>
<td>/</td>
<td>ontology-based matching of service’s automata</td>
</tr>
<tr>
<td>Process Algebra [38]</td>
<td>service’s process algebra</td>
<td>/</td>
<td>bismutation of service’s process algebra</td>
</tr>
<tr>
<td>WS-Resource [40]</td>
<td>service implementation for operating stateful resources</td>
<td>/</td>
<td></td>
</tr>
<tr>
<td>WS-Context [41]</td>
<td>context-based service implement-</td>
<td>/</td>
<td></td>
</tr>
<tr>
<td>environment-based approach</td>
<td>environment-based capability speci-</td>
<td>environment ontology</td>
<td>ontology-based matching of state-based mechanisms. These state-based mechanisms are generated from the service’s effects on environment</td>
</tr>
</tbody>
</table>
The main contributions of this paper include:

- an effect-based capability profile that makes the Web service specification more accurate without exposing the Web service’s realization details and
- the structured effects of the sharable environment entities modeled by FCHMs, which make the Web service specifications more understandable with each other and make the Web service capability expressive.

Furthermore, the sharable environment ontology allows the interoperability and the communication between Web services to occur so that the automatic Web service matchmaking of different degrees can be realized. The case study in Section 3.4 has disclosed this kind of potential.

The ECF4WS is an ideal complementary with other available efforts on the semantic Web service description. In the future, more “real-world” cases will be studied to examine the validity, advantages, and limitations of the environment-based approach. We also are developing a more efficient service discovery support system, as well as a service composition mechanism that is based on the service environment’s effects.

ACKNOWLEDGMENTS

The authors wish to thank the anonymous reviewers and the associate editor for their constructive comments and suggestions. This work was supported by the National Natural Science Foundation of China under Grant 60625204, the Key Project of National Natural Science Foundation of China under Grant 60736015, the National 973 Fundamental Research and Development Program of China under Grant 2002CB312004, the National 863 High-tech Project of China under Grant 2006AA01Z155.

REFERENCES


Puwei Wang received the MS degree in computer science from Wuhan University, China, in 2004. He is currently a PhD candidate in the Institute of Computing Technology, Chinese Academy of Sciences. His research interests include Web services, the semantic Web, and knowledge-based software engineering methodologies.

Zhi Jin received the MS degree in computer science and the PhD degree from the Changsha Institute of Technology, China, in 1987 and 1992, respectively. She is now a professor of computer science in the Academy of Mathematics and System Science, Chinese Academy of Science. Her current research is on software requirements engineering and knowledge engineering. She is the coauthor of a book published by Kluwer Academic Publishers and has published more than 90 referred papers in the area of requirements engineering and knowledge-based software engineering. She is a member of the IEEE.

Lin Liu is an associate professor in the School of Software, Tsinghua University. Her research interests are in the areas of software requirements engineering, information systems, and knowledge engineering. Her past and current research are centered on agent-oriented requirements engineering methodologies. She has published 20 papers, including journal papers with the Information Systems Journal and the Requirements Engineering Journal and conference papers in the proceedings of RE, CAiSE, ER, etc. She has also participated in more than 10 research projects and is currently leading the investigation of two research projects in the area of requirements engineering. She is a member of the IEEE.

Guangjun Cai received the MS degree in computer science from the Xinjiang Technical Institute of Physics and Chemistry, Chinese Academy of Sciences, in 2006. He is currently a PhD candidate in the Institute of Computing Technology, Chinese Academy of Sciences. His research interests include semantic Web services.

For more information on this or any other computing topic, please visit our Digital Library at www.computer.org/publications/dlib.