# Ad-Hoc Usage of Web Services with Dynvoker

Josef Spillner, Marius Feldmann, Iris Braun, Thomas Springer, and Alexander Schill

Dresden University of Technology, Dept. of Computer Science, Chair for Computer Networks, 01062 Dresden, Germany
{josef.spillner,marius.feldmann, iris.braun,thomas.springer,alexander.schill}@tu-dresden.de

**Abstract.** While web services are often targeted at machine-to-machine communication, they are also increasingly used directly in the interactions between humans and machines. Instead of developing specialised client applications for the invocation of these services, a generic human-driven ad-hoc usage is beneficial in many scenarios, including rapid service testing and dynamic inclusion of services as plugins into applications. We argue for the need for such a usage and extract requirements for generic web service clients. We then present a few selected use cases and introduce the Dynvoker client which already passes the majority of evaluation criteria. With its technical capabilities and open and vivid development, we consider it the most suitable and flexible generic client available and therefore highlight its role as a central component in a user-centric web service research project.

## 1 Introduction

Rich and thin client applications provide a human interface to computational functionality. In rich client applications, the interface and the functional part are tied together, contrasting the rather loose coupling in thin clients. Some functional parts are designed to allow primarily programmatic access and provide an API over the network. Among these are Web Services, which are in most cases self-described, stateless components. Sometimes, only an informal, textual description of the interface exists, and the provider offers custom-made toolkits to foster client development. Nevertheless, there are widely used file format specifications to describe aspects like the message syntax, operational semantics and non-functional properties. When these descriptions are present, it is possible to call the services with generic clients by introspecting the descriptions and deducing behavioural information.

There are several use cases where combining evolving services with existing applications can benefit from ad-hoc usage. In applications with plugin support, many plugins rely on a specific service interface. With automatically generated forms to access the service, the service can evolve and be improved without the need to transfer a new GUI component to the client. Furthermore, once a form generator for a GUI technology has been developed, it allows the access to

P. Mähönen, K. Pohl, and T. Priol (Eds.): ServiceWave 2008, LNCS 5377, pp. 208–219, 2008.

all existing services without the need for cooperation from the service authors. Even if a custom client is going to be developed, a generic client can assist in rapid functional tests. Generic clients are also useful in mobile scenarios to avoid development and installation of custom clients [1].

The process of generating the user interface based on various information from the service provider and enabling an interaction between the user and the service consists of a number of steps which are being researched in the area of WSGUI, or *Web Services Graphical User Interfaces* [2]. Dynvoker, a dynamic service explorer and invoker, is an implementation of these concepts, generally called generic web service client or more technically WSGUI engine. Compared to existing approaches, we contribute such a WSGUI engine which accepts multiple web service description formats as its input and can generate user interfaces and forms in various output formats. This design choice leads to greater user experience by offering a higher number of services on a higher number of devices, but also presents some challenges in handling the differences between the formats. We will show that this design choice is superior to single-format implementations and will outline results from practical experience with our open-source implementation.

The structure of this paper is as follows: First, related work is evaluated, concentrating on run-time user-centric web service interaction tools. Afterwards, a requirements analysis of features common to most of these approaches is performed, followed by a number of features in Dynvoker which help to fulfil the requirements. The text concludes with a brief report on process handling and an outlook on how Dynvoker will be used in an existing research project.

## 2 Related Work

Generating user interfaces dynamically to access a well-defined interface or an underlying data model has been in the focus of research for several years. Central ideas used in UI generation for web services have been extracted from similar fields of research and apply as well to areas such as automatic dialogue generation from the underlying configuration schema[1] or inferring user interfaces from database models [3]. However, the specific field of ad-hoc usage of web services by automatically created UIs is only unsatisfactorily covered by research and development projects. Though a lot of preliminary work has been invested into this area, only a few implementations are still available. An early approach that clearly formulated the intention to dynamically integrate web services into a user interface has been the web service User Interface (WSUI) initiative that had the goal to embed services as visual components into web portals. The initiative has stopped its activities shortly after its foundation. Neither the website nor the specification draft are still available.

A further historic approach which forms an important building block for current projects is the original WSGUI project [4] which influenced Dynvoker in many aspects. Besides inferring basic information about the user interfaces from

[1] KXForms dialogue generation: `http://www.lst.de/~cs/kode/kxforms.html`

the web service it introduced the annotation format GUI Deployment Descriptor (GUIDD)[2] that enables aspects such as attaching multilanguage human-readable labels to input or output fields. After merging inferred information and the optional GUIDD data, the resulting form based screen data was transferred via XSLT to a concrete user GUI representation.

The open source library Xydra[3] can be used for ad-hoc creation of UIs for web services. It produces XHTML files with web forms based on an inference mechanism for WSDL and associated XML Schema. Besides describing service annotations based on ontologies it employs a technique called TreePath to be able to represent arbitrary XML structures as key-value pairs required by XHTML browsers. The project development was stopped in 2003.

Further purely inference-based mechanisms are the Dynamic SOAP Portlet[4] and the SOAPClient[5]. Whereas the first one follows a portlet concept that dynamically offers a UI for a generic client for web services [5], the second one can be seen as a testing tool for web service development. It creates on-demand a rudimentary HTML form for all operations found within a WSDL file specified by the user. There is no information available about the interior of this application.

Some common development tools offer support for web service UI creation. An advanced implementation is the XML Forms Generator[6] available as Eclipse plug-in. Though it does not fit into the category of ad-hoc UI generation, it offers interesting concepts relevant for the on-demand creation process as well. It analyses a WSDL document and enables combining derived data with an Eclipse Modelling Framework model like an XML Schema file or a UML diagram for providing information such as type information. The tool generates an XHTML output with associated CSS style sheet. For REST-based interfaces described by WADL, the NetBeans IDE[7] offers a forms inference tool for testing services during the development time.

Academic publications covering the topic of ad-hoc UI generation for web services are rare. [6] directs a focus on dynamic creation of multimodal UIs using XForms and VoiceXML elements generated from WSDL inference. The transformation to concrete UI representations is based on XSLT. Though it is pointed out that service descriptions can be imported to a system specific proxy server for providing additional information to improve the quality of the UI, no details about this possibility are given. Furthermore, various future research intentions are mentioned though none of them have been realised yet. In [7], a further system for UI generation at runtime is proposed using four different WSDL annotation files containing UI related information. The system supports a profile-driven adaptation to different user-clients. No arguments are provided

[2] GUIDD specification: `http://wsgui.berlios.de/guidd/`
[3] Xydra generic client: `http://www.extreme.indiana.edu/xgws/xydra/`
[4] Dynamic SOAP Portlet: `http://soap-portlet.sourceforge.net/`
[5] SOAPClient: `http://soapclient.com/`
[6] XML Forms Generator: `http://www.alphaworks.ibm.com/tech/xfg`
[7] NetBeans: `http://www.netbeans.org/`

for the chosen system architecture. The different WSDL annotation formats are not described in any detail beyond their overall focus.

The mentioned approaches do not offer a generic solution covering various service interface descriptions such as WADL or WSDL at once. All of them are bound to concrete service technologies. Furthermore, XSLT is a quite common means to realise the transformation to concrete UI representations, although the difficulties regarding its complexity are well-known. Only a few of the analysed projects are still active and offer a directly testable implementation. Despite some of them providing basic information about the overall mechanisms, they mainly do not provide any internal details.

## 3 Aspects of Ad-Hoc Usage

Ad-hoc usage of simple services requires at least navigation to find the desired service, form generation and submission as functionality. While submission is done in the background and involves the interaction with a service, navigation and form generation involve the user and are therefore of interest to us.

### 3.1 Navigation to the Service

Navigation guides the user from the expression of a goal to the input form, which is generated automatically. After the submission of the form, the service is invoked and the output form is rendered based on the results. All of these steps bound together form an interaction pattern. For simple cases, the goal would be expressed as a direct link to the service description file as shown in Fig. 1. For more advanced cases like interacting with processes consisting of many services, like selecting a service from a registry first before using it, a more sophisticated interaction pattern needs to be defined. Interestingly, it could be derived from a formal process model, too. The relationship between navigation, form generation, submission and interaction in such an advanced case is represented in Fig. 2. It is worth mentioning that forms can either be pure input and output forms, or be of a hybrid nature, using previous input or output information to pre-fill parts of the input form.

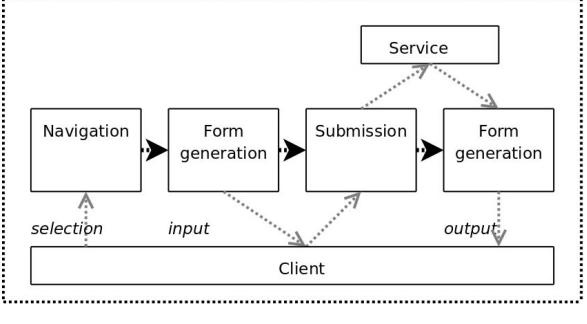


**Fig. 1.** Basic interaction model for ad-hoc service usage

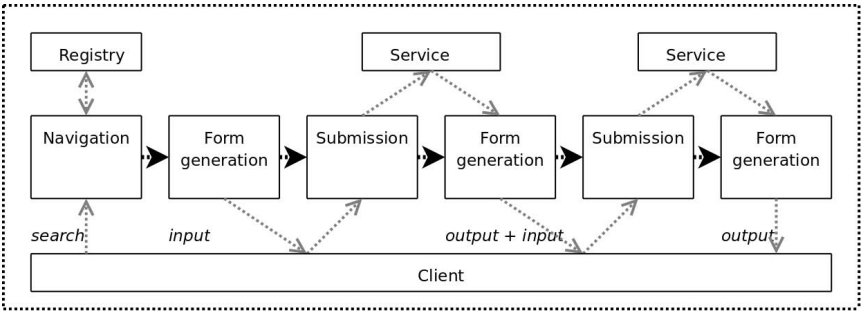


**Fig. 2.** Complex interaction including search and orchestration

### 3.2 Form Generation

Form generation is a traditional topic in the model-driven and human factors communities. Like most other approaches, we are focusing on the generation itself and do not currently evaluate usability concerns, although we acknowledge their importance for acceptance with users.

A number of individual steps have to be performed in order to achieve a suitable form. Among them are the creation of form elements, layout and embedding the form into an application context like a desktop dialogue or a website.

An additional requirement for practical use is that it should be possible to augment existing services with local hints for the graphical representation. This does not exclude an approach which integrates such hints with the service description, but allows for a greater independence from service providers.

In summary, we have identified the following technical requirements for a generic web service client:

- Ability to understand a variety of web service description formats, with or without integrated hints for graphical representation
- Ability to load external graphical, textual and semantic hints
- Ability to generate user interfaces in a variety of formats, either abstract or concrete
- Ability to define interaction models to not limit the engine to a single web service invocation
- Complete and correct visual representation of the programmatic interface

## 4 Dynvoker Approach

Following the discussion of requirements, this section is presenting the features and implementation of Dynvoker as a generic web service client. Before delving into the feature set, the overall architecture is briefly presented in Fig. 3. Dynvoker consists of a relatively small application core which can be run as a servlet, a web service or a command-line application. The generic handling of

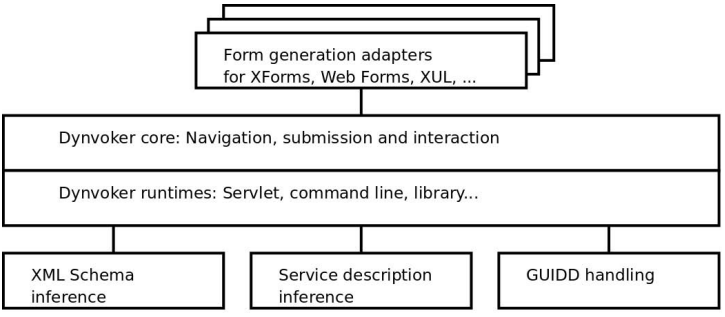


**Fig. 3.** Overall architecture of Dynvoker

input, i.e. web service descriptions, and output, i.e. user interfaces, is reflected in the modular architecture. It contains several adapters to generate forms, and inference modules for various service description formats.

### 4.1 Inference from Web Service Descriptions

In order to use web services without prior knowledge of their expected input or behaviour, it is necessary to infer this knowledge from the service description. Knowledge about the service methods, parameter names and structure can usually be derived from it. We have previously reported on details and issues of inference of user interfaces from XML Schema [8] and will therefore concentrate on the nature of inference from generic service description formats. The dominant description format is WSDL 1.1, which is used mostly for method-centric, i.e. SOAP-based services, although its successor WSDL 2.0 also contains bindings for resource-centric, i.e. REST-based services. However, alternative formats like WADL, the Web Application Description Language, exist for generic REST-based services, and even specialised formats like OpenSearch [9] for the specific domain of REST-based search services. Both WSDL and WADL use XML Schema to define the structure of the messages or resource representations, whereas OpenSearch is limited to formatted query URLs for the input and extended RDF for the output.

Dynvoker is able to infer the contents of a service, like the operations or resources it offers, from WSDL and WADL files, and will generate output which lets the user navigate to the service of choice and select the appropriate service. When a WADL-described service is chosen, the service selection interaction is extended by offering a number of resources for each method. Input and output forms are generated based on the XML Schema. The generation architecture is shown in Fig. 4.

Alternative service descriptions can be supported through transformations. OpenSearch descriptions are converted to WSDL first and can then be handled as usual without additional code. D-Bus, the dominant application-level interprocess communication (IPC) system on Linux, provides its own IDL-like method

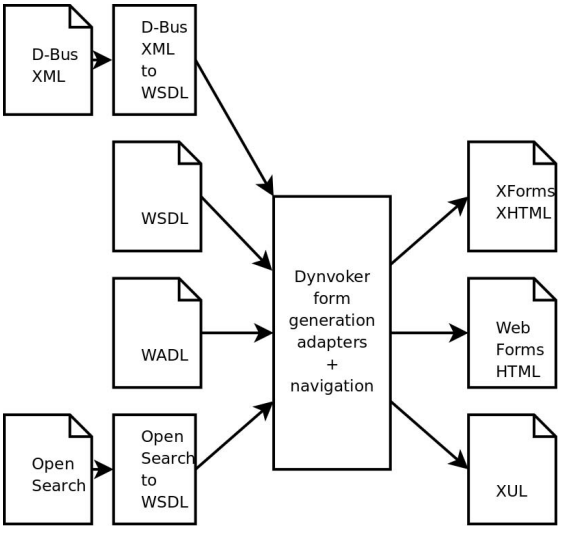


**Fig. 4.** Inference transformation process

description format which can be retrieved through service introspection. We have implemented a bidirectional gateway between web services and D-Bus, which works independently from Dynvoker, to prove our claim.[8] Since WSDL provides a superset of the service description abilities of D-Bus, the conversion always works in the direction we need for Dynvoker.

### 4.2 Additional GUI Hints

Automatically generated user interfaces are at risk of providing inferior quality and usability than manually designed ones, depending on the completeness of the information in the model or any web service description. On the other hand, a strictly rule-based design leads to consistent interfaces which can completely encompass the service functionality and automatically adapt to evolving services, including the alteration of message formats [10][11].

Therefore, as many aspects of the generation process as possible should be configurable without endangering the consistency and completeness properties. The amount of hints needed decreases with the expressiveness of the service description format. For common WSDL-described services, Dynvoker can use GUIDD files containing semantic hints, UI hints and UI services to improve the resulting forms, as shown in Fig. 5.

*Semantic hints* are useful in combination with purely syntactical description formats like WSDL to yield more appropriate user interfaces. For example, the only inferable information about the password field in Fig. 7 is that it is of type `string`. To avoid a free-form input field and use a special password entry field

[8] D-Bus Web Service Proxy: `http://techbase.kde.org/Projects/D-Bus-WS`

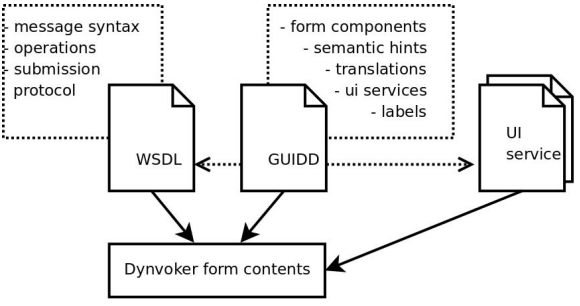


**Fig. 5.** Information sources containing additional GUI hints

instead, a semantic hint is added and will result in password fields independent of the output format.

*UI hints* include labels with translations, frame captions and substitutes for otherwise auto-generated fields, so-called form components. As opposed to semantic hints, they depend on the resulting output format. For web-based interfaces, style sheets can be used to give form components a consistent look and feel. UI hints for abstract user interfaces are also possible and are discussed in the evaluation part.

*UI services* represent a novel concept which lifts explicit GUI hints to a service-oriented level. This lifting makes it possible to exchange the hints or the providers

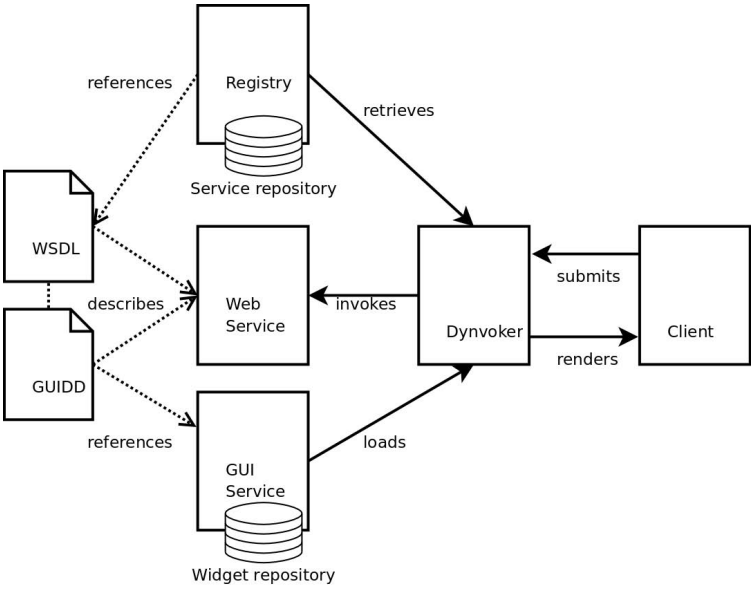


**Fig. 6.** UI services and web services in dual use

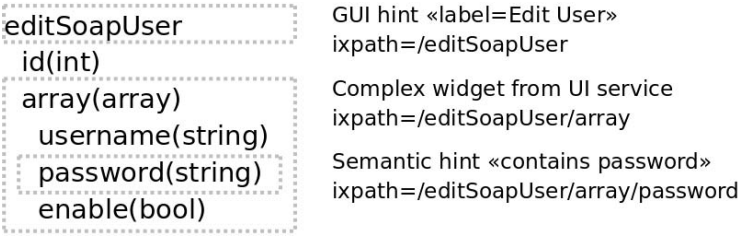


**Fig. 7.** Hint locators in a GUIDD file applied to a SOAP message instance

of the hints, therefore driving the customisation of applications. Our implementation of UI services is based on a widget repository with query interface for Dynvoker and a submission interface for UI widget designers as shown in Fig. 6. A widget connector within Dynvoker searches for available widgets and renders them into form components, aligning the further processing with UI hints. This includes a distinction between simple and complex UI hints, the latter ones covering complex types like lists.

All of these three groups of hints are stored in the already mentioned GUIDD files. If they are passed to Dynvoker, the generated forms can be improved. The reusability of GUIDD files, especially in combination with reusable data schemas in WSDL files, helps in further advancing the acceptance of SOA by eliminating redundant client development.

In Fig. 7, the locator mechanism for interleaving GUI hints for a user management operation in the SOAP API of the Asterisk telephony server is shown. GUIDD uses higher-level schema and instance XPath expressions which are aligned with the reusability of schema components. In the figure, the semantic hint for the password entry field collides with the UI service for the complex user data type `array`. The use of a GUIDD editor can help avoiding such collisions.

### 4.3 Process Integration

Up until now we have assumed the interaction between a user and a single service in our explications. This is not always sufficient in dynamic service landscapes with complex interactions between humans and processes.

We have previously proposed the WSInterConnect distributed architecture to integrate humans into processes based on interactions with Dynvoker [12]. The industry proposal BPEL4People/WS-HumanTask was already mentioned as a potential hook for this distributed architecture and has matured since then, but implementations are still not widely and freely available. Major flaws of this extension include an insufficiently specified visual representation of messages to the user and a lack of process launching interaction. A Dynvoker-based approach named Unified Process and Task Management Interface, or UPATMI, is currently being developed by us to solve this problem.

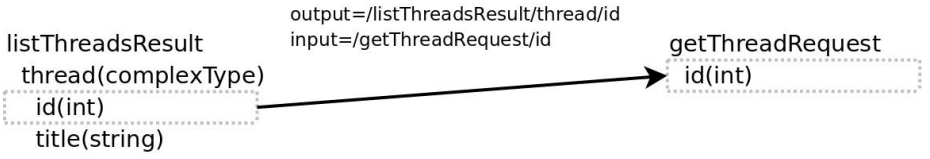


**Fig. 8.** Links between service operations

Another approach is to reduce the number of interactions needed with a process by inspecting it, essentially treating it as a grey box, whenever possible. The Dynvoker variant GUI4CWS has proven it to work for a subset of BPEL [13].

Finally, light-weight links between service calls without the need for an executed process were implemented as a GUIDD extension. This makes it possible to implement interactive applications with purely declarative syntax. For example, the list of topics in a forum as output message of the default operation `listThreads` would add a link to each thread which invokes Dynvoker with the operation `getThread` like shown in Fig. 8.

### 4.4 Status of the Resulting Implementation

Dynvoker has been developed for about two years now, entailing a number of improvements in a still ongoing process. On the other hand, it has uncovered a number of weaknesses in existing standards and implementations especially for XML Schema and XForms. This section compares the current implementation with the list of requirements, reasons about deviations and confirms the necessity of some of the assumptions we made.

Abstract user interface languages are currently not supported, but the Dynvoker architecture allows the creation of new output adapters for such languages. The resulting forms could then be displayed in applications which can render them, or convert them for display on legacy applications. This approach can also be followed with the existing XForms adapter by converting the output to HTML with JavaScript. However, according to our tests, even advanced tools like Chiba do perform this task correctly, as can be verified by anybody by selecting this transformation mode on the Dynvoker website. Therefore, we focused on writing adapters for concrete UI languages, but appreciate the potential of abstract languages.

We have not yet implemented the interaction models as executable processes within Dynvoker. All interaction patterns are currently hard-coded. We strive to add this in a future version based on GUI4CWS.

All the remaining requirements we have outlined are already supported by Dynvoker. In particular, the ability to use both resource-centric and method-centric web services contributes to hiding protocol details from the user. Additional GUI hints are supported in a way that the correctness and completeness properties from the inference mechanism will not be violated.

A large number of services with WSDL and WADL descriptions can already be used with Dynvoker. For those services for which a GUIDD exists, the user

experience is clearly better than for those without. We follow a live validation approach where any interested person can verify our results on the Dynvoker portal.[9]

## 5 Summary and Future Steps

Building up on previous detailed analysis of issues in ad-hoc service usage, we have shown that Dynvoker is a viable generic client which solves many of the issues. None of the alternative approaches can dynamically explore method-centric and resource-centric services alike, output forms in various formats or integrate GUI services to provide a richer user experience. The generic design of many parts of Dynvoker has yielded a lightweight architecture which is freely available to any interested person as an open source project.[10]

In the future, we expect to integrate even more process-related functionality and add collaboration methods to the Dynvoker portal to help building communities of users of explorable services. Furthermore, a major focus will be directed to the optimisation of UI design for complex web services, especially in the dimension of usability by solving partial aspects during a design-time stage. Its central goal is to create a model-driven service engineering methodology supported by design-time concepts and tools for the development of client applications for single and composed web services. Due to the obvious fact that some aspects such as dynamic binding of concrete services and runtime optimisation are not feasible during design-time, we aim to define a runtime platform for handling these and further runtime dynamic concerns within the ServFace project.[11]

## References

1. Sánchez-Nielsen, E., Martín-Ruiz, S., Rodríguez-Pedrianes, J.: Mobile and dynamic web services. In: Proceedings of the ECOWS 2006 Workshop on Emerging Web Services Technology, Zurich, Switzerland (December 2006)
2. Spillner, J., Braun, I., Schill, A.: Flexible human service interfaces. In: Proceedings of ICEIS. Volume HCI. International Conference on Enterprise Information Systems (ICEIS), Funchal, Madeira - Portugal, pp. 79–85 (June 2007)
3. Bajaj, A.: Inferring the User Interface from an EER Data Schema. In: Proceedings of the Americas Conference on Information Systems (AMCIS), paper 471, Acapulco, Mexico (August 2006)
4. Kassoff, M., Kato, D., Mohsin, W.: Creating GUIs for Web Services. IEEE Internet Computing 7(4), 66–73 (2003)
5. Gesser, C.E.: Uma abordagem para a integraçã dynâmica de servios web em portais. Master's thesis, Universidade Federal de Santa Catarina (2006)
6. Steele, R., Khankan, K., Dillon, T.: Mobile web service discovery and invocation through auto-generation of abstract multimodal interface. itcc 2, 35–41 (2005)

[9] Dynvoker portal: `http://dynvoker.org/`
[10] Dynvocation research project: `http://dynvocation.selfip.net/`
[11] ServFace website: `http://www.servface.eu/`

7. He, J., Yen, I.L.: Adaptive user interface generation for web services. In: Proceedings of the IEEE International Conference on e-Business Engineering, Hong Kong, China (October 2007)
8. Spillner, J., Schill, A.: Analysis on inference mechanisms for schema-driven forms generation. In: Tagungsband XML-Tage, Berlin, Germany, June 2007, pp. 113–124 (2007)
9. LeVan, R.: OpenSearch and SRU: Continuum of searching. Information Technologies and Libraries (ITAL) 25(3), 151–153 (2006)
10. Trapp, M., Schmettow, M.: Consistency in use through model based user interface development. In: The Many Faces of Consistency in Cross-Platform Design at CHI 2006, Montréal, Québec, Canada (April 2006)
11. Nichols, J., Chau, D.H., Myers, B.A.: Demonstrating the viability of automatically generated user interfaces. In: Proceedings of the SIGCHI conference on Human factors in computing systems, pp. 1283–1292 (2007)
12. Spillner, J., Braun, I., Schill, A.: WSInterconnect: Dynamic composition of web services through web services. In: Eliassen, F., Montresor, A. (eds.) DAIS 2006. LNCS, vol. 4025. Springer, Heidelberg (2006)
13. Bleyh, N.: Analyse und Vergleich von Ansätzen zur Einbindung von menschlichen Interaktionen in komplexe Web Services. Master's thesis, TU Dresden (June 2006)