https://doi.org/10.20965/jaciii.2010.p0540

Paper:

Using Planning and Case-Based Reasoning for Service Composition

Chiung-Hon Leon Lee*, Alan Liu, and Huan-Hsian Huang****

*Department of Computer Science & Information Engineering, Nanhua University, Taiwan
**Department of Electrical Engineering, National Chung-Cheng University, Taiwan
E-mail: chlee@mail.nhu.edu.tw, aliu@ee.ccu.edu.tw
[Received October 14, 2009; accepted May 19, 2010]

**Planning commonly applied to automating Web Service composition involves two problems –
(i) overlooked user needs combined with services provided by the systems themselves and outside services providing a much more flexible service model.
(ii) "Speeding up" and "facilitating" services by not recording information about service providers already having served users and about planning already processed. We propose merging internal and external services to meet user needs. Internal services include system functions designed to meet user needs. External services mean Web services provided by outside service providers. We plan to combine both types of services to create planning to meet user needs. We apply case-based reasoning to store planning and related information in a case base to make planning much faster when users have similar needs.**

Keywords: service composition, planning, case based reasoning

1. Introduction

Computer systems have long proposed providing users with more varied and richer services in ways as easy as making requests from a human assistant. Service-oriented computing [1, 2] is emerging as a new and promising computer paradigm. The Web's loosely coupled reusability make it as a good choice for enhancing computer service capabilities through technologies such as SOAP, WSDL, and UDDI, for example. To fully meet business application requirements, however, current technologies still must overcome security, composition, and semantic problems [1].

One attractive Web service is composition in which simple services are found, selected, and reorganized into value-added composite services to provide users with more convenient services and solve more complex problems [3–5].

Semantic Web services [6] solve Web service problems semantically and address Web services descriptions as a whole [7]. Semantic markup languages such as OWL-S and its predecessor DAML-S [8] describe Web service capabilities and contents in a computer-interpretable language and improve service discovery, invocation, composition, monitoring, and recovery quality. An agent is a software entity having human properties such as autonomy, reasoning, learning, and knowledge-level communication [9]. To facilitate service access, agents are widely used in Web service research [1, 4, 10], enabling users to discover, interact, and compose Web services to meet user goals and intentions.

Despite the many techniques and standards proposed to solve service provision problems, a large gap remains between human users and service providers, especially for users wanting systems to serve them automatically with minimal user interaction. Assuming, for example, that a user wants to buy a book, the system must understand the user's intent regardless of whether the user has detailed information on the book. The system must, for example, find a service helping the user get details such as the ISBN, correct title, publisher, and provider. The system must also display book information to the user, interact with the user to select the book, and provide the correct order procedure. If more than one choice exists for the same book, the system sorts choices by price, for example, to provide an inexpensive one. The book is thus eventually ordered and displayed to the user.

It is naïve to expect to directly use results returned from service providers to meet a user's request, since the system may have to ask the user for more information for suitable Web services, find or select services, or compose services and process information from them, again displaying results to the user. How to integrate Web services and system capabilities and elicit user information becomes an issue in enabling intelligent Web services. How to systematically design a service-oriented system that understands the user's service request and delivers appropriate services remain open research issues in the service-oriented research domain.

In the simple bookstore service example, we show that the user's request cannot be satisfied directly if the system determines only the book in question and orders it directly. The system must interact with the user to get more book information and process book lists from the service provider. This has motivated some researchers to propose a goal-driven approach modeling service requests from users and integrating Web services, system func-

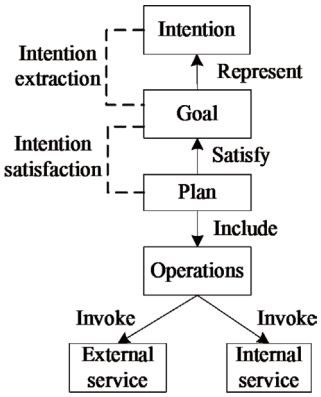

Fig. 1. Intention extraction and satisfaction.

tions, and user information to meet the request [11, 12]. How to interpret service request intent from user input is called user intention extraction and how to integrate internal and external services distributed on the Internet to meet user requests is called intention satisfaction.

We propose an approach for constructing a goal model and extracting intent from service requests [12], focusing on intention satisfaction and merging internal and external services to meet user needs. Internal service means system functions designed to meet user needs. External service means services provided by external service providers. We discuss planning [9] to combine both types of services to create planning made by a series of operations to meet user needs. We apply Case-Based Reasoning (CBR) [13] to store planning and related information in a case base to create planning much faster when users have similar needs.

The operating concepts of our proposal are shown in **Fig. 1**. Intention I is a vector consisting of a set of terms extracted from a user service request string, $I = \{T_1, T_2, \ldots, T_n\}$, in which T is terms in the service request string and n is the number of terms. Goal G abstractly describes the capability the system provides. Intention extraction problem IE is described as $IE : I \rightarrow G$, meaning that given intention I, after process IE, we map I to an abstract description of system capability G.

Plan P consists of a series of system operations. System operation OP describes a single or composite system function. Operations are used to invoke internal or external services. An internal service is a system function used to serve the user or combined with external services to generate more complex services. An external service is provided by external objects – a Web service provider or other software agents. Intention satisfaction problem IS is $IS : G \rightarrow P$, meaning that given goal G, after process IS, the system derives plan P for achieving G as follows:

Here we focus on the intention satisfaction problem, merging internal and external services to meet user needs. Internal service means system functions designed to meet user needs. External service is provided by external service providers. We apply planning to combine both services to create planning involving a series of operations to meet user needs. We apply CBR to store planning and related information in a case base to create planning much faster when users have similar needs.

This paper is organized as follows: Section 2 reviews background work. Section 3 introduces our approach to service. Sections 4 and 5 develop our proposals. Section 6 presents conclusions.

2. Related Works

Systems such as MIND [14] and Pistor [15] use planning in Artificial Intelligence (AI). Our work adds the feature of user satisfaction in service using planning and remembering the use of previous service requests, designing the use of planning in the form of Hierarchical Task Network Planning (HTNP) [14] in service composition using CBR in reusing previous requests and information.

Our prototype uses previous work on intention-aware goal model [16] and personal ontology [17] to provide services most suitable to the user. Using CBR, the system remembers how the user was previously satisfied by services to reuse previous experience in future use. HTNP is used as the base in deriving services [14]. OWL-S files provided by service providers are converted to the domain in the tool, SHOP2, and the service request entered by the user is used in planning with domain information. A description file resulting from the service in OWL-S is then produced. We use HTNP [18] for service composition, but the difference between the work above and this is that they provide an algorithm, called Enquirer, in the query manager to obtain information. The advantage is that it produces a reasonable result in initial stages when information is still lacking. An approach in planning, called model checking, is also used [19]. We use MBP Planner [20] to solve nondeterministic and partially observable problems, together with problems associated with extended goals. CBR is used [21] for service composition. We use six different relationships among services, and the service name together with its service description is used to retrieve cases to provide solutions.

Table 1 summarizes background work with features and criteria, such as main methods, capability in handling unexpected situations, capability in achieving goals, recording what service providers had used, and composition among internal and external services. All systems achieve their goals from known providers, but only some can handle unexpected situations. Since none learns providers they have previously used, all service requests are processed from scratch. No systems can compose internal and external services together. If a system has internal services, it can use such services in place of external services, saving resources in search and transmission. Such internal services may replace other external services in case of failure.

Our research uses HTN planning with CBR to provide user services. For first-time users, the system uses HTN planning to compose a service by exploring external and internal services for possible combinations. The system simultaneously learns the use through CBR. If the user is-

Table 1. Comparison.

Work	Method	Unexpected situation	Goal achievement	Learning	External + internal services
[14]	HTN Planning	No	Yes	No	No
[15]	Planning as Model Check	Yes	Yes	No	No
[18]	HTN Planning	Yes	Yes	No	No
[21]	CBR	No	Yes	No	No

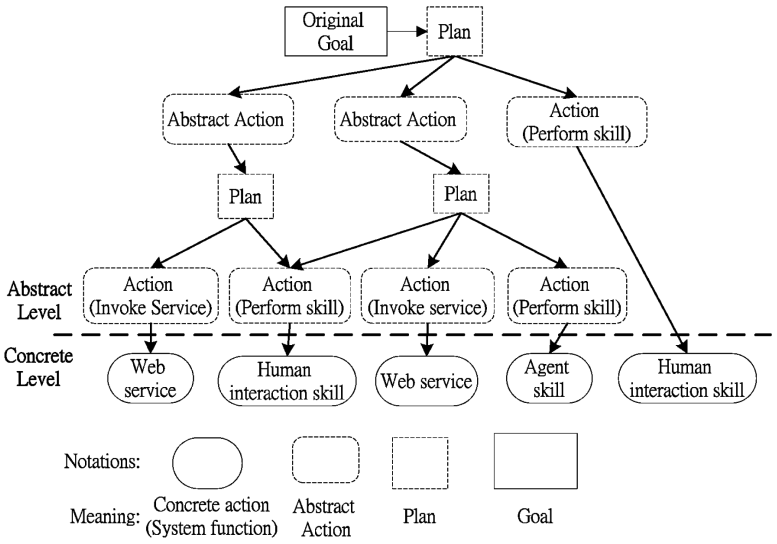

Fig. 2. Goal-plan hierarchy.

sues a similar request in the future, CBR finds a composite service from previous experience.

3. Combinations

One of our features makes it possible to combine internal services originated at the user and with external services from other service providers. Another aim is to record use of a particular user to deliver a composite service without reorganizing it from scratch if the service request has been sent previously. Two AI techniques used in our system – planning and CBR – are discussed below, combining planning and CBR as an extension [22]. If the goal model cannot be solved by CBR based on our case base, the planner takes over and finds a composite service for the user. If the solution can be found using CBR, then the user has the choice of reusing the solution or recomputing a new solution.

3.1. A Goal-Plan Hierarchy Concept

A plan consists of a series of actions that should be provided for achieving the goal. Using goal information and the system execution status, the system retrieves and adapts a series of actions from a plan library to achieve the user goal. If successful or failed results are produced by the execution of a plan, these messages may be passed to the system. If a "soft" original goal of the user fails, the partial result completed by subplans is displayed to the user.

The goal-plan structure shown in **Fig. 2** is divided into abstract and concrete levels. The abstract level contains the relationship and descriptions of plans and actions. The concrete level includes different concrete functions provides by the system.

A plan consists of several actions. An action may generate a subplan or call a concrete system function to complete requirements for action. We hold that the action to generate a subplan is an abstract action, and the action to call a predefined function in the concrete level is a concrete action. If a plan contains one or more abstract actions, then it is an abstract plan; otherwise, it is a concrete plan. Descriptions of action decomposition are stored in the plan library.

The concrete level contains three types of functions provided by the system – Web services, human skills, and agent skills. The Web service acquires results from the service provider or service broker, the human skill elicits required information from the user, and the agent skill triggers the agent to perform actions for processing data acquired from the Web service or user. The original goal could be any actor-specific goal predefined in the goal model, and action generation depends on the situation when the system is executing. The advantages of such a hierarchy are reusability and flexibility.

Table 2. Obtaining external services.

Action	Precondition	Status change
Select_Service	Service (S) Select (S)	Haveservicedata (S) ¬ Select (S)
Check_Service_Data	Haveservicedata (S) ¬Have_main_service_data Main_goal (S)	Have_main_service_data
Check_all_Condition	Sort_Condition (S) (C) Filter_Condition (S) (C) Haveservicedata (S) (C)	Check_Ready ¬ Sort_Condition (S) (C) ¬ Filter_Condition (S) (C)
Show_Data	Check_Ready Have_main_service_data	Finish

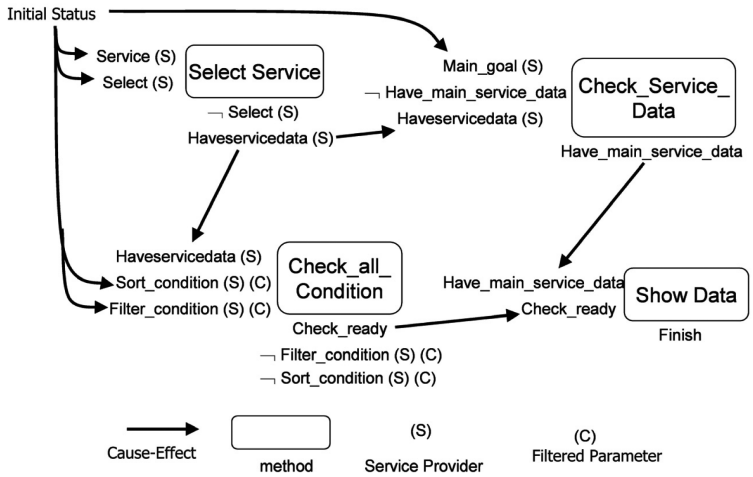

Fig. 3. Relationship among four methods.

3.2. HTN-Planning

Using a planning mechanism requires more information than a single service request from a user. From our previous research in analyzing user intention and ontology is defined, a system transforms the user query into a goal model consisting of the following attributes [16]:

1. Actions – a user's intended action
2. Objects – a user's preferred type of service
3. Constraints – a user's constraint toward a service
4. Parameters of objects – a user's preference toward a service

When using an external or internal service, we may face unexpected situations such as timeouts or execution failure during execution of a plan. To solve such nondeterministic problems, we add a monitoring action to planning. Because we use HTN planning, we insert a monitoring action into some actions with the potential of failure.

We use $P = (T, S)$ as our definition of planning, where P is the plan generated, T the set of tasks, and S the initial plan problem status. The planner uses S and T to determine a plan. In the task network, T, t' is a subtask. A subtask may be a method or an operator, so task decomposition has the following rules:

1. If t' is a method, then t' must be decomposed until t' becomes an operator.
2. An operator is the basic action and the basic action cannot be divided, so decomposition stops when t' is an operator.

t_1 may be viewed as 4 methods – Select_Service, Check_Service_Data, Check_all_Condition, and Show_Data. **Table 2** summarizes preconditions for actions and status changes afterward. The relationship among these methods is shown in **Fig. 3**, in which S is a service provider and C the parameter for filtering information after a service.

Figure 3 shows preconditions needed to be satisfied before a method is executed and status changes a method brings after execution. Note, for example, that to execute Show_data, we must meet two statuses – Check_ready and Have_main_service_data. These two statuses rely on two methods – Check_all_condition and Check_service_data. These two methods instead rely on Select_service and Select_service is triggered when Service(S) and Select(S) are true. The status Haveservicedata(S) means that after the execution of Select_Service the system have data from selected service.

The four methods in **Table 2** are further divided into methods or operators. Taking the method Select_Service as an example, we divide it into primary and secondary services, which in turn are defined as Main_Goal and Second_Gaol, and have different preconditions as shown in **Fig. 4**.

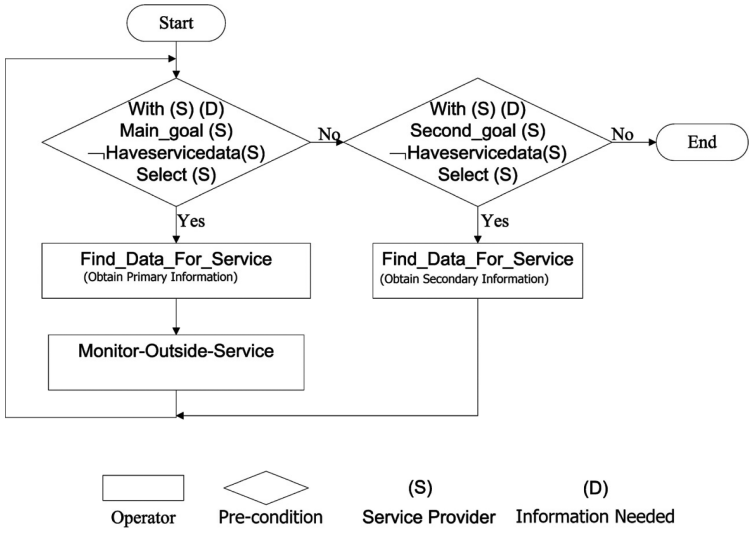

Fig. 4. Select_Service flow.

For Check_all_Condition, the main purpose is to check whether sorting or filtering status still exists in the service. If a service still requires sorting or filtering, Check_all_Condition cleans up the status after sorting or filtering and produces Check_Ready status.

3.3. Process with CBR

We use the three attributes – action, object, and constraints – to define the description of a case. For the solution, this includes locations of OWL-S files by service providers, parameters needed for service providers, and plans.

At case retrieval, we use the goal model provided as features for finding a case. Attributes Action, Object, and Constraints are features used. At case reuse, we check the content of the goal model and the retrieved case. If content is an exact match, we simply copy the solution as a composite service for the user. If a difference remains, an adaptation modifies the solution accordingly.

For the prototype, we have not yet found a satisfactory adaptation, so we simply use substitution for adaptation, finding data on Constraints and Parameters of objects for comparison and apply If-Then rules to modify content. Rules include the following:

A. Rules based on constraints (18)

- **IF** Case_Constrain = 'Before Date' **AND** New Constrains = 'After Date'
  Then Update Case_Constrains_Data
- **IF** Case_Constrain = 'Before Date' **AND** New Constrains = 'About Date'
  Then Update Case_Constrains_Data
- **IF** Case_Constrain = 'After Date' **AND** New Constrains= 'Before Date'
  Then Update Case_Constrains_Data
- ...

B. Rules based on Parameters of objects (8)

- **IF** Case_Date != New_Date **THEN C**ase_Date = New_Date
- **IF** Case_DepartureTime != New_DepartureTime **THEN** Case_DepartureTime = New_DepartureTime
- **IF** Case_ ArrivalTime != New_ ArrivalTime **THEN** Case_ ArrivalTime = New_ ArrivalTime
- ...

When all services in a plan generated by the HTN planner are completed, we apply QoS methods [23] to evaluate the result. If the result is favorable, this successful case is stored in the case base.

4. System Implementation

We use domain-independent planning tool JSHOP2 [24] as our planner. For defining a planning problem and planning domain, we use definitions as follows:
[defproblem problem-name domain-name $(a_1, a_2, \ldots, a_n)T$]; where problem-name is given by the user and domain-name are picked from the planning domain. a_1, a_2 and to a_n are the initial states, and T is the tasks which can meet the initial states.
[defdomain domain-name $(d_1, d_2, \ldots, d_n)$]; where domain-name is given by the user, and d_1, d_2, all the way up to d_n is operations.

JSHOP2 transforms the planning problem and planning domain into Java, and plans are delivered after Java is executed. Planning problems vary with the user. We use OWL-S API [25] in developing our system. With this API, we execute an OWL-S file containing an atomic process in WSDL grounding or use sequence, unordered, and split to control composite process in OWL-S.

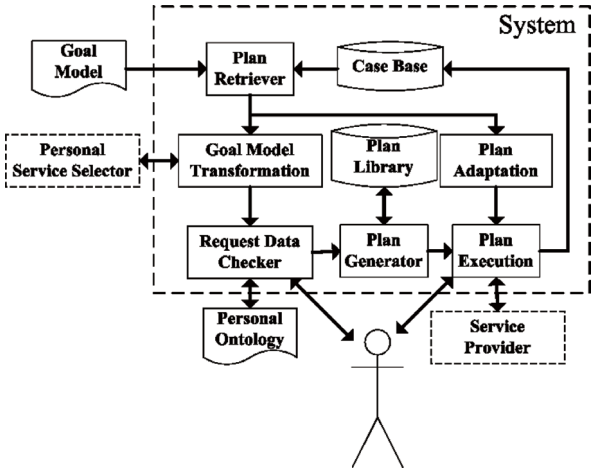

Fig. 5. System architecture.

Figure 5 shows the system architecture consisting of service providers and personal service selector in the outer part with goal models and personal ontology coming through an external interface. In this prototype, plan execution is simply assumed to execute resulting plans.

Our system has 6 important components:

(i) Plan Retriever retrieves a case for a plan if there is a case to be found. It then transfers goal models to the request data checker. If there are several similar cases, the system displays choices for the user.

(ii) Goal Model Transformer uses Action and Object fields in a goal model to understand what type of a provider a user is seeking. If using external services is needed, parameters related to input and out are given to the personal service selector.

(iii) Request Data Checker checks whether information for executing a service is complete. If not, then personal ontology is first checked. If more information is needed, then the user is asked to enter information.

(iv) Plan Generator uses JSHOP2 to generate plans based on the problem domain previously defined.

(v) Plan Adaptation checks the goal model and case retrieved in the case base for differences, then uses a rule to modify solutions.

(vi) Plan Execution verifies how well a plan can be executed. When a plan is executed, an evaluation is made.

The sequence of system execution is shown in **Fig. 6**, with rectangles showing system functions. The dashed rectangle is the outer system. Since there are many function blocks in system architecture, we organize them into 4 objects represented as rounded-rectangle in the following sequential diagram for reducing complexity. When the system receives the goal model from the external system, it executes a plan retrieval to cause a search in a case base. For a match, the goal model is transferred to Data Gather.

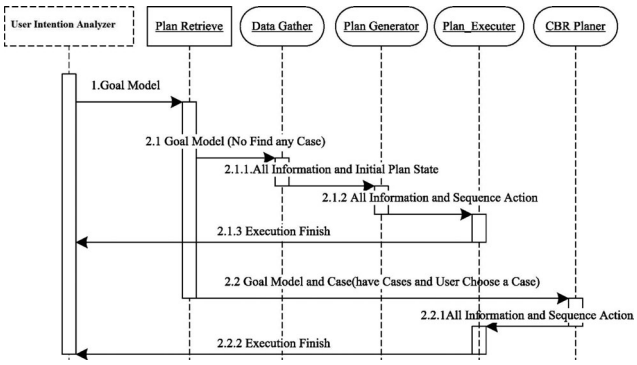

Fig. 6. System sequence diagram.

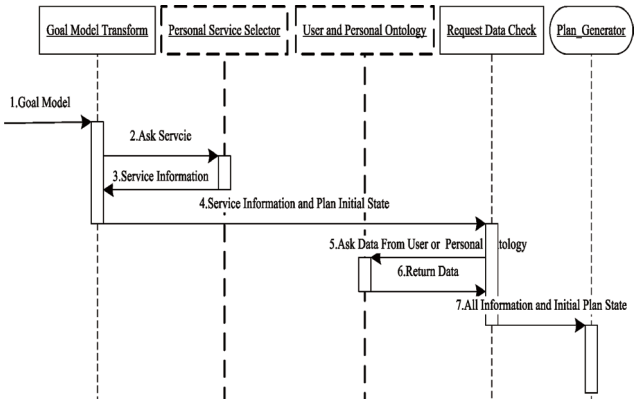

Fig. 7. Data_gather sequence diagram.

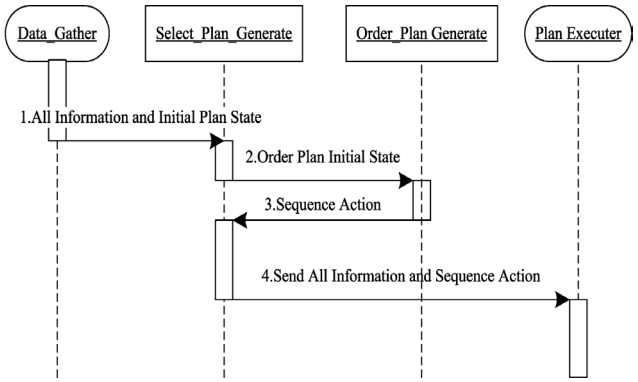

Fig. 8. Planner generator.

After the goal model enters the data gatherer, it uses Goal Model Transform and Request Data Checker as shown in **Fig. 7**. In Goal Model Transform, the goal model is analyzed to determine what types of services the user prefers. If the system does not have enough such services, the system uses Personal Service Selector to pick a suitable service. The sequential diagram of Data Gather delivers service information and planning results to Planner Generator as shown in **Fig. 8**. **Fig. 9** shows the CBR Planner sequence.

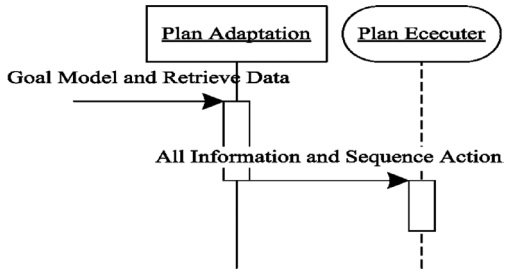

Fig. 9. CBR planner.

5. Example

To demonstrate our proposal, we implemented a service-based travel planning prototype for simulation. We assume that our prototype and all related outer systems use common ontology. The programming language for implementing our prototype is Java. Because there is no traveling Web service in Taiwan, we collected flight, train, and bus schedule information from the Web and implemented related traveling Web services. The scenario used to demonstrate our approach is as follows:

We assume that user Huang requests the system to book a flight from Taipei to Kaohsiung. The extracted user goal model is as follows:

Action: Book,
Object: Flight,
Constrains: Before DepartureTime, and
Parameters of object:
Date: 2006/07/14,
DepartureTime 09:00,
DepartureCity: Taipei,
ArrivalCity: Kaohsiung.

When the system receives the goal model and passes it to Plan Retriever, the system tries to determine whether the system has had a previous similar case. If not, the system uses HTN to plan a new plan. To confirm that the system extracts the correct user intent, the system shows a confirmation dialog for user interaction and the user modifies data or adds information for the request so a plan is generated as shown in **Fig. 8**.

Once generated, the plan is sent to Plan Execution. When the system selects the service, related information is retrieved by OWL-S API. Input and output data of a flight schedule query Web service is shown in **Fig. 9**.

When the system gets data from related Web services, actions do_sort_condition and do_filter_condition are executed, then sorted and filtered data is shown to the user for selection as shown in **Fig. 10**.

Once the user chooses the flight, the system continues to execute ticket ordering. Ticket order information processed by OWL-S API is shown in **Fig. 11**.

When the system got the data from related Web services, the action do_sort_condition and do_filter_condition will be performed. After that, the sorted and filtered data will be shown to the user. A snapshot of this step is shown in **Fig. 12**.

Once the user chosen the flight, the system will continue to execute ticket ordering plan. The ticket order information processed by OWL-S API is shown in **Fig. 13**.

When the flight is booked successfully, the system shows results to the user.

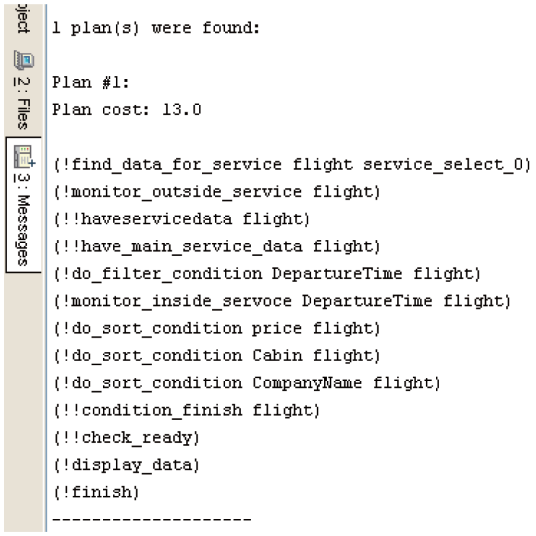

Fig. 10. Flight ticket order plan.

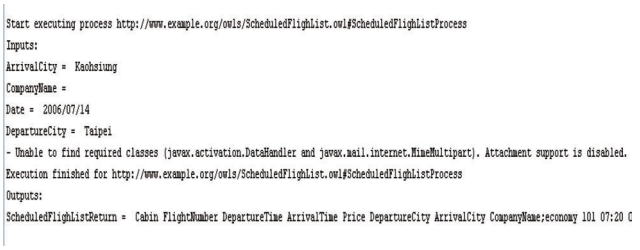

Fig. 11. Input and output data of flight schedule query Web service.

6. Conclusions

We have built a prototype that accepts a service request from a user through intent analysis producing a goal model by extending the service request with keywords representing the intent. We used simulated Web services for transport, including airline tickets and other services. The result was satisfactory using a planner for building a composite service from scratch and reusing experience through CBR. We found that the planner is flexible in adding and deleting services. CBR effectively provides composite service quickly.

Acknowledgements

This work was supported in part by the National Science Council under NSC 98-2221-E-343 -007.

網路服務名稱flight

Data Choose	Data No	Cabin	FlightNumber	DepartureTime	ArrivalTime	Price	DepartureCity	ArrivalCity	CompanyName
◉	1	Business	101	07:20	08:10	3120	Taipei	Kaohsiung	Far_Eastern
○	2	Business	103	08:25	09:15	3120	Taipei	Kaohsiung	Far_Eastern
○	3	Business	253	07:10	08:00	2990	Taipei	Kaohsiung	Nandeain
○	4	Business	257	08:15	09:05	2990	Taipei	Kaohsiung	Nandeain
○	5	Business	0803	07:50	08:40	3200	Taipei	Kaohsiung	UNI_AIR
○	6	Business	0805	08:50	09:40	3200	Taipei	Kaohsiung	UNI_AIR
○	7	Business	561	07:30	08:25	3110	Taipei	Kaohsiung	TrabsAsia
○	8	Business	563	08:35	09:30	3110	Taipei	Kaohsiung	TrabsAsia
○	9	economy	101	07:20	08:10	2120	Taipei	Kaohsiung	Far_Eastern
○	10	economy	103	08:25	09:15	2120	Taipei	Kaohsiung	Far_Eastern
○	11	economy	257	08:15	09:05	1990	Taipei	Kaohsiung	Nandeain
○	12	economy	0803	07:50	08:40	2200	Taipei	Kaohsiung	UNI_AIR
○	13	economy	0805	08:50	09:40	2200	Taipei	Kaohsiung	UNI_AIR
○	14	economy	561	07:30	08:25	2110	Taipei	Kaohsiung	TrabsAsia
○	15	economy	563	08:35	09:30	2110	Taipei	Kaohsiung	TrabsAsia
○	16	FirstClass	101	07:20	08:10	5120	Taipei	Kaohsiung	Far_Eastern
○	17	FirstClass	103	08:25	09:15	5120	Taipei	Kaohsiung	Far_Eastern
○	18	FirstClass	253	07:10	08:00	4990	Taipei	Kaohsiung	Nandeain
○	19	FirstClass	257	08:15	09:05	4990	Taipei	Kaohsiung	Nandeain
○	20	FirstClass	0803	07:50	08:40	5200	Taipei	Kaohsiung	UNI_AIR
○	21	FirstClass	0805	08:50	09:40	5200	Taipei	Kaohsiung	UNI_AIR
○	22	FirstClass	561	07:30	08:25	5110	Taipei	Kaohsiung	TrabsAsia
○	23	FirstClass	563	08:35	09:30	5110	Taipei	Kaohsiung	TrabsAsia

確認完成 重新設定

Fig. 12. Snapshot of system execution.

```
Inputs:
Cabin =  Business
CompanyName =  Far_Eastern
CreditCard =  4521141003655879
Date =  2006/07/14
FlightNumber =  103
TEL =  2523123245
UserName =  Huang
Execution finished for http://www.example.org/owls/OrderFlightService.owl#OrderFlightServiceProcess
Outputs:
OrderFlightServiceReturn =  Ready
```

Fig. 13. Flight ordering information.

References:

[1] M. P. Singh and M. N. Huhns, "Service-Oriented Computing: Semantics, Processes, Agents," John Wiley & Sons. Ltd., New York, 2005.

[2] H. Wang, J. Z. Huang, Y. Qu, and J. Xie, "Web services: problem and future directions," ELSEVIER J. Web Semantics, Vol.1, No.3, pp. 309-320, April 2004.

[3] P. P. W. Chan and M. R. Lyu, "Dynamic Web Service Composition: A New Approach in Building Reliable Web Service," In Proc. of The IEEE Int. Conf. on Advanced Information Networking and Applications, pp. 20-25, 2008.

[4] K. Sycara et al., "Dynamic discovery and coordination of agent-based semantic Web services," IEEE Internet Computing, Vol.8, No. 3, pp. 66-73, May-Jun 2004.

[5] S. Y. Hwang et al., "On Composing a Reliable Composite Web Service: A Study of Dynamic Web Service Selection," In Proc. of The IEEE Int. Conf. on Web Services, pp. 184-191, 2007.

[6] D. Martin et al., "Bringing Semantics to Web Services: The OWL-S Approach," In proc. of the First Int. Workshop on Semantic Web Services and Web Process Composition, 2004.

[7] A. Hibner and K. Zielinski, "Semantic-based Dynamic Service Composition and Adaptation," In proc. of the IEEE Congress on Service, pp. 213-220, 2007.

[8] http://www.daml.org/services/owl-s/

[9] S. Russel and P. Norvig, "Artificial Intelligence: A Mordern Approach 2ed.," Prentice Hall, London, 2003.

[10] C. H. L. Lee and A. Liu "User intention satisfaction for agent-based semantic Web services systems," Proc. of The 12th Asia-Pacific Software Engineering Conference (APSEC'05), Taipei, Taiwan, Dec. 2005.

[11] J. M. G'omez et al., "Godo: Goal oriented discovery for semantic web services," Discovery on the WWW Workshop (SDISCO'06), Beijin, China, 2006.

[12] C. H. L. Lee and A. Liu, "A Goal-Driven Approach for Service Request Modeling," Int. J. of Intelligent Systems, 2009.

[13] I. Watson, "Applying Case-Based Reasoning: Techniques for Enterprise Systems," Morgan Kaufmann Publishers, California, 1997.

[14] E. Sirin, B. Parsia, D. Wu, J. Hendler, and, D. Nau, "HTN Planning for Web Service Composition Using SHOP2," J. of. Web Semantics, Vol.l, No.4, pp. 377-396, 2004.

[15] M. Pistore, F. Barbon, P. Bertoli, D. Shaparau, and P. Traverso, "Planning and monitoring web service composition," Proc. of Int. Conf. on Artificial Intelligence, ethodologies, Systems, and Applications (AIMSA), pp. 106-115, 2004.

[16] C. H. L. Lee and A. Liu, "Model the query intention with goals," Proc. of The First Int. Workshop on Ubiquitous Smart Worlds (USW2005) (Conjunction with AINA2005), Taipei, Taiwan, pp. 535-540, Mar. 2005,

[17] M. N. Huhns and L. M. Stephens, "Personal Ontologies," IEEE Internet Computer, pp. 85-87, 1999.

[18] U. Kuter, E. Sirin, D. Nau, B. Parsia, and J. Hendler, "Information gathering during planning for web service composition," Proc. of 3rd Int. Semantic Web Conf. (ISWC -2004), Hiroshima, Japan, November 2004.
[19] M. Pistore and P. Traverso, "Planning as Model Checking for Extended Goals in Non-deterministic Domains," Proc. of 7th. IJCAI'01. AAAI Press, August 2001.
[20] P. Bertoli, A. Cimatti, M. Pistore, M. Roveri, and P. Traverso, "MBP: a Model Based Planner," Proc. of the IJCAI'01 Workshop on Planning under Uncertainty and Incomplete Information, Seattle, pp. 93-97, August 2001.
[21] B. Limthanmaphon and Y. Zhang, "Web Service Composition with Case-Based Reasoning," Proc. of the Fourteenth Australasian database conf. on Database technologies 2003, pp.201-208, 2003.
[22] H. Munoz-Avila et al., "SiN: Integrating Case-based Reasoning with Task Decomposition," Proc. of Seventeenth Int. Joint Conf. on Artificial Intelligence, 2001.
[23] C. H. L. Lee and A. Liu, "Service Quality Evaluation by Personal Ontology," J. of Information Software and Engineering, Vol.25, No.5, 2009.
[24] O. Ilghami and D. S. Nau, "A general approach to synthesize problem-specific planners," Tech Report CS-TR-4597, UMIACS-TR-20060, University of Maryland, 2003.
[25] E. Sirin et al., OWL-S API, 2004. http://www.mindswap.org/2004/owl-s/api/index.shtml

Name:
Chiung-Hon Leon Lee

Affiliation:
Department of Computer Science & Information Engineering, Nanhua University, Taiwan

Address:
No.55, Sec.1, Nanhua Rd., Zhongkeng, Dalin Township, Chiayi County, Taiwan (R.O.C.)

Brief Biographical History:
2006 Received the Ph.D. degree in Department of Electrical Engineering, National Chung Cheng University, Taiwan
2006-2008 Assistant Professor of Department of Computer Science and Information Engineering, ChungChou Institute of Technology, Taiwan
2008- Assistant Professor of Department of Computer Science and Information Engineering, Nanhua University, Taiwan

Main Works:
• "Pattern Discovery of Fuzzy Time Series for Financial Prediction," IEEE Trans. on Knowledge and Data Engineering, Vol.18, No.5, pp. 613-625, 2006.

Membership in Academic Societies:
• Taiwan Software Engineering Association
• Institute of Electrical and Electronics Engineers, Inc. (IEEE)

Name:
Alan Liu

Affiliation:
Department of Electrical Engineering, National Chung Cheng University, Taiwan

Address:
168 University Road, Minhsiung Township, Chiayi County 62102, Taiwan (R.O.C)

Brief Biographical History:
1994 Received the Ph.D. degree in Electrical Engineering and Computer Science from the University of Illinois at Chicago
1994-2006 Associate Professor of Department of Electrical Engineering, National Chung Cheng University, Taiwan.
2006- Professor of Department of Electrical Engineering, National Chung Cheng University, Taiwan.

Main Works:
• "A Flexible Architecture for Navigation Control of a Mobile Robot," IEEE Trans. on Systems, Men, and Cybernetics - Part A, Vol.37, No.3, pp. 310-318

Membership in Academic Societies:
• Taiwan Software Engineering Association
• Institute of Electrical and Electronics Engineers, Inc. (IEEE)
Association for Computing Machinery (ACM)

Name:
Huan-Hsian Huang

Affiliation:
Department of Electrical Engineering, National Chung Cheng University, Taiwan

Address:
168 University Road, Minhsiung Township, Chiayi County 62102, Taiwan (R.O.C)

Brief Biographical History:
2006 Received the M.S. degree in Department of Electrical Engineering, National Chung Cheng University

Main Works:
• "Service Composition Using Planning and Case-Based Reasoning," Master thesis, Department of Electrical Engineering, National Chung Cheng University

Membership in Academic Societies:
• Taiwan Software Engineering Association