FLEXIBLE HUMAN SERVICE INTERFACES

Josef Spillner, Iris Braun, Alexander Schill
Chair for Computer Networks, TU Dresden, 01062 Dresden, Germany
spillner@rn.inf.tu-dresden.de, {iris.braun,alexander.schill}@tu-dresden.de

Keywords: WSGUI, Web Services, GUI generation, MDA techniques

Abstract: Dynamic web service invocation without special client software may help the adoption of service-oriented architectures on the consumer stage. Ad-hoc usage of services requires a powerful set of concepts to visualise the service input and output messages in a user-friendly, ergonomic and extensible way. Such concepts are collected in a research effort named Web Service Graphical User Interface and are presented in the paper in combination with an algorithm to combine the concepts into one imaginary GUI creation engine, for which a proof-of-concept implementation exists. Extensibility is achieved by using implicit and explicit GUI generation hints in addition to inference mechanisms based on the message structures.

1 MOTIVATION

WSGUI, or *Web Services Graphical User Interface*, describes concepts on how to enrich data model schema information in the context of web services so that it becomes usable for humans in interactive scenarios. The core idea of WSGUI is that dialogs for input and output of structured parameters can be generated dynamically. To achieve usable and appealing GUIs, the formal message specification will almost never suffice, hence there is a need for some additional data on how the GUI should be created. A collection of concepts is presented which is believed to achieve good overall quality without too much effort in the creation of the additional data. There might be other combinations of concepts, unknown as of yet, but the ones presented here seem to be a good choice given their properties: First, they follow the paradigm of model-driven GUI design, making them suitable for changes in the message parameters where the message schema will take precedence over the GUI annotations. Second, they can be used as needed, with few interdependencies and the ability to incrementally improve the generated GUIs without having to interfere with the service functionality.

The term GUI shall be used in a wider sense and will in most cases refer to generated dialogs or masks for input and output of data.

The nature of web services in practice is that they're often described only syntactically, using WSDL files which include schema information to describe the messages. This requires a mechanism to make up for areas where the description lacks by means of additional GUI generation hints.

WSGUI can therefore be thought of as a family of different concepts which, when combined properly, can lead to decent GUIs which are generated automatically while still preserving an appealing look and feel. Platform independence, integration with the surrounding environment and consistency are further advantages of employing WSGUI concepts in applications.

The order of the sections of this paper is such that the concepts are presented first, followed by some information on how they might interdepend on each other. The introduction is complemented by a report on progress since the initial presentation of WSGUI. An implementation project and some related work is presented afterwards.

2 WSGUI CONCEPTS

Concepts are introduced here as abstract steps which are part of the transformation process from the web service message structure to the GUI. All concepts are realised through specific algorithms and filters as described later. The software performing these steps is named *WSGUI engine*, independent of how or where it is actually implemented. For the GUI, any declarative format can be the target of the transformation process. WSGUI does not define a new abstract UI description.

A number of concepts were present in the original WSGUI proposal (Kassoff et al., 2003). Some of them were found not to be needed anymore, while at least two were found to be crucial so they're included in the following list. The others are the result of recent work performed in this area by the authors. Reasons for why to discontinue some of the concepts will be given later.

The two mentioned original concepts are **web service description** and **form components**, and can be used by software implementations by reading in GUIDD files, that is, *GUI Deployment Descriptors* (Kassoff et al., 2006). New concepts added by the authors include a strong **inference** mechanism, **MIME awareness**, the **user-toggle** concept and **templates**.

Each of these concepts helps in shaping a better GUI while still preserving the aforementioned properties. All of them are going to be presented in this section.

2.1 Web Service Description

While most of the WSGUI concepts deal with individual service messages, being either input or output of an operation, some of them deal with the global scope. Navigating to the message requires presentation to the user, too. This includes the selection of the service and operation and passing this information to the user. Such user interface elements include the internationalised representations of name, short description and comprehensive help of both the services and their operations, and also several details such as the caption of the submission button for input forms.

While WSDL files can contain information such as help texts, service providers usually do not translate these texts. Especially for WSDL files generated automatically by reflection from object code or parsing of source code, it becomes necessary to supply external texts to overcome this limitation. The GUIDD file format provides the `operations` section for the task of including an arbitrary number of translated human-readable texts.

2.2 Inference

Dynamic invocation of web services without any additional GUI generation hints can still result in different user experience, depending on how close the GUI represents the message structures.

Instead of just providing simple form controls, say string input fields, for all schema elements, a dynamic invocation engine can already infer that some data types require special treatment. For example, if a type is restricted by an enumeration of values, only those values should be displayed in a combo box, instead of letting the user input arbitrary values. Likewise, an element which may occur *n* times as a child of its parent element can be displayed as a list which may be enlarged or shrunken within the occurrence constraints.

In addition, the XML Schema embedded into WSDL files might provide valuable initial help texts for each form control through its annotations. For the captions of the controls, only the parameter name can be used, which will not be suitable for human display unless they adhere to certain conventions (Steele et al., 2005).

A still largely unexplored area is the coupling of data types to semantics to generate widgets such as password edits or multiline edits. None of them can be inferred automatically. They are all based on the string datatype as is the generic input widget, therefore form components are required in the meantime to force their usage. Semantic annotations provide implicit GUI hints, whereas form components are an explicit concept. How to make use of semantic ontologies and how to address usability concerns is out of scope for this paper and dealt with in other publications (Khushraj and Lassila, 2005).

2.3 Form Components

The main part of the GUIDD file format is about specifying abstract form components, each of which is wrapping a specific GUI control, which can be used to override decisions otherwise being made by the GUI control inference mechanism. The GUI control family currently proposed for GUIDD is XForms (Boyer et al., 2006), which does however not imply any usage thereof in the final generated GUI.

Based on the message schema, both model and instance addressing can be used to locate the positioning of the controls, with the latter one overriding the former since complex data types can be in use multiple times. The need for instance addressing, which was not present in the original GUIDD specification, stems from the fact that due to type definitions being either named or anonymous, there can be syntac-

tical ambiguities in the model which however lead to no difference in the instance. For example, schema elements might appear as anonymous child elements in the schema XML, or they might appear as named reusable types, but the instance XML will be the same in both cases. Both addressing schemes use higher-level XPath locators.

In case the instance data is restricted by an enumeration and this enumeration is likely to change or even created dynamically, the concept of XForms filters has been introduced. This replaces the previously suggested method of dynamic enumerations, and is also useful for handling translations. The GUIDD specification (Kassoff et al., 2006) contains advice on how to handle filters. In the case of enumerations, it ensures that instance data not yet covered by a *select item* in the form component's control gets added to the control automatically, while in the case of translations, all translations which are not requested from the engine will be dropped before delivery of the document to the client.

2.4 MIME Awareness

Especially for output formatting, when receiving raw data in *base64* or hexadecimal encoding, a MIME type can be specified to automate the display or other further use of the data. Since this is only part of WSDL on a message part level, but not available per schema element, WSGUI needs to provide a way to enhance the schema with this information. Some MIME types might resolve to text effectively, for example `application/pgp-signature`, but others might lead to output format specific areas, such as `img` for `image/*` in a web context.

Recently, the proposed XForms 1.1 specification gained support for describing rich output formats for its `output` control. Eventually, MIME awareness might thus be realised using GUIDD's form components sections. However, to maintain backwards compatibility, there is the OutputTypes section where MIME types can explicitly be assigned to output form controls.

2.5 User-toggle Concept

Since for some non-determinable widgets such as the ones outlined above (password edit and multiline edit) no automatic choice can be done, the inference mechanism would always return a generic input control. In such a case, unless a form component is available, the user should have the last word in these cases and be able to override the inference mechanism's choice. This can be achieved by displaying a widget type-toggling button beside the widget so that it can change its type to whatever alternative might be available, and does so by the user's choice.

2.6 Templates

Form generation results in free-standing forms, contrary to the practical need of embedding such forms into host documents such as web pages with custom surrounding graphics. Templates make it possible to define template pages with defined positions for the generated forms, and let the WSGUI engine output the filled-in template instead of only a free-standing form.

This concept is a web-specific extension to the GUIDD format, as its usage is limited to web environments. However, experimental work has led to good form integration with WSGUI templates and therefore environment-specific GUIDD profiles will likely result from this work.

3 CONCEPT INTERDEPENDENCIES

Now that the existing WSGUI concepts are determined, an algorithm is presented which is considered a good compromise between automatic GUI generation and user-designed UIs. Depending on the efforts for the creation of the GUIDD file, the result can be either a generic and visually clumsy user interface or a highly unique service-specific UI, and in most cases will be in between.

3.1 Form Controls Generation

The proposed algorithm consists of a single recursive function, `render()`, which takes as its parameter a schema element, starting with the top-level element of the complex schema tree. For each schema parameter, the best choice is selected for its representation in the user interface, and the chosen UI element is stored into a list as associated with the parameter. For simplicity reasons, schema elements and attributes are both treated as elements in the algorithm description.

```
FUNCTION render(element):
  IF is-complex(element)
    FOREACH child-element OF element
      render(child-element)
  ELSE
    IF form-component-found(element)
      filter-enumeration()
      filter-translation()
```

```
    ui = apply-form-component()
  ELSE
    IF mime-info-found(element)
      ui = mime-widget()
    ELSE
      filter-documentation()
      IF inference-possible(element)
        ui = inferred-control()
      ELSE
        ui = generic-control()
        user-toggle()
store(child-element, ui)
```

The algorithm describes how to iterate over the whole schema, recursively in the case of complex elements which have child elements, to find out all elements. For each of them, it is decided whether to use explicit GUI information in the form of **form components** if they are available, or to apply implicit hints such as **MIME awareness** or possibly other semantic annotations. If a form component can be located, the contained form control must pass some filters, as is implied by the form components concept. If neither explicit nor implicit information is available, **inference** is the fall-back mechanism, which in some cases needs to add a **user-toggle** button to the side of a control.

Most concepts are present in this algorithm. The remaining ones deal with web service-specific information such as operations and are handled outside the scope of the creation of the form controls themselves. The algorithm should instead be part of a pipeline which lays out the list of form controls according to how the GUI should eventually look like.

3.2 Pipelining the Concepts

While the presented algorithm transforms message parameters into form controls, it barely covers layout and geometry settings, that is, how to place the controls onto the resulting GUI. The only way of inferring those settings is to either build up a tree-like view, with complex data types being containers for simple types being represented as controls, or to use generic schema-based GUI element positioning algorithms as explored in (Kruschinski, 1999).

Since there is no predominant way of specifying layouts, WSGUI does currently not include anything else but an implementation-dependent tree layout convention. The tree-like structure of the messages the forms are based on suggests that this is an acceptable resolution.

The usage of explicit layout and geometry information could well be seen as a second transformation step in a pipeline, with the algorithm presented above being the first step. A third (and last) step would be to integrate resulting forms into host documents, as described in the WSGUI **templates** concept entry. Figure 1 describes how the pipeline works. It is noteworthy that not only the schema, but also related context information such as operation-specific information (name and submission URL) and the position in a complex process, if any, are used to determine the generated GUI.

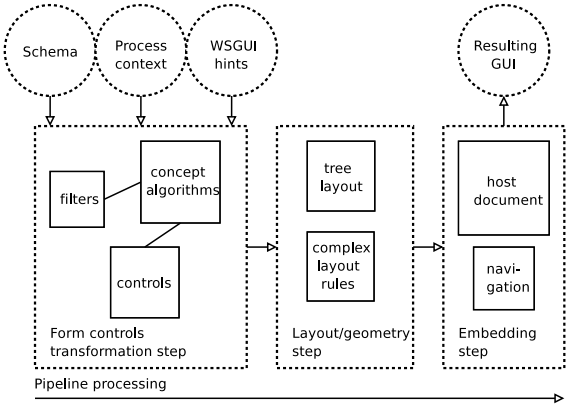

Figure 1: WSGUI engine as a pipeline

4 COMPARISON TO THE ORIGINAL WSGUI CONCEPTS

In 2003, the first paper on WSGUI was published (Kassoff et al., 2003), which outlined the basic ideas present in current-day WSGUI research. However, with other standards progressing fast, in particular those in the area of XML, it was time to reconsider the ideas of the paper and re-evaluate them for practical usage. The ideas will be listed, described and evaluated in the forthcoming paragraphs.

Form components as specified in a GUIDD file are still considered a necessary WSGUI concept. Several subtle differences to the original idea exist, such as using XForms controls directly (apart from filtering) as opposed to merely borrowing from the XForms syntax. The central idea of form components is explicit assignment of visual controls to parts of the data model. The reliance on XForms is discussed in the section about the form components concept and helps keeping WSGUI lean.

Form navigation is an interesting concept which can morph the UI around a selection between di-

rect editing, item selection and constraint-based query views. The intended implementation looks doable, nevertheless this feature is not included in the current research since it requires additional work from the authors of GUIDD files. The new concept of **user-toggle** is a similar one, which stresses the choice on the part of the user.

Variable-sized lists is definitely needed for web services which contain schema elements of varying quantities. However, it was found that WSGUI engines can automatically generate GUIs which take care of adding and removing controls according to the allowed range of quantity. For example, XForms provides the *xf:repeat* module which eliminates the need to copy the functionality in the GUIDD file.

Virtual operations are best implemented within processes, and delivered transparently to the client as part of a WSDL file by a process engine. Standards such as BPEL, the *Business Process Execution Language*, make it possible to build such virtual operations without the direct need for WSGUI concepts, since the elements of a GUIDD file can refer to such operations as well.

Dynamic enumeration refers to special web service method which return a list of all possible values of a variable, for display in a non-editable drop-down box or a similar widget. Value constraints are better specified explicitly in a schema file as part of a simple type enumeration restriction. As before, GUIDD form components can map to these lists of values, but no special requirement is needed for the service. XForms filters ensure that if GUIDD and the enumeration instance data go out of sync, the resulting GUI does still display formally correct choices to the user. Therefore, the current GUIDD files do not need the corresponding XML descriptions anymore, and web services do not need to be enhanced with helper methods anymore.

Look-and-feel stylesheets The task of the L&F stylesheets is to convert an abstract user interface to a concrete one. The wide adoption of abstract GUI description languages such as XForms and our implementation work has led us to believe that no such stylesheets are needed per se as part of GUIDD.

To conclude this section, a number of concepts could be saved, partially thanks to alternative approaches leading to similar or better user experience, and at the same time leading to less work for both the GUIDD authors and the service providers.

5 RESEARCH AND IMPLEMENTATION PROJECTS

The failure of adoption of previous efforts similar to WSGUI led to the conclusion that file formats like GUIDD would only be deployed if sufficient support for it is available from software developers. The authors of the paper will present their implementation in this section, but also refer to others which share some of the ideas.

5.1 Project Dynvocation

Most of the WSGUI concepts described above are being implemented as part of *Project Dynvocation* (Spillner et al., 2006). The Dynvoker servlet generates input forms and result pages on the fly from web services, using inference as heavily as possible and applying GUIDD hints wherever they are available.

Most of the implementation work was aimed at creating a servlet which produces XHTML pages with embedded XForms. Thanks to the use of XForms in GUIDD form components, the XForms controls found in the GUIDD form components can be used natively after the XForms filters have been applied. As opposed to its predecessor, HTML forms, XForms offers schema binding and thus also type safety and separation of data from its presentation.

Its focus on inference makes it possible to invoke web services without any additional information, and already provides type safety and flexibility in terms of changing list sizes and user-toggle actions. Adding a GUIDD will however greatly enhance the user experience. An editor for the creation of GUIDD files has been developed as part of a thesis work of one of the authors recently. It reuses some of the WSGUI concepts to make the authoring process as simple as possible. The decoupling of a web service description file (the contract) and GUI hints (the presentation layer) greatly helps in authoring and customising the latter.

However, Dynvoker is not limited to producing XHTML pages for the interaction with simple web services. A few extensions will be described now.

5.1.1 Existing Extensions to Dynvoker

The Dynvoker software has been extended to handle complex service interaction scenarios like BPEL-composed services. The work is called GUI4CWS (*GUI for Complex Web Services*) (Bleyh, 2006) and uses a subset of BPEL in combination with WSGUI concepts to achieve goal-oriented workflows.

Beside XHTML output, a way to create XSWT dialogs for the Eclipse platform has also been worked on. While the XHTML generation works mostly programmatically, the Eclipse dialog generation is performed by a XSL transformation. However, not all corner cases could be handled to full satisfaction. Other projects (Moebius, 2006) also try to generate GUIs from XML Schema or WSDL files purely by applying a XML transformation and similarly leave out the more complicated transformations. It remains to be seen to which extent the logic behind the GUI generation can be represented in a declarative transformation document if the goal was to implement all WSGUI concepts.

5.1.2 Details of the GUI Generation

Before a form is rendered by Dynvoker, the user is guided to select a web service and an operation. All of the navigational user interface elements are derived from a navigation schema as well, thus re-using Dynvoker's ability to generate dialogs for several target platforms.

Hence, Dynvoker presents itself to the user with an interface which is automatically generated from an XML Schema file and the corresponding GUIDD file. It allows the user to select a service by entering the URL of a WSDL file, possibly in combination with a GUIDD file in addition, and some option buttons which will influence the generation of the GUI, mostly to be able to see the difference between using and not using any of the mentioned WSGUI concepts. If no GUIDD file is specified, the whole GUI generation process relies on inference mechanisms based on the information in the WSDL file.

After submitting the service selection form, the GUI for the input message of the WSDL file is rendered, and the user again fills out all of the fields. Another submission action now invokes the service using Dynvoker as a proxy, and returns the result to Dynvoker. The results are then rendered as an output form which is sent to the user.

Architecturally (figure 2), a DynvokerEntrance object receives the parameters of the call, providing an abstraction layer between the Dynvoker implementation as a servlet, portlet or library on one side and the remaining parts on the other side. DynvokerCore as a state machine then takes over the parameters and dispatches the call as needed. In most cases, an XFormsAdapter object will be called which implements the first step of the WSGUI pipeline. This includes assigning form components to the individual parameters of the currently selected web service message, calling XFormsFilter whenever translations need to be eliminated from the form controls

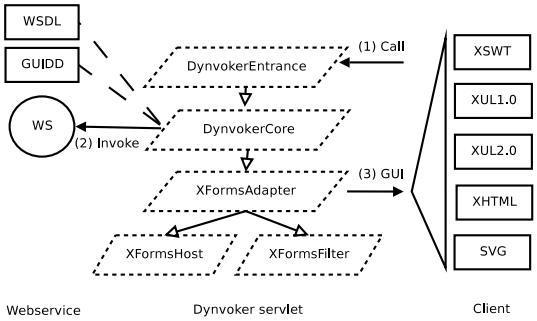

Figure 2: Dynvoker architecture

or enumerations are transformed. To render a suitable XHTML page which is to include the generated form or to include the form into an existing template page, XFormsHost can be called. This object also handles the placement of the form controls into a tree layout.

5.1.3 XForms Filters

As mentioned before, form controls are not easily suitable for dynamically generated list datatypes, for which only the control's name and help could be given deterministically, while the values might change. When they don't change too often or some of them are present in most invocations, XForms filtering comes to the rescue. The graphics 3 shows how a partially applicable form control definition is layered on top of the corresponding auto-generated form control.

XPath expressions are used to match the related XML nodes of both controls. Similar to how form controls themselves are referenced using XPath expressions, name spaces are preserved and handled correctly, making it a suitable mechanism even for complex schemas.

The GUIDD extension proposed in the GUI4CWS document for handling list types by introducing a placeholder `<items/>` tag is not necessary anymore thanks to the filtering concept.

Filters are also used in Dynvoker to hide all translations which are not requested by the client in labels, help texts and hint texts. Additionally, they're also used to use existing XML Schema annotations as help texts if no help text has been given for a control through the GUIDD file.

5.2 Other WSGUI-related Projects

A project similar in its goals is the XML Forms Generator (Kelly et al., 2006), which is part of the Eclipse framework. It already supports external translation

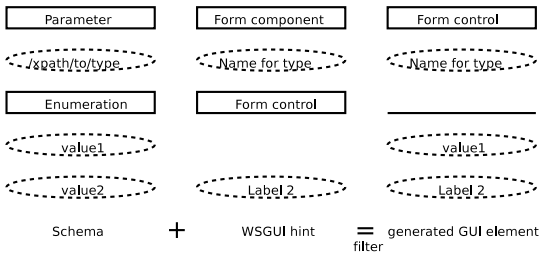

Figure 3: XForms filters

texts, which matches the GUIDD facility of form components, but also deals with rendering options. Its focus on web services would make it an ideal candidate for using further WSGUI concepts so that those issues found in real-world web services and GUI generation of their messages could be overcome.

An architecture for discovery and dynamic invocation of services has been presented in (Steele et al., 2005). It focuses on multimodal access more than on in-depth discussion of GUI generation for visual access, which is still the most important kind of interaction between services and humans. Building a service discovery model on voice recognition is another idea present in this work, which will further help in ad-hoc service usage. The presented way of generating the GUI could profit from employing WSGUI concepts, as the ambiguity of WSDL parameter naming is admitted openly in this work. The proposed solution, to adhere to naming conventions for WSDL message variables, will hardly address internationalisation, as opposed to WSGUI **form components**.

Departing from the initial focus on web services, WSGUI concepts are already used in other areas such as configuration dialogs and printer dialogs, which are generated automatically from the underlying configuration schema (Schumacher, 2005). The number of shared concepts is high enough to warrant a broader scope of WSGUI, separating the web service-specific concepts from the generic ones. The former are centred around web service descriptions such as the common WSDL (*Web Service Description Language*), while the latter are focusing on the generation of forms from the message, its parts and their data types.

6 CONCLUSION

Interfaces between web services and humans are clearly on the agenda of researchers and influence future service development. The WSGUI concepts collectively presented in this paper, while not completely eliminating the need for custom interfaces to web services, are powerful enough to make the dynamic invocation of web services feasible. The conditional usage of implicit and explicit GUI hints are of great help in separating the work of web service developers and interface designers.

The Dynvoker implementation handles complex datatypes and namespaces within the XML Schema part of WSDL files, unlike many other implementations which are only suitable for relatively simple services. Its output in the XForms format ensures type-safe data submission. Finally, the rigorous use of WSGUI concepts including several best-effort strategies when no implicit or explicit GUI generation hints are provided lead to usable GUIs without much effort. Implicit semantic-driven GUI hints will be one of the main topics of research in the near future.

REFERENCES

Bleyh, N. (2006). Analyse und Vergleich von Ansätzen zur Einbindung von menschlichen Interaktionen in komplexe Web Services. Master's thesis, TU Dresden.

Boyer, J. M., Landwehr, D., Merrick, R., Raman, T., et al. (2006). *XForms 1.0 (W3C Recommendation)*, 2nd edition. `http://www.w3.org/TR/xforms/`.

Kassoff, M., Kato, D., and Mohsin, W. (2003). Creating GUIs for Web Services. *IEEE Internet Computing*, 7(4):66–73.

Kassoff, M., Spillner, J., et al. (2006). *GUIDD: GUI Deployment Descriptor file format specification*, 0.99.5 edition.

Kelly, K. E. et al. (2006). XML Forms Generator. IBM alphaWorks.

Khushraj, D. and Lassila, O. (2005). Ontological approach to generating personalized user interfaces for web services. *Lecture Notes in Computer Science*, 3729:916–927.

Kruschinski, V. (1999). *Layoutgestaltung grafischer Benutzungsoberflächen*. PhD thesis, Ruhruniversität Bochum.

Moebius, J. (2006). xsdtransformer. `http://xsdtrans.sourceforge.net/#xsd2fx`.

Schumacher, C. (2005). KXforms - automatic configuration dialog generation. aKademy conference.

Spillner, J., Bleyh, N., et al. (2006). Project Dynvocation. `http://dynvocation.selfip.net/`.

Steele, R., Khankan, K., and Dillon, T. (2005). Mobile web service discovery and invocation through auto-generation of abstract multimodal interface. *itcc*, 2:35–41.