We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

4,800
Open access books available

122,000
International authors and editors

135M
Downloads

Our authors are among the

154
Countries delivered to

TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities

WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com

16

User Interface for Automatic Service Composition

Incheon Paik
University of Aizu
Aizu-Wakamatsu City, Fukushima,
Japan

1. Introduction

Service-oriented computing provides an evolving paradigm for flexible and scalable applications of open systems. Web services are already providing useful application programmers' interfaces (APIs) for open systems on the Internet and, thanks to the semantic Web, are evolving into the rudiments of an automatic development environment for agents. To further develop this environment, automatic service composition (ASC) aims to create new value-added services from existing services, resulting in more capable and novel services for users.

Consider an ASC example. If a user is planning a trip from Aizu (a city in Japan) to San Francisco for an international conference, the user needs to find a transportation sequence from the departure location to the arrival location, a hotel, and forms of entertainment. Then, reservations and payment will be made. Manually, this takes time and effort. ASC can achieve it dynamically and automatically, with minimal human effort and interaction.

ASC requires several stages, namely finding a workflow to fulfill the user's goal, locating service instances for the workflow, selecting services to satisfy nonfunctional properties (NFPs), and executing the selected services. When a user gives a request to the composer, the request has to be understood by the composer, and the composition process started. If the composition completes after receiving the request from the user, only one interaction (inputting the user's goal) exists. However, there are many cases where the user needs to interact further with the composer. This interaction can happen at each stage or just at the start and end of the composition.

The composers (or agents) are computer-based, and are displayed in the form of user interfaces (UIs). The UIs enable human users to communicate with composers. Users supply a request to the composer via the UI that comprises a functional requirement (goal) and nonfunctional requirements such as preferences, constraints, or quality of service (QoS) issues. Usually, the whole composition does not finish without interaction with the user. The user needs to respond to questions from the composer for interim decisions to be used in the composition. The UI is important as the gateway through which the composer receives several requests from the external human user. Therefore, those parts that involve communication between human users and the composer will be defined together with the ASC architecture. The necessity for, and the contents of, the communications between them

Source: User Interfaces, Book edited by: Rita Mátrai,
ISBN 978-953-307-084-1, pp. 270, May 2010, INTECH, Croatia, downloaded from SCIYO.COM

should also be considered in detail. The design of the ontology for data and workflow of the UIs will be explained, and examples of UI implantation will be introduced.

2. ASC

ASC usually involves four stages (Claro et al., 2006), namely 1) planning a workflow of individual service types, 2) locating services from a service registry (i.e., finding service instances), 3) selecting the best candidate services for deployment and execution by using NFPs, and 4) executing the selected services (Fig. 1). If an exception occurs during execution, the planning or selection might have to be repeated to satisfy the composition goal (Shi et al., 2004), (Claro et al., 2006). Each stage can be ranked and overridden for the best service execution result (Agarwal et al., 2008). Some stages can be merged according to the domain, problem, and various composition conditions (Lecue et al., 2007), (Lecue & Delteil, 2007), (Kona & Gupta, 2008), (Oh et al., 2008).

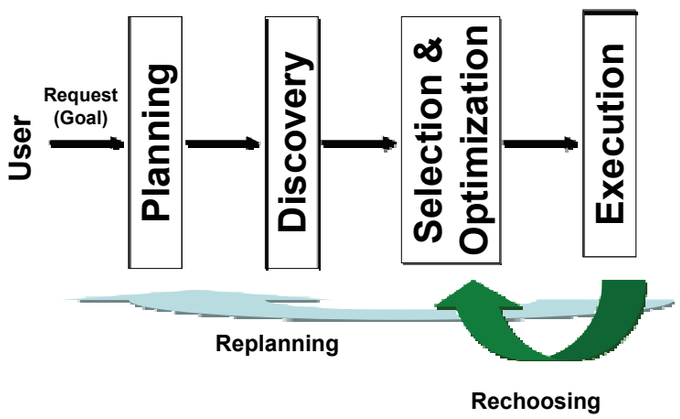


Fig. 1. Stages of ASC

2.1 The four-stage composition architecture

1. Planning Stage

The planning stage generates a finite sequence of Web services (we call it the abstract workflow). The result is an execution order of tasks to fulfill the functionality of the composition goal. The decision process chooses a finite sequence of Web services from a service registry via its own decision approach.

In the planning stage, firstly, the definition of the problem space should be considered. The elements of the composition problem space are a set of Web services with a set of initial input parameters and desired output parameters. The elements can be transformed into a state-space model within which a planner can work. In the state-space model for service composition, the states are usually a collection of parameters when the planner has no additional knowledge or planning information, as described in (Oh et al., 2008), (Kona & Gupta, 2008).

The second issue is a decision about the sequence of services. To automate the finding of a service sequence for an abstract workflow, several planning methods have been used, such as hierarchical task networks (HTNs), finite state machines, constraint programming, and

Petri nets (Narayanan & McIlraith, 2002), (Nau et al., 2004). There have been arguments about which of planning and constraint programming is the better method (Nareyek et al., 2005).

The third issue is the type of abstract workflow. There are several patterns for workflows. They can be described as a simple sequence of tasks or a directed acyclic graph using a Petri net, a workflow language, a services composition framework such as the semantic Web ontology language (OWL-S) or the Web service modeling ontology (WSMO), a business process execution language (BPEL) (Andrews et al., 2003), a Web service choreography interface (Arkin et al., 2002), etc.

2. Discovery Stage

The candidate services for the task created in the planning stage are found in the discovery stage. In general, this stage finds services matching service advertisements and service requests. The discovery process comprises preprocessing of service requests, matchmaking, and postprocessing of discovery results. The most important function, matchmaking, discovers the best candidates for matches between the service advertisements and requests. There are several methods for matchmaking of services, based on keywords, tables, concepts, or ontologies (Paolucci & Sycara, 2002). To achieve better performance, several aspects are considered, including services representation for functionality, context information, definition of joint knowledge between service providers and service requestors, reasoning behind the matching operation, and other methods that decide the uncertainty of the matching such as text mining or statistical methods (Klusch & Sycara, 2006).

3. Selection and Optimization Stage

With the increasing number of services and better performance of services discovery, there may be many candidate services for the tasks identified in the planning stage. The selection and optimization stage selects an optimal set of candidate service instances to fulfill the NFPs. The main issues of this stage are the modeling of NFPs, the matrix of service instances and tasks, and how to solve the optimization problem of selecting a set of service instances to satisfy the objective function with the given NFPs (Hassine et al., 2006). Much work is required in modeling a complete NFP to be applicable to any set of properties.

4. Execution Stage

The selected service instances are executed in this stage. The stage should manage execution monitoring. The monitor aims at maintaining better quality and analysis of execution performance and exception handling. When the monitor finds errors or exceptions, a handling mechanism for them will be executed. An exception manager can handle actions for recovery such as rechoosing and replanning in the architecture. There have been several approaches to execution monitoring on various execution platforms such as the OWL-S virtual machine and the BPEL engine. Checks of functional properties and NFPs during execution, languages for run-time execution monitoring, and combined approaches have been developed (Baresi & Trainotti, 2009). These approaches can deal with the role of the planning or selection stages in the execution stage to some extent. However, service execution monitoring is very complex.

2.2 Additional functional blocks for ASC

In addition to the functional blocks of four-stage ASC, there are other important functional blocks in a complete service composition. These blocks handle NFP transformation, property translation, and workflow orchestration management. The whole ASC architecture is shown in Fig. 2.

1. Property translation

In terms of abstractness and the users' technological perspective, there are two domains, namely the goal (or business) domain and the service domain. While the goal domain refers to the requestors' (human or machine) perspective, the service domain refers to the concrete services at the system level. When a user makes a request to the composer, the composer returns a sequence of services to fulfill the request.

The request usually comes in an abstract form understood by the user in the goal domain. In the specified request given to the composer by the user, a goal consists of the functionality to be achieved, nonfunctionalities, and other related information (WSMO, 2005). There may be other nonfunctionalities that are not related to the requests. There are two types of goal, namely the one understood by requestors only, and the other registered so that it can be understood by the system. The registered goals can help the discovery service to locate the corresponding services in the service domain. All services, including terms for nonfunctionalities in the service domain, can be located from any service registry. The abstract requests must link to the corresponding services, and it is important to refine the generic and abstract goals into concrete goals and to discover services from the abstract goals.

2. NFP transformation (Takada & Paik, 2009)

The functional property of a goal is to be used in the planning stage to fulfill the functionality of the goal, and will be located in the discovery stage. On the other hand, an NFP is generally used at the service selection stage. Users supply abstract NFPs, which cannot be understood in the selection stage.

There are three levels of NFP. The first level includes abstract-level constraints. (Here, we define the constraint as the representative term for an NFP.) These constraints are at a high abstraction level close to natural human concepts. All terms are abstract, and the constraints may not be defined in formal terms. They can be in natural language or may contain several complex meanings in a keyword.

The second level includes intermediate-level constraints. Each comprises a relation, two terms, context information, and an operator. They are generated by extracting abstract relations, terms, and context information from abstract terms (which may include context information) in natural language or compound terms at an abstract level. All the terms are terminal (not compound) and have not yet been bound to concrete terms. The role of the translator is to find the context information, operator, and variables by referring to the ontology.

The third level includes concrete-level constraints. These have relations, terms as binding information, and indexes of abstract workflow. For example, "LessThan(Sum(AllService.Cost))" is transformed to "LessThan(Sum(task[0].Cost,task[1]. Cost, ..., task[n].Cost))". "Cost" refers to the "getCost" method in a real Web service.

While the translator locates the terms in the service domain from abstract terms in the business domain, the transformation obtains the information binding the intermediate terms to the concrete terms that will be used in the selection stage.

3. Workflow orchestration management

There have been many studies of ASC, but they have only considered it as a one-step composition. Where one-step composition does not achieve the goal requested by a user, we must orchestrate further processes dynamically to reach the final goal. This procedure can be recognized as multistep composition via orchestration of the workflows in a nested composition structure.

For example, consider a scenario involving a tour group for a conference (traveling from Aizu to San Francisco. To create the tour group package (the top goal), there must be a composition of three subprocesses, namely (1) trip scheduling, (2) making reservations, and (3) creating the tour group package.
The trip scheduling service can be composed by ASC. Here, the ASC planner generates an abstract workflow (using staged composition and execution) for traffic routes and hotels between Aizu and Los Angeles, and selects an optimal workflow using a metric of preconditions. Then, ASC discovers service candidates, and selects optimal instances of services using QoS and user constraints on the workflow, which are normal steps in an ASC activity (OWL-S, 2003).
However, to achieve the final goal, the selected trip schedule should be passed to the reservation process, and the results of these two processes must be combined to create the tour group. Therefore, the results of subprocesses must be orchestrated by an outer ASC to achieve the final goal. The workflow orchestration manager orchestrates the nested compositions and the whole composition flow.

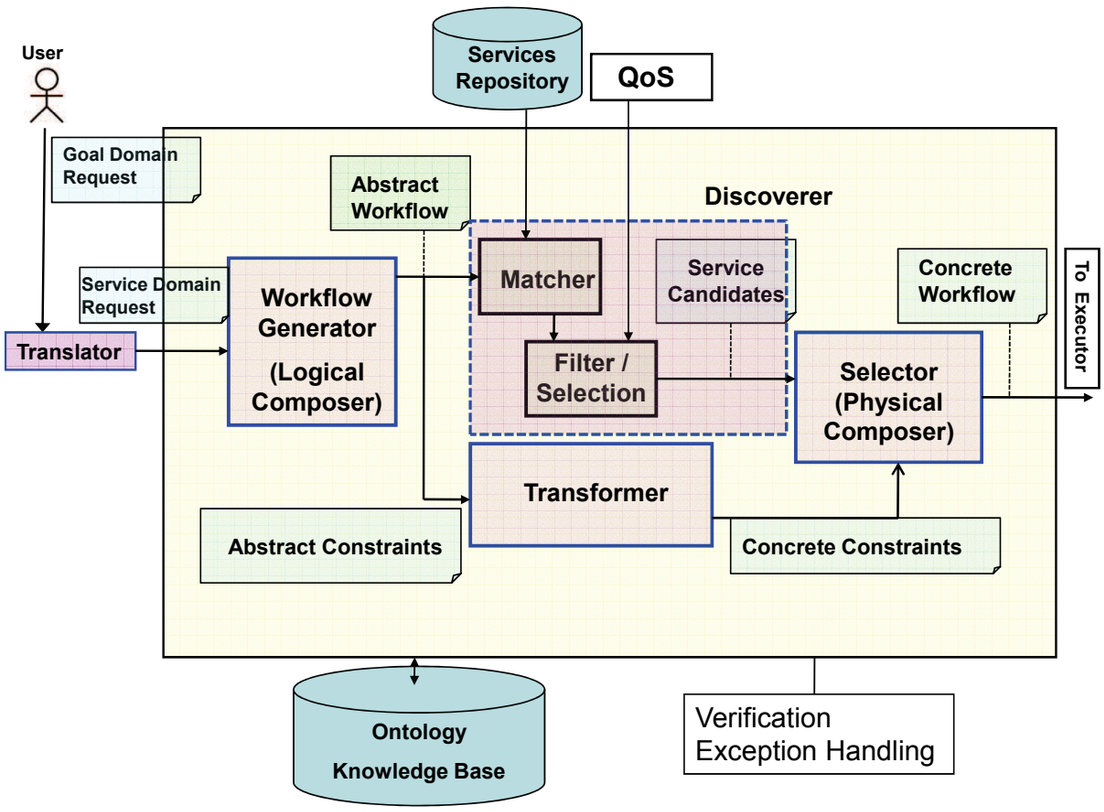
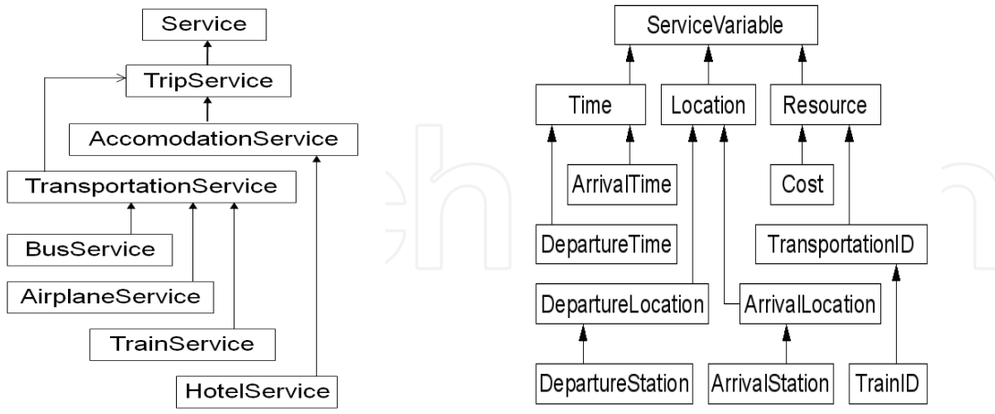


Fig. 2. ASC architecture

2.3 Service domain ontologies

For translation and transformation, many ontologies for service and service terms are needed. The transformation algorithm uses the ontologies to include all classes of service and the service variables being transformed, as shown in Fig. 3, and they will be used for the UIs as well. According to the characteristics of the various service domains, the ontologies

for the domains can be changed. If new services and conditions are added to the domain, the ontology should be changed dynamically and gradually.



(a) Service domain ontology (b) Variable domain ontology

Fig. 3. Domain ontologies for transformation

3. User interaction with service composer

It is important to decide the component parts of interactions between the user and the composer, and the contents of the interaction. Let us consider each functional block in Table 1 using this scenario.

1. Translator

When a user supplies a request about composing a new service in ASC, the request should be captured semantically. For example, consider the user request:

"I want to make a trip from a location A to a location B during October 1 – October 15. Total cost should be less than 300,000."

The request should be captured in a recognizable form by ASC. This can be in first-order logic (FOL) or via a graphical user interface (GUI). The natural-language goal can be described in the FOL form of Example 1.

Example 1. Service-level goal with abstract constraint.

① ServiceDomain(Trip).
② TripLocation(A, B).
③ TripDuration(2009-10-,2009-10-15).
④ LessThan(TotalCost,300000).

The service-level goals contain services and relations in a service and relation registry. However, the terms of constraints may still be nonterminal. For instance, the term "TotalCost" contains a compound meaning, namely the total cost of all services for the trip. Therefore, the term "Total" can be categorized as an operator (here, the sum), and the term "Cost" can be a variable of the constraint. The translator converts properties in the business domain into those in the service domain.

The user inputs the request via the UI in the translator, and the UI outputs/emits the translation result as a basic function. The user inputs a request (with both functional and nonfunctional elements) in the goal domain, with additional context information such as

Interaction Functional Blocks	Contents of Interaction with Machine	
	Input	Output
Translation	M: N/A H: - Request in goal domain - Additional context information	M: Request in service domain H: - Possible inquiries for checking translation result
Planning	M: Request in service domain H: Additional request in service domain	M: - Abstract workflow - Interim constraints H: - Possible inquiries for checking planning result
Discovery	M: - List of abstract tasks of the workflow - Additional QoS requirements H: - Additional context information - Additional QoS requirements	M: Service instances H: - Possible inquiries for checking discovery result
Selection	M: - Service instances Nonfunctional concrete constraints H: - Additional constraints Context information	M: Selected service instances optimally H: - Possible inquiries for checking selection result
Execution	M: Selected service instances H: Additional execution condition	M: - Execution trace - Exception after the execution H: - Possible inquiries for checking execution result - Possible inquiries for selecting exception handling method
Transformation	M: Intermediate constraints from the orchestration manager H: - Additional constraints in intermediate form - Additional context information	M: Concrete constraints (How can the human check this correctness?) H: - Possible inquiries for checking transformation result
Orchestration	M: Interaction with all the other blocks. H: Decision guide input	M: Interaction with all the other blocks for orchestration H: - Possible inquiries for checking orchestration management

Legend:

- *M: Machine (one of the ASC blocks) interacts with the human world via the API and defined data format, but sometimes via the UI when required.*
- *H: Human being interacts with the machine (one of the ASC blocks) via the UI.*
- *There are two types of interaction, namely input and output, but, according to the target, we differentiate the types of interactions as "input/output" for human beings, and "receive/emit" for machines (i.e., UI).*

Table 1. Interactions in ASC

additional/changed goals and constraints. The translator outputs the translation result for the user to check, and receives an input of the user reply about any additional request after the check.

2. Planner

The planner, also called the logical composer (LC), generates a workflow to fulfill the functionality of the request. The workflow comprises several abstract tasks that can reach the final goal state. The planner is inputted (receives) requests in the service domain. A request includes a top-level functionality and nonfunctionalites that affect the functionality. It becomes a sequence of abstract tasks, together with interim constraints related to the tasks generated by the planner.

The UI in the planner receives service-domain requests from the translator or obtains service-level requests from users directly. Additional service-domain requests can be supplied by users. The planner emits an abstract workflow to the discoverer or outputs abstract workflow information for the user to check. The user can then input modifications or possible additional inquiries to the planning result via the UI.

3. Discoverer

The discoverer receives the list of abstract tasks that were generated by the planner, and outputs/emits service instances for each abstract task. Users can input QoS information to the discoverer for further filtering of matched service instances.

Therefore, the UI of the discoverer receives abstract tasks from the planner, or obtains inputs of additional constraints such as QoS factors to choose more-suitable service instances for the user. In addition, it emits the service instances discovered to the selector, and outputs the discovered result to the user for checking.

4. Selector

The selector, also called the physical composer (PC), selects the optimal service instances that satisfy all the constraints from users or other composition blocks. It receives service instances from the discoverer, and emits the selected service instances to the executor.

The UI of the selector obtains the input of additional constraints or context information such as the user's additional preferences or the detailed semantics of variable terms in the constraints. It also outputs the selection result to the user for checking. The checking process can be repeated according to the user and the result.

5. Executor

The executor receives the sequence of service instances, i.e., the result of services chosen optimally by the selector, and executes the sequence. In addition, it outputs/emits the execution result to the orchestrator or the user.

The UI of the executor obtains the input of additional execution conditions or context, and outputs/emits an execution result such as the execution trace, information about exceptions, or errors. The user can choose how to deal with any exceptions via the UI.

6. Transformer

The transformer receives intermediate constraints from the orchestrator or users and emits or outputs the result as concrete constraints to the selector. It shows the transformation result to the user for checking the correctness of the result or for re-binding the constraint to another service instance.

The UI of the transformer obtains the input of additional constraints or context for the constraints in intermediate form from the user. It also outputs the transformation result, which includes linkage between constraint terms and the corresponding variables of real service instances. The UI can provide a user editing function for the links to be decided by the transformer. The procedure can be repeated several times.

7. Orchestrator

The orchestrator interacts with all the blocks both internally and via users. The orchestrator can instantiate the UIs of other blocks, and manage blocks to guide decisions. This means that other blocks can input/output and receive/emit all their user information via the UI of the orchestration manager.

4. Ontology for the ASC UI

Generally, the ontology for the UI describes the visual component, the data, and the workflow, together with a UI specification for the human-computer interaction (Tsai & Chen, 2008). The data and workflow for ASC and their ontology are the main components of the design of the ASC UI.

4.1 Ontology for data in ASC

There are two kinds of data for the UI in ASC, namely the UI itself and the composition of the data used by the UI. The ontology for the data to describe the UI is shown in Fig. 4. The UI data profiles are modeled as input, output, emitting, or receiving. The figure shows the detailed ontological structure of the four data profiles. The UI has input/output (IO) types that inherit each data profile. In addition, each data profile is used by the corresponding UI.

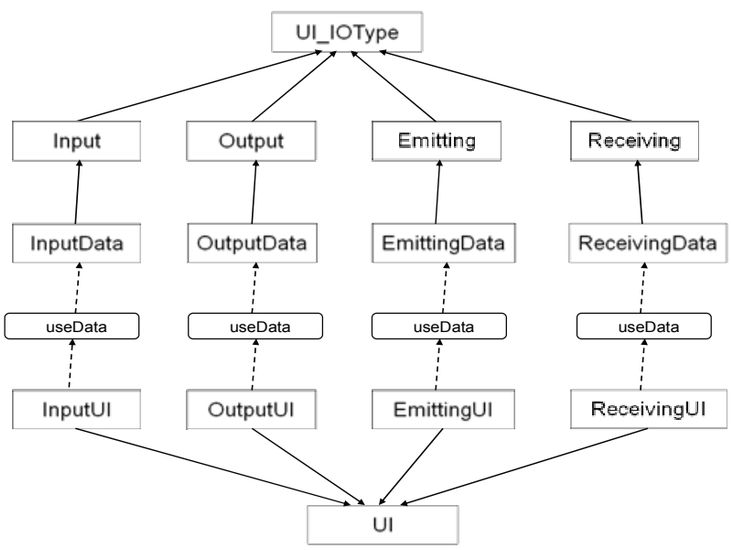


Fig. 4. Ontology for data profiles related to the UI

The data used by the UI in ASC are very extensive in various domains. As explained in the previous section, the composer comprises seven functional blocks, each having its own UI. The ontology for the main UIs and the input/output data for the whole composer are shown in Fig. 5. The request is the initial data from a user, which initiates the composer, and is important data for the operation of the composer. The request contains functional and nonfunctional elements. The ontology for a request is shown in Fig. 6. The request in the business domain may not have detailed service information, but may have abstract service information only. The request in the service domain contains request information registered in the service registry. These can be recognized by the service composer.

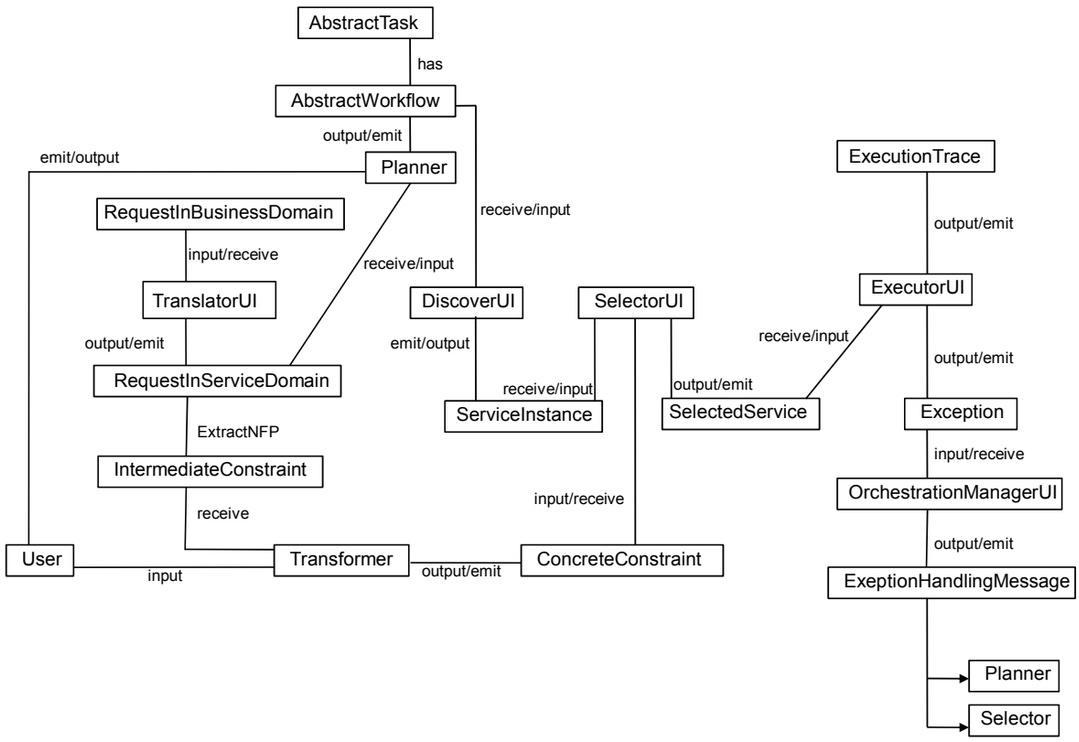


Fig. 5. Ontology for the whole composition: blocks and data

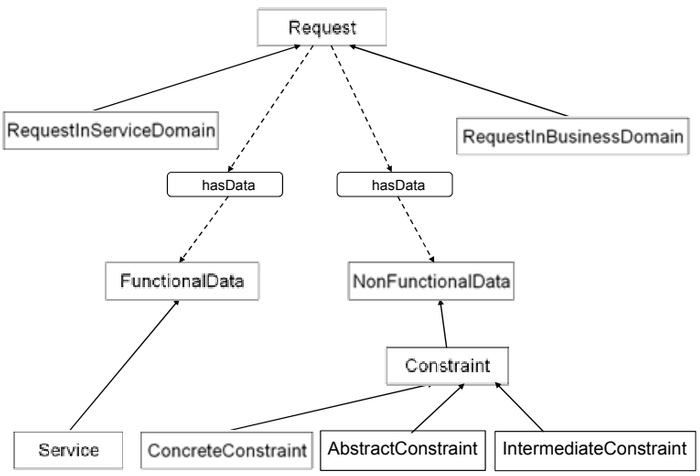


Fig. 6. Ontology for a request

4.2 Composer UI workflow

Most top-level workflows of the UI for composition are related to the functional blocks of the composer. The workflows are described in terms of a sequence of interactions among the

blocks and users, and the data of the interaction. Users supply input data that the UIs read, or deal with the output data that the UIs display. In addition, the UIs emit data that other UIs will receive. Figure 7 shows an example of a workflow of a selector UI interacting with other UIs and the user.

At first, the SelectorUI receives the ServiceInstances that have been emitted or input by the DiscovererUI or by users. It also receives any ConcreteConstraint that has been emitted by the TransformerUI. The user can input the constraints directly and the SelectorUI will read them. When the SelectorUI finishes the selection procedure, it displays the result as SelectedService. If the user wants to edit the constraint according to the result, the user sends an EditedConstraint that the SelectorUI will read. The SelectorUI may display the result (SelectedService) repeatedly until the user is satisfied. Finally, when the SelectorUI gets an OK signal from the user, it emits the result (SelectedService) to the ExecutorUI that belongs to the service executor.

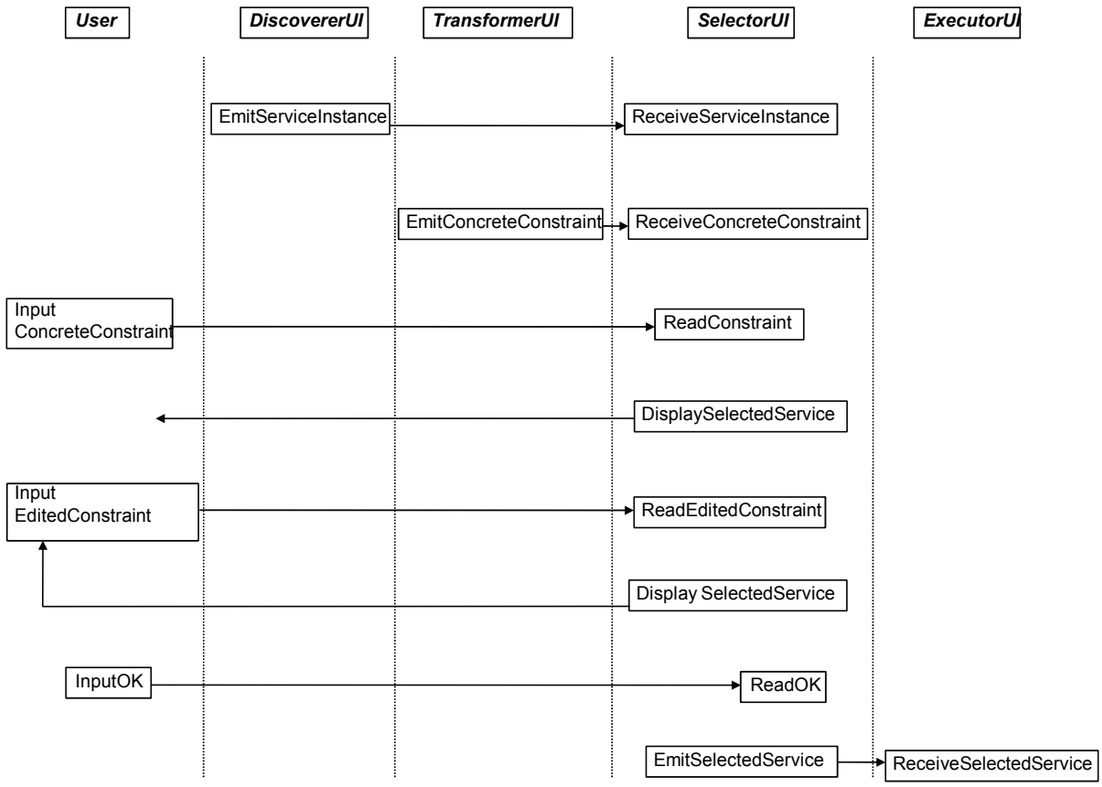


Fig. 7. Workflow of UI data handling in the SelectorUI

5. Case study of UIs for ASC

There are main UI points at seven functional blocks in the composer. Each UI can create sub-UIs such as result windows, dialogs, and message boxes for subsequent activities. Figure 8 illustrates a case of UIs for ASC of a trip domain (Takada & Paik, 2008). The UI uses the ontology, generates a web form, and sends user demands to the LC planner and a transformer. A task search engine searches the task using keywords input by users from an HTN planner ontology and a service domain ontology and proposes task candidates. The UI

provides several GUI forms, namely a task search and select form, a user constraint form, and a result form. Users use them sequentially. The HTN ontology describes information for the planner and the task search engine. It has four classes and six properties (see Fig. 9). The task search engine searches for the name of the task and the domain to which it belongs using keywords and suggests results from the HTN ontology.

An Example Scenario

The scenario is trip planning from Aizuwakamatsu (a city in Japan) to San Francisco. If a user inputs the keywords "trip aizu sanfrancisco" in the task-search GUI form (Fig. 10), the instance Trip_Aizuwakamatsu_SanFrancisco is proposed by the task search engine and the user can select it. The LC planner generates an abstract workflow as follows.

A1 = Train_Aizuwakamatsu_Koriyama
A2 = Train_Koriyama_Tokyo
A3 = Train_Tokyo_Narita
A4 = Airplane_Narita_SanFrancisco

Abstract tasks and abstract terms belong to the service-domain ontology. Abstract terms are described as term objects (output of services) and term context information, as shown in the Table 2.

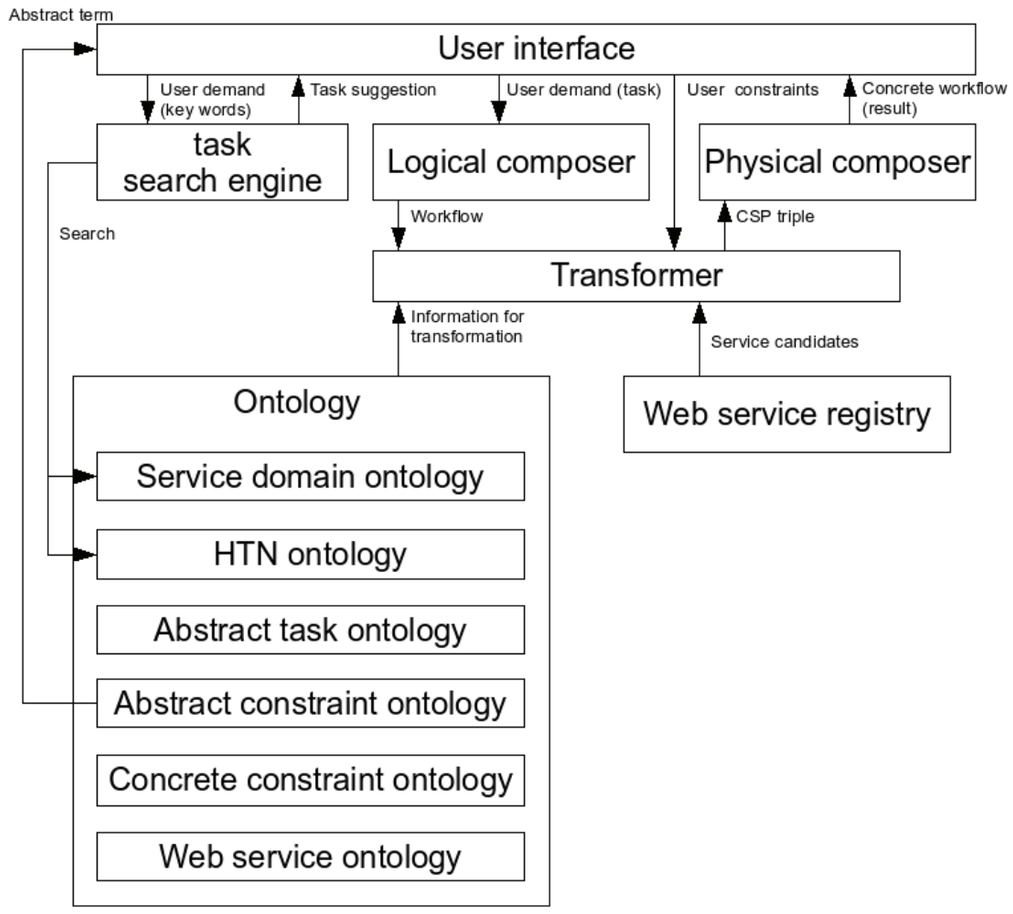


Fig. 8. An example of ASC implementation, including UI

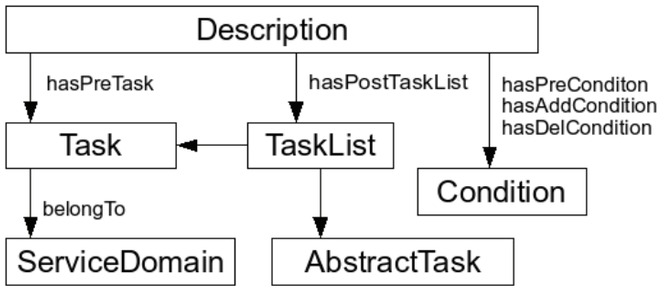


Fig. 9. HTN ontology

Abstract term	**Term object**	**Context**
AT_StartTime	TO_TimeFrom	First
AT_EndTime	TO_TimeTo	End
AT_TotalCost	TO_Cost	Sum
AT_SeatClass	TO_SeatClass	
AT_Smoking	TO_Smoking	
AT_NowArrivalTime	TO_TimeTo	Now
AT_NextDepartureTime	TO_TimeFrom	Next

Table 2. Abstract terms in the trip domain.

Instances of the trip's subclasses are proposed via the user's constraint generation. NextArrivalTime and NextDepartureTime are not proposed because there are terms for hard constraints (as opposed to user demands). Users can supply constraints such as TotalCost < $2,000 and SeatClass = Economy, as shown in Fig. 11. The user constraints are transformed in the transformer to concrete constraints such as Sum(Cost) < $2,000 and A4.SeatClass = Economy. The term object's domain is used to determine abstract tasks such as those related to AirplaneService and SeatClass. Service candidates are provided by the service registry. Each concrete service has its own QoS, departure time, cost, grade, etc. Service candidates and concrete constraints are common spatial pattern (CSP) triples. The PC selector solves the CSP triple to select the concrete services in the final selection result (see Fig. 12).

Fig. 10. Task search and select form

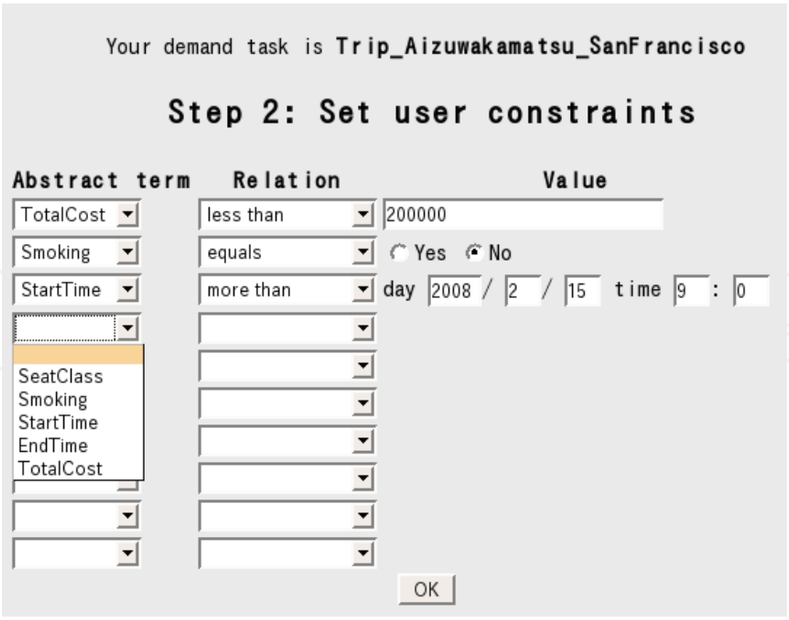
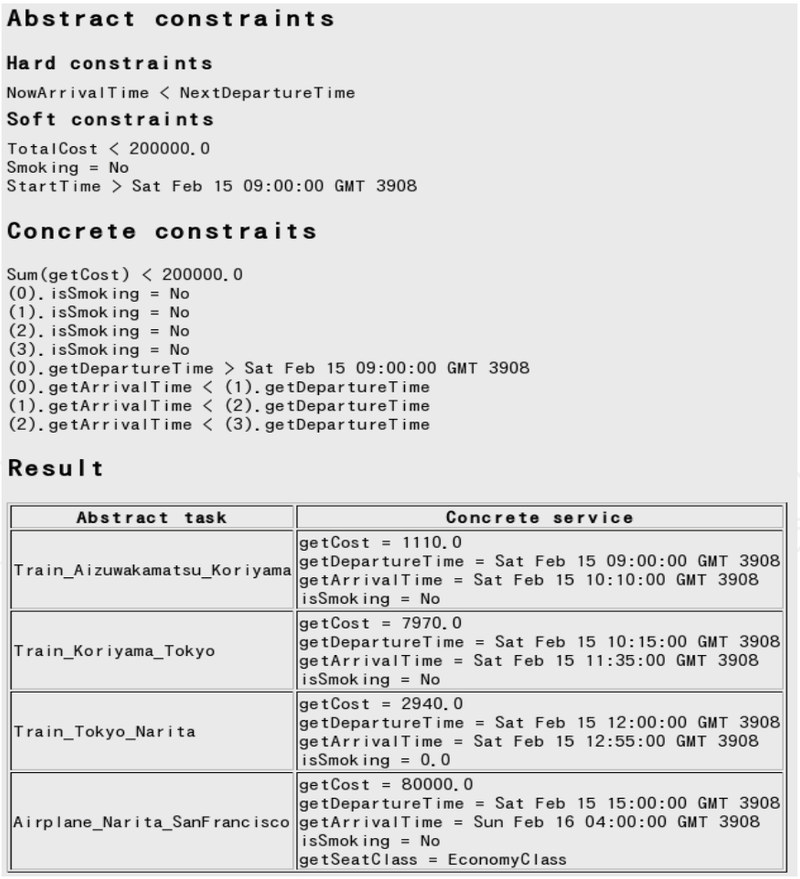


Fig. 11. User constraint form



Abstract constraints

Hard constraints

NowArrivalTime < NextDepartureTime

Soft constraints

TotalCost < 200000.0
Smoking = No
StartTime > Sat Feb 15 09:00:00 GMT 3908

Concrete constraits

Sum(getCost) < 200000.0
(0).isSmoking = No
(1).isSmoking = No
(2).isSmoking = No
(3).isSmoking = No
(0).getDepartureTime > Sat Feb 15 09:00:00 GMT 3908
(0).getArrivalTime < (1).getDepartureTime
(1).getArrivalTime < (2).getDepartureTime
(2).getArrivalTime < (3).getDepartureTime

Result

Abstract task	Concrete service
Train_Aizuwakamatsu_Koriyama	getCost = 1110.0 getDepartureTime = Sat Feb 15 09:00:00 GMT 3908 getArrivalTime = Sat Feb 15 10:10:00 GMT 3908 isSmoking = No
Train_Koriyama_Tokyo	getCost = 7970.0 getDepartureTime = Sat Feb 15 10:15:00 GMT 3908 getArrivalTime = Sat Feb 15 11:35:00 GMT 3908 isSmoking = No
Train_Tokyo_Narita	getCost = 2940.0 getDepartureTime = Sat Feb 15 12:00:00 GMT 3908 getArrivalTime = Sat Feb 15 12:55:00 GMT 3908 isSmoking = 0.0
Airplane_Narita_SanFrancisco	getCost = 80000.0 getDepartureTime = Sat Feb 15 15:00:00 GMT 3908 getArrivalTime = Sun Feb 16 04:00:00 GMT 3908 isSmoking = No getSeatClass = EconomyClass

Fig. 12. Result of trip planning scenario

6. Conclusion

The overall concept of ASC was explained first. According to this concept, all possible interaction points and contents were investigated. To devise UIs for ASC, the data ontology, UIs, and workflows were designed and introduced. Finally, examples of UIs for ASC based on this design were given.

The complete ontology set for the top-level UI was introduced, and an example of workflow for service selection was illustrated. It can be extended to other UI workflows and detailed data ontologies. Mapping to real GUIs is for interested readers to consider. We should remember that there are many possibilities for variation in service composition, particularly for goals and services that are more flexible.

7. Reference

Agarwal, V.; Chafle, G; Mittal S.; & Srivastava, B. (2008) Understanding Approaches for Web Service Composition and Execution, Proceedings of COMPUTE 2008, Bangalore, India, 2008

Andrews T.; & 16 others. (2003) Business Process Execution Language for Web Services version 1.1, BEA Systems, IBM, Microsoft, SAP AG, and Siebel Systems (May 2003). http://www-106.ibm.com/developerworks/library/ws-bpel/

Arkin A.; & 11 others. (2002) Web Service Choreography Interface (WSCI) 1.0, draft specification, BEA Systems, Intalio, SAP AG, and Sun Microsystems (2002). http://www.sun.com/software/xml /developers/wsci/wsci-spec-10.pdf

Baresi, L.; Guinea, S.; Pistore, M; & Trainotti, M. (2009) Dynamo + Astro: An Integrated Approach for BPEL Monitoring, Proceedings of IEEE International Conference on Web Services (ICWS'09), pp. 230–237, Jul. 2009, L.A, USA.

Claro, D.; Albers, P; & Hao, J. (2006). Web Services Composition in Semantic Web Service, Processes and Application, Springer, New York

Hassine, A.; Matsubara, S.; Ishida, T. (2006) A Constraint based Approach to Horizontal Web Service Composition, Proceedings of ISWC 2006, Athen, U.S.A

Klusch, M; Fries, B.; & Sycara, K. (2006) Automated Semantic Web Service Discovery with OWLS-MX, Proceedings of AAMAS, Hakodate, Hokkaido

Kona, S.; Bansal, A.; Blake, M.; & Gupta, G. (2008) Generalized Semantics-based Service Composition, Proc. of IEEE Int. Conf. on Web Services, p. 219-227, Beijing, China

Lecue, F.; Delteil, A. (2007) Making the Difference in Semantic Web Service Composition, Proceedings of AAAI-2007, pp. 1383-1388, British Columbia

Lecue, F.; Delteil, A.; Leger, A. (2007) Applying Abduction in Semantic Web Service Composition, IEEE International Conference on Web Services (ICWS 2007), pp. 94-101, Salt Lake City/Utah, USA

Narayanan, S.; McIlraith, S. (2002). Simulation, Verification and automated Composition of Web Services, In Proceeding 11th Int. Conf. WWW, Honolulu, Hawaii, USA

Nareyek, A.; Freuder, E.; Fourer, R; Giunchiglia, E.; Goldman, R.; Kautz, H.; Rintanen, J; & Tate, A. Constraints and AI Planning, IEEE Intelligent Systems, Mar./Apr. 2005, pp. 62-70

Nau, D.; Au, T.; Ilghami, O.; Kuter, U.; William Murdock, J; Wu, D.; & Yaman, F. (2004). SHOP2: An HTN Planning System, Journal of Artificial Intelligence Research

Oh, S.; Lee, D.; & Kumara, S. (2008) Effective Web Service Composition in Diverse and Large-scale Service Networks, IEEE Transactions on Services Computing, 2008, vol. 1, no. 1, pp. 15-32.

OWL Services Coalition, (2003) OWL-S: Semantic Markup for Web services, OWL-S White Paper http://www.daml.org/services/owl-s/1.0/owl-s.pdf

Paolucci, M.; Kawamura, T.; Payne, T.; & Sycara, K. (2002) Semantic Matching of Web Services Capabilities, Proceedings of the First International Semantic Web Conference, Sardinia, Italy

Shi, Y.; Zhang, L.; & Shi, B. (2004). Exception Handling of Workflow for Web Services, Proceedings of International IEEE Conference Computer and Information Technology, pp. 273-277, Shanghai, 2004

Takada, H. & Paik, I. (2008). Design of General User Interface for AutomaticWeb Service Composition, Joint Workshop on Frontier of Computer Science and Technology (FCST), Nagasaki, Japan, Dec. 2008.

Takada, H.; Paik, I. (2009). Transformation of Non-Functional Properties for Automatic Service Composition, Proceedings of The 3rd Workshop on Non-Functional Properties and SLA Management in Service-Oriented Computing NFPSLAM-SOC'09 , Nov. 2009, Stockholem, Sweden

Tsai, W.; Huang, Q.; Elston, J.; & Chen, Y. (2008) Service-Oriented User Interface Modeling and Composition, Proceedings of IEEE International Conference on e-Business Engineering, pp. 21-28, Xian, China

WSMO. (2005) The Web Service Modeling Ontology (WSMO) Primer. Final Draft. Available at: http://www.wsmo.org/TR/d3/d3.1/v0.1/

User Interfaces

Edited by Rita Matrai

ISBN 978-953-307-084-1
Hard cover, 270 pages
Publisher InTech
Published online 01, May, 2010
Published in print edition May, 2010

Designing user interfaces nowadays is indispensably important. A well-designed user interface promotes users to complete their everyday tasks in a great extent, particularly users with special needs. Numerous guidelines have already been developed for designing user interfaces but because of the technical development, new challenges appear continuously, various ways of information seeking, publication and transmit evolve. Computers and mobile devices have roles in all walks of life such as in a simple search of the web, or using professional applications or in distance communication between hearing impaired people. It is important that users can apply the interface easily and the technical parts do not distract their attention from their work. Proper design of user interface can prevent users from several inconveniences, for which this book is a great help.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Incheon Paik (2010). User Interface for Automatic Service Composition, User Interfaces, Rita Matrai (Ed.), ISBN: 978-953-307-084-1, InTech, Available from: http://www.intechopen.com/books/user-interfaces/user-interface-for-automatic-service-composition

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen