Task-Driven Service Discovery and Selection

Kyriakos Kritikos
HIIS, ISTI-CNR
Via G. Moruzzi 1
56124 Pisa, Italy
+39 050 3153117
Kyriakos.Kritikos@isti.cnr.it

Fabio Paternò
HIIS, ISTI-CNR
Via G. Moruzzi 1
56124 Pisa, Italy
+ 39 050 3153066
Fabio.Paterno@isti.cnr.it

ABSTRACT

Services are becoming more and more widely used. When designing interactive applications based on services one important issue is how to identify those services most relevant for the application functionalities. The proposed approach takes as input a task model, which includes the user's view of the interactive system, and an ontology capturing the application domain, and automatically discovers a set of ordered service descriptions for each system task of the model. The discovered descriptions can be used in order to invoke a particular service operation that fulfils a task’s required functionality. In this way, the whole application functionality can be realized by a set of service operations without writing a single line of code. As a result, the application development time is significantly reduced and it is possible to complete the development of interactive front-ends by integrating our solution in existing model-based HCI approaches.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques – User Interfaces, H.5.2 [**Information Interfaces and Presentation**] User Interfaces – User-Centered Design, H.3.5 [**Information Storage and Retrieval**]: Online Information Systems – Web-based Services

General Terms

Design, Human Factors, Algorithms.

Keywords

Service Front-Ends, Service Discovery, Interactive Service-Based Application Design, Semantics

1. INTRODUCTION

Service-orientation is a design paradigm that is adopted in the design of an increasingly great number of applications. The reason for this wide adoption lies in the capabilities of services to be composed into new added-value applications. In this way, a whole application can be built from scratch or existing intra- or inter-organizational applications can be integrated together in a seamless and loosely-coupled way.

The main mechanisms for supporting service integration are the service discovery and composition processes. The service discovery process is realized by service discovery algorithms, which take as input a user-provided *service query* formulated in a service description and a set of service advertisements (i.e. descriptions) stored in a service registry and produce as output a categorization of the service advertisements based on their matching degree with the service query. The most prominent among the service discovery algorithms use a combination of Information Retrieval (IR) and Semantic Web (SW) techniques in order to increase the accuracy of the service discovery results. The service composition process can be realized by various approaches (see for example [5]).

By exploiting the most promising service discovery and composition approaches, an interactive application can be designed and built from scratch without writing a single line of code. However, such an application needs to interact with the end-user. A number of HCI approaches have been proposed [1,2,3,6,9] that remedy this limitation by building service-based applications with a UI customized according to the context-of-use, i.e. the end-user’s capabilities, level of expertise, platform, and working environment [13]. Such work often follows a Model-Driven Approach (MDA) that starts with high-level descriptions (such as task models) and ends with the generation of the final UI code.

Unfortunately, a big disadvantage of the above HCI approaches is the limited automatic support. In particular, as far as functionality is concerned, the designer has to manually select those services that are able to fulfil the functionality of the application.

The above limitation can be overcome by using more formal representations, which capture the semantics of terms, and reasoning mechanisms that exploit them [13] in order to automate the various activities that the designer has to perform. To this end, in this short paper we propose an MDA approach, which is able to automatically produce a semantically-enriched task model that can be used to obtain interactive applications with some application logic implemented through external services. In particular, our approach starts with the task model of an interactive application and transforms it into a semantic service model by associating the task application objects to concepts of an ontology of the associated application domain. Then this service model is sent to a semantic service discovery engine as a set of service queries in order to discover those services that are able to fulfil the application’s functionality. Finally, the discovery results are merged into the service model. Our approach provides designers with useful automatic support. Moreover, the application development time is reduced, as a part, if not all, of the application’s functionality is realized through the use of external services. In addition, our approach can be used in the above HCI approaches as a particular automated component by replacing those parts that usually require manual intervention to fulfil the same goal. In this way, a value-added approach can be obtained that takes into consideration both UI and functional aspects in order to effectively design an interactive and context-of-use-aware service-based application.

2. INTERACTIVE APPLICATIONS DESIGN

The design of an interactive application should take into account both UI and functional aspects and the implications that one aspect may have on the other one. In addition, this design should be performed in parallel following an MDA approach.

Our envisioned MDA approach starts with a task model, which provides important information, such as all the interactive and system tasks of the application, and their temporal order and interactions. As this information concerns both UI and functionality aspects, which are separable, the task model is transformed into two parts according to each aspect: an *Abstract User Interface* (AUI) and a *service model*. Each part is processed in parallel and then the final results are joined together into the application code. While work for deriving the user interface has been carried out (e.g. [12]) there are various open issues on how to integrate existing Web services. In this work we present a solution for the service discovery and selection part.

The usual path followed for the UI's model-based design [4] is that the AUI, an interaction modality independent model, is transformed into a *Concrete User Interface* (CUI), which is further transformed into the *Final User Interface* (FUI). The CUI is a UI model dependent on the user's platform and interaction modality but independent of the UI implementation language, while a FUI is an operational UI running on a particular computing platform either by interpretation or by execution.

The design of the application's functionality follows a similar path but has an additional level of abstraction. At the topmost level, the *service model* is an abstract representation of the application's functionality defined as an orchestration of services and structured as a hierarchical tree having a similar formation to the task model. In this hierarchical tree, the higher the level of the node, the more composite is the service that it represents, while service operations are represented by the nodes at the lowest level. Each node is connected with its siblings by temporal operators in order to preserve the temporal semantics of the initial task model.

A service model is transformed into an *enriched service model* when it is populated by the service discovery engine with existing services that realize the functionality of its nodes. The enriched service model depends on the pragmatics of the services world. This means that there may be a part of the application's functionality which is not realized by existing services. In addition, there may be cases where changes to the UI must be made. For instance, some additional application objects must be attached to particular interaction tasks in the event that some input of the associated services is missing.

The product of service concretization on an enriched service model is a *concrete service model*. In particular, service concretization is performed by selecting only one from the candidate services of particular nodes of the enriched service model according to functional and context-of-use requirements. This selection must take place at the leaf nodes of the enriched service model according to the pragmatics of the services world, which indicate that it is impossible to discover services that correspond to higher level nodes as this would require the description of the temporal semantics of their operations. Indeed, the majority of the service concretization approaches perform service selection at the leaf/operation level, i.e. they select one service from the candidates for each task of the service-based application. A concrete service model is tightly coupled to the corresponding CUI as they reference each other's content and they are finally transformed into the application code. Moreover, similar to the case of the enriched service model, there may be many candidate services for a particular node of an enriched service model, but none of them satisfies the context-of-use requirements. In this case, the functionality encapsulated by this node should be implemented by the developer.

After all parallel reification transformations take place under the designer's supervision, the final outcome of the design process is the application code, which is produced by the developer and composed of three parts: the FUI, the implemented functionality code, and the invocation code of the selected services.

3. PROPOSED APPROACH

We propose an approach that takes as input a CTT [11] task model and a domain model, which are both provided by the designer, and automatically produces an enriched service model. The automation is carried out through the use of semantics. In particular, the domain model is expressed with a domain ontology and a term-to-ontology concept matching algorithm is used in order to map a task's application objects to concepts from the ontology. In this way, the task description is transformed into a semantic service query that is sent to a semantic service discovery engine. The result is a set of semantic service descriptions that are associated to the particular task. Thus, all the system tasks of a task model are eventually associated to a set of services that can be used to realize their functionality. Moreover, each task-associated set of services is classified into various categories that characterize the degree of match of the service with the task's functionality. In this way, the designer is assisted in selecting the appropriate service that best suits its functional goals.

The proposed environment is named *User-Centered Service Discovery System* and is composed of the following components: *Service Discovery Controller*, *Transformer*, *Term Matcher*, and *Service Matchmaker*. The *Service Discovery Controller* interacts with the *Model-Based Service Front-End Editor* and coordinates the tasks and I/O of the other components. In the following, we analyze the functionality of the rest of the components.

3.1 Transformer

The *Transformer* is responsible for performing three main transformations. The first transformation concerns the production of the service model from the task model, while the second one concerns the production of a set of service queries from the service model. The third transformation is used to integrate the service discovery results with the service model, thereby producing the final, enriched service model.

During the *Task-to-Service Model* transformation, the given task model is transformed into a service model by selecting only the abstract and system tasks, while respecting the hierarchy and the temporal semantics of the task model. This transformation is meaningful if the following three assumptions hold: a) the designer specifies at least the application objects manipulated by each task; b) the application objects manipulated by interaction tasks can be considered as input to the remaining system tasks, if the latter tasks have objects with the same name; c) if a system task has application objects that match those of previous system tasks, then these objects can be considered as its input.

The transformation is performed in a Depth First Search (DFS) fashion. Depending on the type of node visited for the first time, the following cases are considered:

- A cognitive (user), interaction, or an abstract task node with no children is removed. However, the application objects of interaction tasks are stored in a Global Object List (GOL).
- Leaf-node system tasks are renamed as *operation* type tasks, while higher-node system tasks are renamed as *service*. The following checks are performed:
    - If an object name matches a GOL object name, then an *input* node is created, which is a copy of the GOL object, and attached to the task node.
    - If there are two copies of the same object, and this object matches a GOL object, then the first copy will be attached as an *input* node and the second as an *output* node to the task node.
    - If an object name is not matched, then an output node is created, which is a copy of it, and is attached to the task node.
- If a node is an abstract task with system task descendants, then it is renamed as *service*.

Depending on a non-root node's placement, the following two cases has to be considered when removing it: a) if the node is the left-most or rightmost child of its parent, then its right or left temporal operator also has to be removed, respectively; b) otherwise, we have to remove this node's right operator and connect its left sibling with its right one with its left operator.

During the revisiting of a node, the number of its children must be checked. If this number equals to one, then this node is replaced by its sole child. If this number is zero, then this node is removed.

The *Service Model-to-Service Queries* transformation produces a service query for each operation task of the service model. Each query is represented by a service description with one operation, where the service is named after the parent and the operation is named after this operation task, while the I/O parameters are named after the input and output objects of this task.

During the *Service Model and Discovery Results-to-Enriched Service Model* transformation, the service discovery results produced for the generated service queries are merged into the service model. In particular, for each operation task that corresponds to a particular service query, its discovery results are enclosed within its description. These results are represented by the candidate services' names, URIs, and full descriptions.

3.2 Term Matcher

The *Term Matcher* component is responsible for enriching the generated service queries with the appropriate semantics by mapping their I/O parameters to concepts of the domain ontology. This mapping is performed by using the approach analyzed in [8]. According to this approach, the relatedness between an I/O parameter and an ontology concept is calculated by computing the name similarity of this parameter not only with this concept but also with this concept's related ontology terms (its super-concepts). The name similarity between two terms is assessed by the Google distance metric [7], which uses the relative frequency with which the compared terms appear on the Web and is well-founded on information distance and Kolgomorov complexity.

3.3 Service Matchmaker

The *Service Matchmaker* component is responsible for matching the semantic service queries with the service advertisements stored in its registry. For realizing this component's functionality, we have selected OWLS-MX [10], a prominent hybrid (i.e. semantic and IR) service matchmaker written in Java. The rationale of this selection is that this service matchmaker is able to match semantic service descriptions, produces categorized service discovery results with high accuracy, and provides a ranking of the service advertisements in each result category based on the textual IR similarity of the ranked service with the service query in terms of their I/O parameters. In this way, the designer is assisted in selecting the first proposed service of the best category to realize the functionality of the corresponding system task.

4. CASE STUDY

In this section, the functionality of our proposed approach is illustrated by presenting a case study, which is drawn from the Web Shopping application domain and concerns the ordering of a particular book by a user. The task model pertaining to this case study is given in Figure 2. We start by analyzing the task model and then we explain the various models produced by our system.

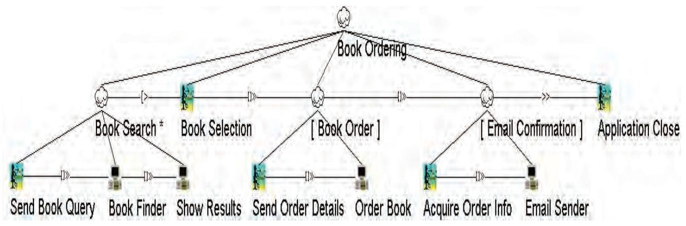


Figure 2. The task model analyzing the book ordering application

The task model's root abstract task, named "Book Ordering", represents the user's main goal and is realized by sequentially executing five tasks, namely: "Book Search", "Book Selection", "Book Order", "Email Confirmation", and "Application Close".

The "Book Search" task concerns the procedure of searching for a book and is decomposed into three sequential tasks. The first task, named "Send Book Query", is an interaction task used to retrieve the book search parameters of the user's query (the author's name and the book's title). The second task, named "Book Finder", is a system task that searches for the books (output) that match the user query terms (input). The third task is a system task that shows the details of the discovered books (input).

The "Book Selection" is an interaction task that interrupts the "Book Search" task and enables the user to select the book that he desires. This task is followed by the "Book Order" optional abstract task. The latter task represents the steps followed for ordering the selected book and is decomposed into two tasks, namely "Send Order Details" and "Order Book". The first task is interactive and enables the user to provide the ordering details (user name, tel. number, credit card number, and quantity). It is followed by the second task that is a system task that performs the actual book ordering (its output is the actual order).

The "Email Confirmation" is an abstract optional task concerning the procedure of sending a confirmation email to the user's mobile phone. It is decomposed into two tasks, namely "Acquire Order Info" and "Email Sender". The first task is interactive and allows the user to request for getting the order details, while the second one is a system task that sends the order details (input) to the user's mobile phone. The "Application Close" is the last child of the root "Book Ordering" task. It is interactive and enables the user to close the interactive application.

The above task model is given as input to the *Transformer* component of our system, which transforms it to a service model that is depicted in Figure 3, where the *service* tasks are represented with the *abstract* task symbol and the *operation* tasks are represented with the *system* task symbol. As can be easily seen, the interactive tasks of the task model have been removed, while the abstract tasks with only one system task child have been replaced by this child. In a next step, the *Transformer* produces a set of four (OWL-S) service queries out of the service model. These four queries are then given to the *Term Matcher* component, which enriches them by mapping their I/O concepts to the concepts of the domain ontology. Next, the enriched service queries are sent to the *Service Matchmaker*, which matches them one by one and produces their discovery results. Finally, all the discovery results are merged into the service model by the *Transformer* in order to produce the enriched service model.

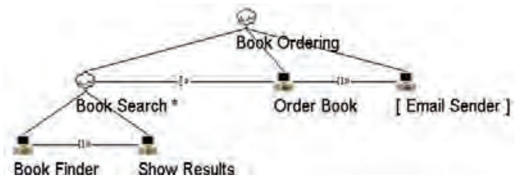
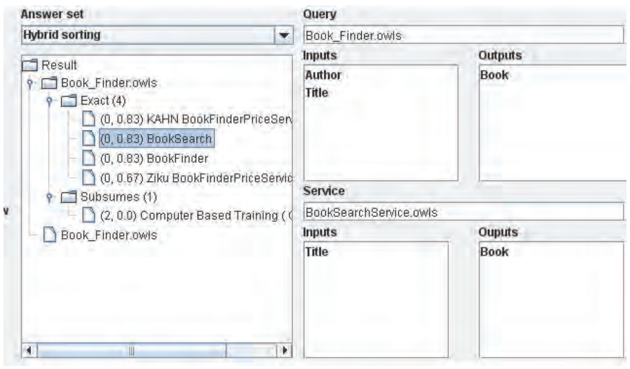


Figure 3. The service model produced from the task model



```
Answer set                                  Query
Hybrid sorting                              Book_Finder.owls
Result                                      Inputs        Outputs
  Book_Finder.owls                          Author        Book
    Exact (4)                               Title
      (0, 0.83) KAHN BookFinderPriceServ
      (0, 0.83) BookSearch
      (0, 0.83) BookFinder
      (0, 0.67) Ziku BookFinderPriceServic
    Subsumes (1)                            Service
      (2, 0.0) Computer Based Training ( (  BookSearchService.owls
  Book_Finder.owls                          Inputs        Ouputs
                                            Title         Book
```

Figure 4. The first enriched service query's discovery results

The discovery results pertaining to the first enriched service query are shown in Figure 4. As can be seen, two result categories have been produced having their results sorted according to their textual similarity with the user query (Figure 4, left side).

5. ACKNOWLEDGEMENTS

This work was carried out during the tenure of an ERCIM "Alain Bensoussan" Fellowship Programme and was supported by the ServFace (http://www.servface.eu) ICT EU project.

6. CONCLUSIONS

In this paper, we have presented a solution to the design of interactive-based applications that takes into consideration both UI and functional aspects. This approach automatically transforms an initial task model to an enriched service model. The latter model specifies the way services can be combined in order to realize the functionality of the application. This model assists the designer by indicating which system tasks are associated to existing services and enables him to select the highest ranked service from the sorted service candidate list that is provided inside the model's description for each system task. In this way, from-scratch implementation of these tasks is not needed so the application development time is reduced. In addition, this model can be used by service composition engines to perform the service concretization. Finally, the proposed system can contribute to the development of interactive front-ends.

7. REFERENCES

[1] K. Arabshian, C. Dickmann, and H. Schulzrinne. Ontology-Based Service Discovery Front-End Interface for GloServ. In *ESWC*, volume 5554 of LNCS, pages 684–696, Heraklion, Crete, Greece, 2009. Springer.

[2] K. Breiner, O. Maschino, D. Görlich, and G. Meixner. Towards automatically interfacing application services integrated into an automated, model-based user interface generation process. In *MDDAUI Workshop at IUI*, Sanibel Island, Florida, USA, 2009. Ceur.

[3] G. Broll, S. Siorpaes, M. Paolucci, E. Rukzio, J. Hamard, M. Wagner, and A. Schmidt. Supporting Mobile Service Interaction through Semantic Service Description Annotation and Automatic Interface Generation. In *Semantic Desktop and Social Semantic Collaboration Workshop in ISWC*, Athens, GA, USA, 2006. ISSN 1613-0073.

[4] G. Calvary, J. Coutaz, D. Thevenin, Q. Limbourg, L. Bouillon, and J. Vanderdonckt. A Unifying Reference Framework for multi-target user interfaces. *Interacting with Computers*, 15(3):289–308, June 2003.

[5] F. Casati, S. Ilnicki, L.-j. Jin, V. Krishnamoorthy, and M.-C. Shan. Adaptive and dynamic service composition in eflow. In *CAiSE*, pages 13–31, Stockholm, Sweden, 2000. Springer-Verlag.

[6] A. Celentano, S. Faralli, and F. Pittarello. The situation lens: Looking into personal service composition. In *ER Workshops*, pages 165–174, Barcelona, Spain, 2008. Springer-Verlag.

[7] R. L. Cilibrasi and P. M. B. Vitanyi. The Google Similarity Distance. *IEEE Trans. on Knowl. and Data Eng.*, 19(3):370–383, 2007.

[8] J. Gracia and E. Mena. Web-Based Measure of Semantic Relatedness. In *WISE*, pages 136–150, Auckland, New Zealand, 2008. Springer-Verlag.

[9] D. Khushraj and O. Lassila. Ontological Approach to Generating Personalized User Interfaces for Web Services. In *ISWC*, volume 3729 of LNCS, pages 916–927, Galway, Ireland, 2005. Springer.

[10] M. Klusch, B. Fries, and K. Sycara. OWLS-MX: A hybrid Semantic Web service matchmaker for OWL-S services. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(2):121 – 133, 2009.

[11] F. Paternò. *Model-Based Design and Evaluation of Interactive Applications*. Springer-Verlag, 1999.

[12] F. Paternò, C. Santoro, and L. D. Spano. Support for authoring service front-ends. In *EICS*, pages 85–90, Pittsburgh, PA, USA, 2009. ACM.

[13] N. Partarakis, C. Doulgeraki, A. Leonidis, M. Antona, and C. Stephanidis. User interface adaptation of web-based services on the semantic web. In *UAHCI*, pages 711–719, San Diego, CA, USA, 2009. Springer-Verlag.