

Construction of Generic Web-Based User Interfaces

Arno Puder

San Francisco State University
1600 Holloway Avenue
San Francisco, CA 94132
arno@sfsu.edu

Abstract. Several middleware technologies exist today that facilitate the development of applications in distributed heterogeneous environments. Interoperability is the property by which an application can span different middleware technologies. Much attention has been given to the application-to-application interoperability. With the rise of web services, we introduce a generic model for user-to-application interoperability. At the core of this concept is a mapping from user-friendly web forms to arbitrary middleware technologies.

1 Motivation

A middleware seeks to facilitate the development of applications in distributed, heterogeneous environments. Several predominant middleware technologies exist today, such as CORBA, EJB or Web Services. As applications often span different technological and administrative domains, it is not uncommon that one application is deployed on several different middleware platforms.

Interoperability defines the property by which different middleware technologies connect to each other. Without interoperability, an application could not span different middleware technologies. Much attention has been paid to interoperability for the predominant middleware technologies. Today there exist solutions to connect the major middleware platforms.

We view the technologies around web applications as another middleware platform. The web browser serves as a generic user interface. Applications are written as Common Gateway Interface (CGI) scripts that run on the side of the web server and communicate with the web browser via HTML over HTTP. Several technologies exist that facilitate the development of web applications such as Java Server Pages (JSP) or Active Server Pages (ASP). While highly flexible, programs written in this technology often merely serve as a bridge between the web browser and some backend service such as a middleware or a database.

In this paper we introduce a model that allows the automatic creation of web-based user interfaces and that does not require any coding on the side of the web server. We describe a generic bridge that can be used to create user-friendly interfaces for arbitrary backend services without having to implement

a custom interface. The interface is derived automatically from the interface specification of the backend service.

In section 2 we take a closer look at interoperability in the context of user-to-application interoperability. Section 3 introduces the architecture of our generic bridge. Some implementation details are highlighted in section 4 while section 5 discusses related work. Finally, section 6 provides a conclusion and outlook.

2 Interoperability

Development of distributed applications is supported by a middleware. The term “middleware” derives from the fact that it is located between the operating system and the application and provides a level of abstraction for distributed applications. One way to think of it is that a middleware platform spreads out like a table-cloth in a heterogeneous environment, hiding different technologies beneath it. The same API is offered at the various access points throughout the network.

Unfortunately it is impossible to impose the same technology everywhere and it can not be avoided that different middleware technologies dominate in different domains. The dashed line in Figure 1 defines the interface between two different technological domains. By standardizing this interface, one achieves *interoperability* of applications thereby decoupling technological domains.

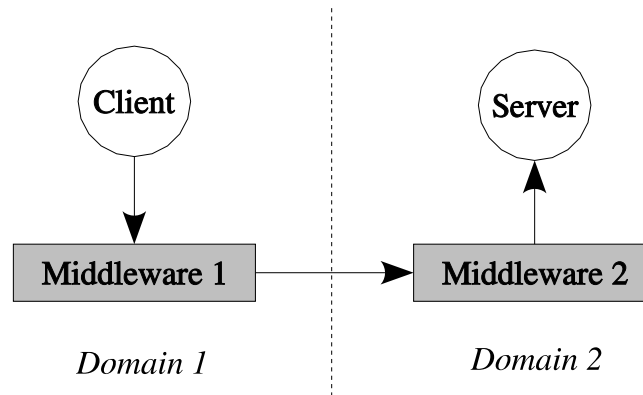


Fig. 1. Interoperability between different domains.

The interface between different middleware technologies is usually defined through a network protocol detailing how the two platforms talk to each other. Interoperability of two domains can only be achieved if the object invocation semantics and type systems that are used to describe object interfaces can be mapped onto each other. Sometimes this leads to an extension of the type system

of one technology. E.g., in order to achieve interoperability between CORBA and EJB, the CORBA type system had to be augmented by value-type-semantics (a.k.a. objects-by-value). Another way of overcoming differences is to introduce a bridge that implements a custom mapping.

Middleware	Interoperability
DCOM	Proprietary, DCOM specific protocol.
EJB	Uses Java specific RMI or optional CORBA-IIOP for the transport layer
SOAP	XML-based marshalling
CORBA	Defined through GIOP/IIOP
Web	HTML over HTTP/CGI

Table 1. Comparison of different middleware technologies

Table 1 summarizes the characteristics of various middleware technologies in use today with respect to the way they define interoperability. We view the infrastructure for web applications as another middleware technology. As applications get increasingly end-to-end, reaching from the user interface to some backend system, we focus our attention to interoperability with web applications.

Web applications are typically implemented as Common Gateway Interface (CGI) scripts that get invoked by the web server. These scripts process any user input and act as a bridge to some backend service. Common techniques for implementing these CGI scripts are Java Server Pages (JSP) or Active Server Pages (ASP) that embed a programming language inside an HTML-page.

For many applications, the CGI scripts merely act as a bridge to a backend service. Data validation, implementation business logic and database access are typically implemented on a backend system and the CGI script only passes user input to this backend and renders an HTML page as a result.

In this paper we introduce a generic bridge that is implemented as a CGI script. The bridge automatically creates user-friendly user interfaces based on the operational interface of the backend system. This approach is particularly feasible for course-grained, loosely coupled systems, because there is a natural mapping between the operational interface and a user interface. This is particularly useful for web services which connect loosely coupled systems.

The advantage of this approach is that no programming is required to build a user interface. This makes it easy to deploy new services quickly and allow access through a regular web browser. The downside of our approach is that the look-and-feel of the user interface is determined by the generic bridge and sometimes it still is preferable to build a custom user interface.

3 Architecture

Figure 2 shows the overall architecture of the generic bridge. The bridge runs as a CGI-script at the side of the web server. A general purpose connector allows

it to access different backend technologies, such as SMTP, SQL, CORBA, and Web Services. As will be shown later, the generic bridge can be given “hints” expressed in XML on how to render the user interface. The following sections give a detailed overview of the various components of the generic bridge.

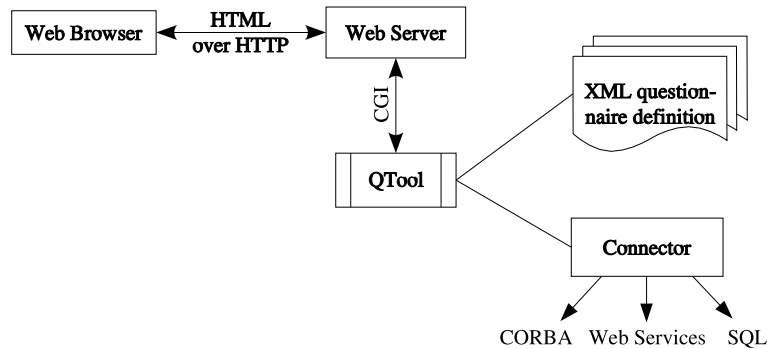


Fig. 2. Architecture.

3.1 User interface model

This section introduces a general user interface model. The model defines the basic building blocks that can be used to define a user interface. While there will certainly be special cases where this user interface model is not powerful enough, we believe that it can be used in a variety of different scenarios. A later section will give an overview where the generic bridge has been successfully used.

The heart of the user interface model is the notion of a *questionnaire*. Abstractly, a questionnaire is a user-friendly visualization of an operational interface. A questionnaire is composed of one or more *sections* that allow the logical grouping of questions. Sections can be nested (a section within a section) and can furthermore be marked as repeatable. For repeatable sections, the user can input more than one instances of this section.

Each section contains one or more *questions*. A question prompts the user for one specific feature. It is rendered in the web browser as an input field. Each question has an associated *question type* that determines the kind of response that is expected from the user. Examples of question types are string, textfields, external documents, etc. Table 2 gives a list of all the question types supported by the generic bridge. Each question type is rendered as a specific HTML element such as a drop-down list or a checkbox.

All components of the general user interface model (questionnaire, section, question) can furthermore contain documentation that will be included in the rendering of the questionnaire. In summary, the general user interface consists of the following components:

Type	Description
String	One line text response
Textfield	Several lines text response
Email	Email address
URL	Web-URL
Bool	Yes/No response
Checkbox	Yes/No response as a checkbox
Feature	Possible values: Yes/No/Unknown/Planned
Ranking	Possible values: 1-10, N/A
Document	External document

Table 2. Question types

Questionnaire:

- Documentation
- List of Sections

Section:

- Documentation
- Can be marked as repeatable
- Contains one or more questions
- Sections can be nested

Question:

- Documentation
- Can be marked as mandatory
- Has a type (see Table 2)
- Question text

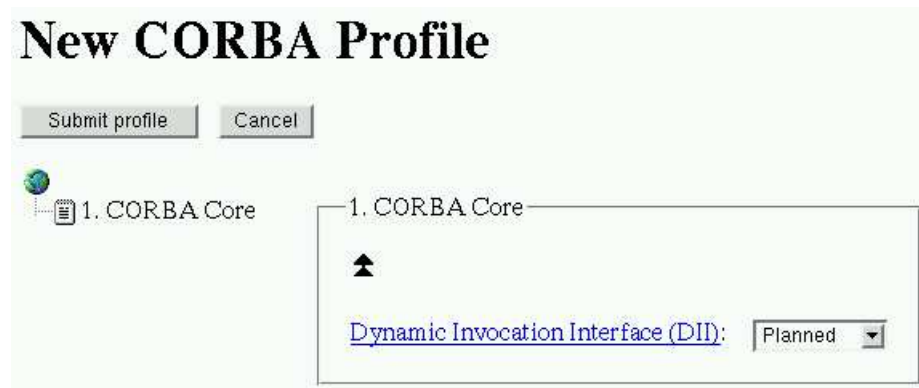
The questionnaire is defined through an XML document. Each of the components of the questionnaire (sections and questions) are represented by appropriate XML-tags. In that sense, the generic bridge defines a new XML-application by providing the “vocabulary” to express the questionnaire. The following shows an excerpt of a question definition:

```
<question name="DII" type="feature" mandatory="no">
  <doc>
    The Dynamic Invocation Interface (DII) is part of the
    client side API. With the help of the DII, a client can
    construct method invocation at runtime without the need
    for an IDL-generated stub.
  </doc>
  <query>
    Dynamic Invocation Interface (DII)
  </query>
</question>
```

The question above was taken out of a questionnaire for CORBA products. This particular question asks about the availability of the Dynamic Invocation

Interface (DII) of a given CORBA product. The question type “feature” has the following four possible values: yes (DII is supported), no (DII is not supported), planned (vendor plans to implement the DII) and unknown (it is unknown whether the DII is supported). The question type “feature” is useful for capturing the features of a product.

Based on the specification of the question a HTML-based user interface is automatically generated (see screenshot depicted in Figure 3). The documentation of the question is displayed in a popup window when clicking on the hyperlink of the question. The question type feature is rendered as a selection box (the user has selected “Planned” in the screenshot). The other elements of the screenshot displayed in Figure 3 such as the title “CORBA” or the section title “CORBA Core” are also defined in the questionnaire and are described by appropriate XML tags.



The screenshot shows a web form titled "New CORBA Profile". At the top, there are two buttons: "Submit profile" and "Cancel". Below the buttons, there is a tree view with a single item "1. CORBA Core" selected. To the right of the tree view, a popup window is displayed, showing the selected item "1. CORBA Core" with an upward-pointing arrow. Below the arrow, there is a label "Dynamic Invocation Interface (DII):" followed by a dropdown menu with the value "Planned" selected.

Fig. 3. HTML-based interface for the DII question.

Since a questionnaire is based on an XML-document, it is possible to describe its syntax through an XML schema. This schema is itself an XML document, but it describes the legal instances of a questionnaire. The following excerpt shows the XML schema definition of a question. Note that the above example of the DII question complies with the syntax defined here:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

```
[...]
```

```
<xsd:complexType name="QuestionType">  
  <xsd:sequence>  
    <xsd:element name="doc" type="xsd:string"/>  
    <xsd:element name="query" type="xsd:string"/>
```

```

    </xsd:sequence>
    <xsd:attribute name="name" type="xsd:string"/>
    <xsd:attribute name="type" type="xsd:string"/>
    <xsd:attribute name="mandatory" use="optional" type="YesNoType"/>
</xsd:complexType>

[...]

</xsd:schema>

```

The above definition expresses that a question consists of a documentation field and a query field. A question must have a unique name and type which are specified as XML attributes. Furthermore, a question can be marked as mandatory or optional. With the former the user is required to provide input for that particular question.

3.2 Meta-questionnaire

In the previous section it was said that the schema of a questionnaire has to be defined through an XML document. This XML document contains all the information that are needed by the generic bridge to render a user-interface. While this approach is very flexible, it requires the end user to provide an XML document for each new questionnaire.

Since the emphasis of the generic bridge is end-user friendliness, it is not acceptable to expect knowledge of XML. For this reason, a web-based management interface is provided that allows the definition and maintainance of questionnaires.

To define a new questionnaire, the user has to fill out a special questionnaire whose purpose is to ask for all the elements of the new questionnaire to be defined. One important insight is that this “meta-questionnaire” is yet another questionnaire. I.e., in order to define a new questionnaire, the user has to fill out a special questionnaire. The special, or meta-questionnaire, contains all the questions that need to be answered in order to define a new questionnaire.

The screenshot depicted in Figure 4 shows the rendered HTML interface of the meta-questionnaire for the DII question introduced in the previous section. First notice that the look-and-feel of the user interface is similar to that of the DII question depicted in Figure 3. The various questions displayed in the screenshot mirror the information that was specified in the XML document of the previous section. This shows that new questionnaires can be defined using the meta-questionnaire and without any knowledge of XML.

Using a meta-questionnaire has two main benefits. The first is that the user is alleviated from having to have any knowledge of XML to specify new questionnaires. Instead the user can use the familiar user interface that queries everything the generic bridge needs to know about the new questionnaire. The second benefit it that the implementation of the bridge is greatly simplified. Instead of having a separate management interface, questionnaires are managed through the familiar user interface itself. The same code can be re-used.

2.2. Question

Question text: (*) Dynamic Invocation Interface (I

Question type: feature

Question mandatory: No

Question documentation: The Dynamic Invocation Interface (DII) is part of the client side API. With the help of the DII, a client can construct method invocation at runtime without the need for an IDL-generated stub.

Fig. 4. Meta-questionnaire.

3.3 Connectors

So far we have discussed two different ways of defining a questionnaire: through a XML document and via the meta-questionnaire. Another way to create questionnaires is to derive them from operational interfaces such as CORBA-IDL or WSDL (Web Service Definition Language). A connector translates between questionnaires and such operational interfaces. The translation requires mapping rules. This mapping is bidirectional: questionnaires can be mapped to operational interfaces and vice versa. In the following we focus on the former. I.e., we assume the existence of a questionnaire and mapping rules map this questionnaire to a backend interface. The reverse mapping is subject to future work.

All the input that is collected by one questionnaire is called a *profile*. Once the generic bridge receives a profile, it has to be forwarded to some backend system. Thus the profile becomes the actual parameter for an invocation of the backend system. The connector represents an abstract interface to various backend technologies (see Figure 5). Through inheritance, special purpose adapters can be created. For each of the special purpose adapters one needs to define a mapping between questionnaires to data types of the specific technology. In the following we describe the mapping for various connectors.

Mail The mail connector collects all user responses to a questionnaire, marks them up as an XML document and sends them to a designated email address. This configuration is useful for user feedback via email. Unless the data is not

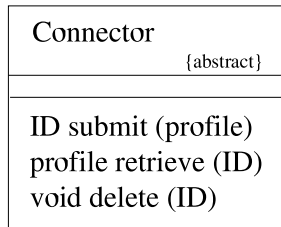


Fig. 5. UML diagram for the connector.

passed to another connector, the profile is simply sent as an email. The mail connector by itself does not store the profile. Right now the content of this email is an XML document. It is possible to add XSLT support to create a more human readable mail.

Database In many cases it is desirable to store profiles in a database. The profile becomes persistent and can be viewed and edited at a later point in time. For this reason our generic bridge includes a connector for relational databases.

Unlike the mail connector described in the previous subsection, the database connector requires a mapping of a questionnaire to a relational database schema. This mapping is defined once and then implemented within the database connector, so that the schema for the database can be automatically created out of the questionnaire definition.

Mapping of a questionnaire to a relational database schema is straightforward. Each question in the questionnaire is mapped to a column of a table representing the questionnaire (see Appendix A). Retaining the logical grouping introduced by sections is not necessary, so that the structure is flattened into a set of questions.

Repeatable sections pose a special challenge since a profile can contain multiple instances of this section. It is not known a priori how many instances a user might supply. For each instance the schema of the table would need to be extended by adding an appropriate number of columns. One way to map repeatable sections is to introduce a second table whose columns represent the questions in that repeatable section. Multiple instances of this section are represented by different rows of that table. Those rows that belong to one profile are linked via a foreign key of the relational database with the table representing the questionnaire itself.

CORBA A CORBA connector acts as a bridge between a user and a CORBA backend. Profiles submitted by the user are forwarded via CORBA object invocations to a backend server. Since in CORBA the interface of an object is specified via the Interface Definition Language (IDL), a questionnaire must be mapped

to an IDL-specification. The connector is then using the Dynamic Invocation Interface (DII) to translate profiles to object invocations.

A questionnaire is mapped to a CORBA interface. Each question type is mapped to an appropriate CORBA type. Sections are mapped to an IDL-structure. The nesting of structures mirrors the nesting of sections. In the case of CORBA, repeatable questions pose less of a problem than with the database connector discussed in the previous section. A repeatable section is mapped to an IDL-sequence that can have zero or more instances of that section.

Web Services The mapping for web services is similar to the one described for CORBA in the previous section. In web services, the interface of an object is defined through the Web Service Definition Language (WSDL). Just as for CORBA, a mapping needs to be defined to map a questionnaire to a WSDL specification. The web service connector generates SOAP messages based on the schema of a questionnaire.

The web services mapping follows the same pattern as the mapping for CORBA. Question types are mapped to appropriate WSDL types. Sections are mapped to structures. Repeatable sections are mapped to sequences of structures.

4 Implementation

The concepts described in this paper have been implemented and have become known under the name QTool (for questionnaire tool). QTool is implemented using the scripting language Python (see [5]). Python has proven to be very suited for this kind of application. In our opinion it is easier to learn and maintain than Perl. In this section we describe some of the implementation details.

4.1 Commands

QTool runs as a CGI script invoked by the web server. Both Apache and Microsofts IIS are supported. The functionality of QTool can be accessed through the usual GET and POST conventions of the HTTP protocol. E.g., to get an empty questionnaire for a schema called "CORBA", the following URL need to be invoked:

```
http://<host>/cgi-bin/qtool.cgi?ACTION=NEW&PROFILE=CORBA
```

There are several different commands with different arguments that can be accessed in a similar way. These URLs can be used as hyperlinks or anchors within a custom HTML page. It is also possible to access the functionality of QTool through Server Side Includes (SSI). With this technology a specially marked up HTML page contains inline calls to a CGI script. The web server will replace those inline invocations with the output generated by the CGI script. This allows custom HTML pages that embeds the output of QTool in a very flexible way.

4.2 PSP

QTool is implemented in Python and is invoked as a CGI script from a web server. The script's main task is to generate HTML that is rendered in the client's browser. There are different approaches to generating the HTML, one popular one being *Java Server Pages* (JSP). In JSP, the web page is interspersed with Java code that emits the dynamic part of that page. Several projects have adopted this scheme for Python, which consequently is called *Python Server Pages* (PSP) (see [1]).

One challenge of PSP in contrast to JSP is the fact that in the language Python programming blocks are marked through different indentation levels. Therefore, instead of marking a block with '{' and '}' as done in Java, the statements belonging to a block in Python have the same indentation level.

This poses a challenge when applying the JSP idea to Python, because it is difficult to maintain the indentation level when HTML code is interspersed in a Python program. One solution to this problem is to extend the Python language by adding explicit begin and end markers for programming blocks, as done in X.

For QTool we have taken a different approach. While JSP is based on the principle "HTML code with interspersed Java code," our version of PSP reverses this principle to "Python code interspersed with HTML code." The Python code defines the level of indentation and the HTML part is embedded naturally at the respective indentation level. The following example illustrates this:

```
# PSP code
for i in range (10):
    <!
    Hello World.
    <P>
    !>
<!
The end.
!>
```

An HTML block is enclosed with the markers "<!" and "!>". Inside this block all code is treated as HTML and not as Python code. Note that the embedded HTML code follows the indentation rules of Python. The PSP code above will generate ten "Hello World" followed by one "The end." In the generated HTML code, the whitespaces that are required to mark the level of indentation within the PSP program will be removed.

Similar to JSP, our adaptation of PSP allows the embedding of Python expressions within an HTML block. The following code excerpt illustrates this:

```

# PSP
numbers = ['one', 'two', 'three', 'four', 'five']
<!
Counting from 1 to 5:
!>
for i in numbers:
    <!
    Number <% i %>.
    <br>
    !>

```

4.3 Connectors

One of the central concepts in QTool is that of a connector that serves as an abstract interface to various backend technologies. Since QTool is written in Python, specific connectors need to be implemented as derived classes in Python. The mail connector uses Python's builtin SMTP library. The CORBA connector uses Fnorb; a Python-based CORBA implementation (see [7]). For web services, QTool makes use of SOAPPy, a web service implementation for Python (see [8]). Finally, the relational database connector makes use of a Python interface to MySQL (see [3]).

4.4 Examples

The concepts presented in this paper have been successfully implemented. The resulting implementation is placed under the GPL license and can be downloaded from <http://www.puder.org/qtool/>. QTool has been used for various projects. One of them is the San Francisco Movie page that lists movies made in that city. Besides the movies, this page also shows locations where certain scenes of a movie have been made. This feature makes use of repeatable sections, where a section called "Location" is marked as repeatable.

Another project where QTool has been used is the CORBA Product Matrix. This page gives a visual overview of the features of over two dozen CORBA products. Both the movie page and the CORBA product matrix use the database connector to store and maintain the information submitted by the user. The QTool homepage mentioned in the previous section contains links to those two QTool applications.

QTool has also been used within AT&T Labs for product evaluations. The relevant criteria of the product under evaluations are collected in a questionnaire for further analysis. A team of experts can easily submit their feedback by answering the evaluation questionnaire. In a different project within AT&T Labs, QTool is used as a generic front-end for a workflow engine. This particular application uses the web services connector to talk to the workflow engine.

5 Related work

We take a pragmatic approach to define the general user interface model. This model is inspired by the recurring design pattern of operational interfaces. This facilitates the mapping between operational and user interfaces. Question types have a natural mapping to types of various backend technologies. Repeatable and non-repeatable section can be mapped to constructed data types.

Web service technologies often include test interfaces where a primitive web-based user interface is automatically generated out of a WSDL (Web Service Definition Language). These are generally only meant for testing purposes and do not support complex data types. Examples are .NET, WebLogic Server, or XML Gateway. In previous work we have demonstrated that a knowledge representation technique called Conceptual Graphs can be used as a generic interface to CORBA's DII (see [6]). While very flexible, this approach is not suitable for end users.

Other approaches for general user interfaces focus on the data modelling aspect and usually have some cognitive model of the real world at the core of their design. Examples are the embodied construction grammars (see [2]) or the formal concept analysis (see [4]). While these approaches offer comprehensive solutions to represent arbitrary domain knowledge, they are too heavy-weight and not user friendly for general purpose user interfaces.

6 Conclusion and Outlook

Interoperability is one of the key issues when developing distributed applications. Interoperability is an end-to-end issue that spans from the user front-end to the backend. One prominent way to implement user front-ends is via a web-browser. In this paper we have introduced a way to dynamically create user interfaces for different backend technologies such as relational databases and different middleware technologies.

One future extension will revolve around the layout of the user interface. Currently, the layout is hard-coded in the implementation itself. We plan to use XSLT to change the look-and-feel through XSLT-style sheets. As another extension we intend to take a closer look at WSDL, the interface definition language for web services. Since both QTool questionnaires and WSDL are expressed through XML, it might be an interesting idea to merge these two formats.

A Mapping

The table below summarizes the mapping of the various question types to SQL, CORBA, and Web Services types. Note that the mapping is bidirectional. All question types have a corresponding mapping for the various backend technologies discussed here. The SQL mapping makes use of builtin types and tables. The CORBA mapping uses IDL-types to map the various question types. The web service mapping makes use of types defined in XML schema. Currently some

types (e.g., a CORBA-struct) can not be mapped to a questionnaire. While a mapping is possible, this will be subject to future work.

Type	SQL	CORBA	WS
String	TINYTEXT	string	<xsd:string>
Textfield	MEDIUMTEXT	string	<xsd:string>
Email	TINYTEXT	string	<xsd:string>
URL	TINYTEXT	string	<xsd:string>
Bool	ENUM	boolean	<xsd:boolean>
Checkbox	ENUM	boolean	<xsd:boolean>
Feature	ENUM	enum	<xsd:enum>
Ranking	ENUM	enum	<xsd:enum>
Document	LONGBLOB	sequence<octet>	<xsd:sequence>
Questionnaire	TABLE	interface	<service>
Non-repeatable section	TABLE	struct	<xsd:sequence>
Repeatable section	TABLE (foreign key)	sequence<struct>	<xsd:sequence>

References

1. R. Barr. Python Server Pages. <http://spyce.sourceforge.net/>, SourceForge, Cornell University, 2002.
2. B. Bergen and N. Chang. Embodied construction grammar in simulation-based language understanding. In Jan Ola Ostman and Mirjam Fried, editors, *Construction Grammars: Cognitive and Cross-Language Dimensions*. Johns Benjamins, 2002.
3. A. Dustman. MySQL for Python. <http://sourceforge.net/projects/mysql-python/>, SourceForge, 2003.
4. F. Lehmann and R. Wille. A Triadic Approach to Formal Concept Analysis. In *3rd International Conference on Conceptual Structures (ICCS'95)*, Santa Cruz, University of California, 14–18 August 1995. Springer Verlag.
5. M. Lutz. *Programming Python*. O'Reilly & Associates, second edition, 2001.
6. A. Puder and K. Römer. Use of Meta-Information in a CORBA Environment. In *Workshop on CORBA: Implementation, Use and Evaluation*, Jyväskylä, Finland, 1997. European Conference on Object-Oriented Programming (ECOOP).
7. R. Smith. Fnorb: A Python-based CORBA implementation. <http://sourceforge.net/projects/fnorb/>, SourceForge, Distributed Systems Technology Centre (DSTC), 2002.
8. C. Ullman and B. Matthews. SOAPpy: Web Services for Python. <http://sourceforge.net/projects/pywebsvcs/>, SourceForge, 2003.