Faculté des arts et des sciences
Département d'informatique
et de recherche opérationnelle

rali

Technical documentation for project NLI-SARC

Fabrizio Gotti – gottif@iro.umontreal.ca
Guy Lapalme – lapalme@iro.umontreal.ca

31 March 2015
Recherche Appliquée en Linguistique Informatique

Version 1 – 31 March 2015

Université de Montréal

1 Introduction

This document describes the software deliverables created for the Natural Language Interface for a Suspicious Activity Report Client (NLI-SARC) project. We describe:

- the structure and content of the archive containing all software and associated resources
- the procedures necessary to deploy and operate the software

The software archive for NLI-SARC contains, among other software, the prototype XSDGuide version 0.2 (as of this writing), which creates Web interfaces from existing XSD schemas.

The software archive for NLI-SARC is named `nli-sarc.zip` and has the following directory structure and content:

Directory	**Content**	**Section**
`xsdguide/bin`	Binaries for the XSDGuide web server	Section 2
`xsdguide/src`	XSDGuide source code	Section 4
`xsdguide/sarrali`	SAR-RALI SAR schema proposition and encoding of a sample scenario	Section 5

The archive also contains the present document.

1.1 The main application XSDGuide

XSDGuide is a web application coded in Java. Besides the library for the project, the jar delivered contains a web server working as a demo. It serves a single-page web application facilitating the creation of XSD-based SAR reports.

XSDGuide is a self-contained web server and does not require Apache or MySQL. It has only been tested extensively on Linux. However, it works correctly on Windows and Mac.

The executable is a single .jar file containing the compiled code and all required libraries. Its name is `xsdguide-XX-jar-with-dependencies.jar`, where `XX` is the version.

Multiple users can use the frontend simultaneously.

2 Launching the XSDGuide server on the command-line

To run the XSDGuide executable, the Java 7 runtime (or later) is needed. Type the following command at the prompt, on a single line:

```
java -jar xsdguide-0.2-jar-with-dependencies.jar
/full/path/to/directory/content/
port
/full/path/to/directory/storing/schemata
catalog.xml
```

or run the convenience script `start-linux.sh` in the `bin` directory to start the server with sensible values. The server will then be available at URL `http://localhost:8080/content`, where `localhost` is the machine XSDGuide has been launched on. On Windows, use the script called `start-windows.bat`.

The 4 arguments to the program are the following:

1. The full path to the directory containing the HTML page and resources for the main page of the application. This directory is archived in the deliverable at path `xsdguide/bin/content`
2. The port at which the Web server will be listening. This can be any port between 1024 and 65000.
3. The full path to the directory storing the schema definitions. This directory contains XSD schemata used to generate the interface automatically. This directory is archived in the deliverable at path `xsdguide/bin/schemata`.
4. The path for the file `catalog.xml`, an XML catalog[1] used to speed up the loading of XSD files by locally storing often-imported XSD files. This catalog file and its associated directory is archived in the deliverable at path `xsdguide/bin/catalogonly.`[2]

For instance, if the zip deliverable is unzipped in directory `/local/home/rali`, then the following command will run the server on port 7777:

```
java -jar /local/home/rali/xsdguide/bin/xsdguide-0.2-jar-with-
dependencies.jar
/local/home/rali/xsdguide/bin/content/
7777
/local/home/rali/xsdguide/bin/schemata/
/local/home/rali/xsdguide/bin/catalogonly/catalog.xml
```

To stop the server, kill the process using (e.g. by using CTRL+C on Linux). This will not lead to data corruption.

[1] http://en.wikipedia.org/wiki/XML_Catalog

[2] These files are taken from the catalog found in oXygen 16.1.

3 Operating XSDGuide

3.1 User interface

Once the server running, it is possible to visit the page

`http://host:port/content/`

in order to use the application. Don't forget the `content` part in this URL.

The navigation bar contains the following menus:

New report	Creates a new report based on the currently active XML schema. Only XSD schemata are supported.
Schema Manager	Manages XSD schemata.
Schema Manager > Add XML schema	Allows the user to upload an arbitrary (but valid) schema or schema zip archive in order to change the currently active XML schema on which new reports are based. See section 3.2.
Schema Manager > Change default XML schema	Allows the user to pick the currently active schema among those currently uploaded on the server.
Switch to simple view / Switch to advanced view	Toggles the level of detail in the report.
Validate XML	Validates the report being created in the interface. See section 3.3.
Save to XML	Opens a new browser window containing the XML document corresponding to the currently edited report. An XML document can be produced at all times, whether the report is valid or not. The user can then save the file using his browser.
Help	Provides a small help message.

3.2 Adding new schemata

It is noteworthy that, when creating a new XML report from a XSD schema, a *root element* must be specified. This root element must be picked among the top-level elements defined in the schema, and essentially specifies "were to start" when creating a new instance document (an XML file).

A user follows these steps to add a schema to the application:

1. Upload of a unique `.xsd` file to the site or a zip archive containing multiple `.xsd` files. The latter case is useful when an XSD schema refers to others, most notably through `import` statements.
2. When a zip archive is uploaded, selection of the file which contains the root element for the report.
3. Selection of the root element for the SAR.

The association schema+root element is then available as a possible choice when creating a new report via the option "Schema Manager > Change default XML schema" in the UI.

Internally, XSDGuide proceeds as follows when a user uploads his files:

1. The uploaded schema file/zip file is copied to the temporary directory of the server machine. This is typically the `/tmp` directory on Linux machines.
2. If the file is a zip archive, it is unzipped in the temporary directory. Only files with extension `xsd` are actually extracted.
3. Each `.xsd` file is validated. If an invalid file is found, the whole operation is aborted.
4. Once the user picks the root file and element and completes the operation on the frontend, all files are copied in XSDGuide's schema directory. This directory corresponds to the 3rd argument described in section **Erreur ! Source du renvoi introuvable.**.
5. An entry is appended to the file `schemadefs` contained in the schema directory.
6. The uploaded files in the temporary directory are deleted.

The `schemadefs` file contains one line per entry. Each entry describes the location of na XSD file as well as the name of the root element. Its format is illustrated below.

```
id1     guideserver/sar-rali.xsd  http://rali.iro.umontreal.ca/sarrali:SuspiciousActivityReport
id2     guideserver/test01.xsd  http://rali.iro.umontreal.ca/test:Test
```

Each line contains a unique identifier, the path of the xsd file relative to the schema directory and the fully qualified name of the root element. Fields are separated by tabs. This file can be edited manually without interrupting the server.

3.3 Report validation

Report validation is carried out using 2 software elements: the frontend validates the data entered in the web page's form fields using ad hoc rules, and the backend validates the complete report using the validation engine provided by Apache Xerces, and driven by the selected XSD schema. The error messages are presented to the user in a transparent fashion: he does not know which software part issued the validation error. The document is submitted for validation to the backend only when the frontend deems it error-free. Otherwise, certain error messages would appear twice.

When no errors are detected, a message is shown saying that the document is valid.

3.4 Security considerations

The prototype presents certain security risks because it allows a user to upload arbitrary files to the server, and because these files are then used internally and served afterward. While we did our best to prevent certain obvious attacks, the prototype is vulnerable to certain malicious interactions, including but not limited to:

- DOS attacks, by uploading huge files or Zip bombs[3]
- Maliciously crafted XSD files, using for instance special `import` statements

[3] http://en.wikipedia.org/wiki/Zip_bomb

- DOS attacks, by overloading the server with temporary files or numerous schemata
- Directory traversal attacks[4], for instance when the front-end asks to serve a particular file, relative to the schema directory.

> ⚠ In light of these vulnerabilities, we highly recommend that the prototype be run on a non-critical server (possibly a virtual machine located in the DMZ) containing no sensitive data whatsoever.

3.5 Housekeeping

When the upload process fails, this may result in the accumulation of temporary files in the temporary directory.

To delete temporary files, simply manually empty the temporary directory (usually `/tmp` on Linux-based machines).

Moreover, the administrator may decide to regularly delete uploaded schemata to avoid cluttering the server.

To do so, remove the unwanted schemata and directories from the schema directory, and update the `schemadefs` file accordingly. See the end of section 3.2 for more details. These modifications can be made “live”, i.e. without stopping the server.

These tasks could be automated using a `cron` job.

[4] http://en.wikipedia.org/wiki/Directory_traversal_attack

4 XSDGuide Java Sources

XSDGuide's backend is written in Java 7, and leverages the Apache Xerces libraries[5]. It has been developed with the Eclipse IDE and, as such, can be imported[6] in Eclipse using the project available in the subdirectory `xsdguide/src/eclipse` of the archive.

The front-end is a web application, and uses HTML and JavaScript. The sources are in the subdirectory `xsdguide/bin/content`.

4.1 General architecture of the complete web application

The frontend is a single HTML page (`index.html`) that interacts with the backend (Java server) through Ajax and JSON. This means that JSON queries are made to the backend and their results are used to populate the interface. Most of this interaction logic is written in JavaScript.

The requests made from the frontend to the frontend concern the following actions:

— Creating a new report

— Creating and deleting an element in the report

— Getting the HTML code for a report (rendering)

— Updating the underlying XML (server-side) with the values entered by the user

— Asking for the validation results for the current report

— Downloading the XML for the currently edited document

— Uploading a new schema

4.2 Java library

4.2.1 General organization

The Java 7 library written for this project resides in the package name `ca.umontreal.rali.xsdguide`. Here is the package and subpackage organization, under the package `ca.umontreal.rali.xsdguide`.

`instance`	Package containing the logic to maintain an instance of a SAR report, both the UI guide and the actual XML document.
`guides`	Package creating UI guides from an arbitrary XSD document
`gui`	Package rendering a UI guide into a concrete interface. In our case, the class `HtmlFragmentRenderer` creates all HTML elements necessary to render the interface. The other renderers are not completely implemented.
`tools`	Utility classes.

[5] http://xerces.apache.org/

[6] See the procedure described here : http://help.eclipse.org/luna/index.jsp?topic=%2Forg.eclipse.platform.doc.user%2Freference%2Fref-70.htm

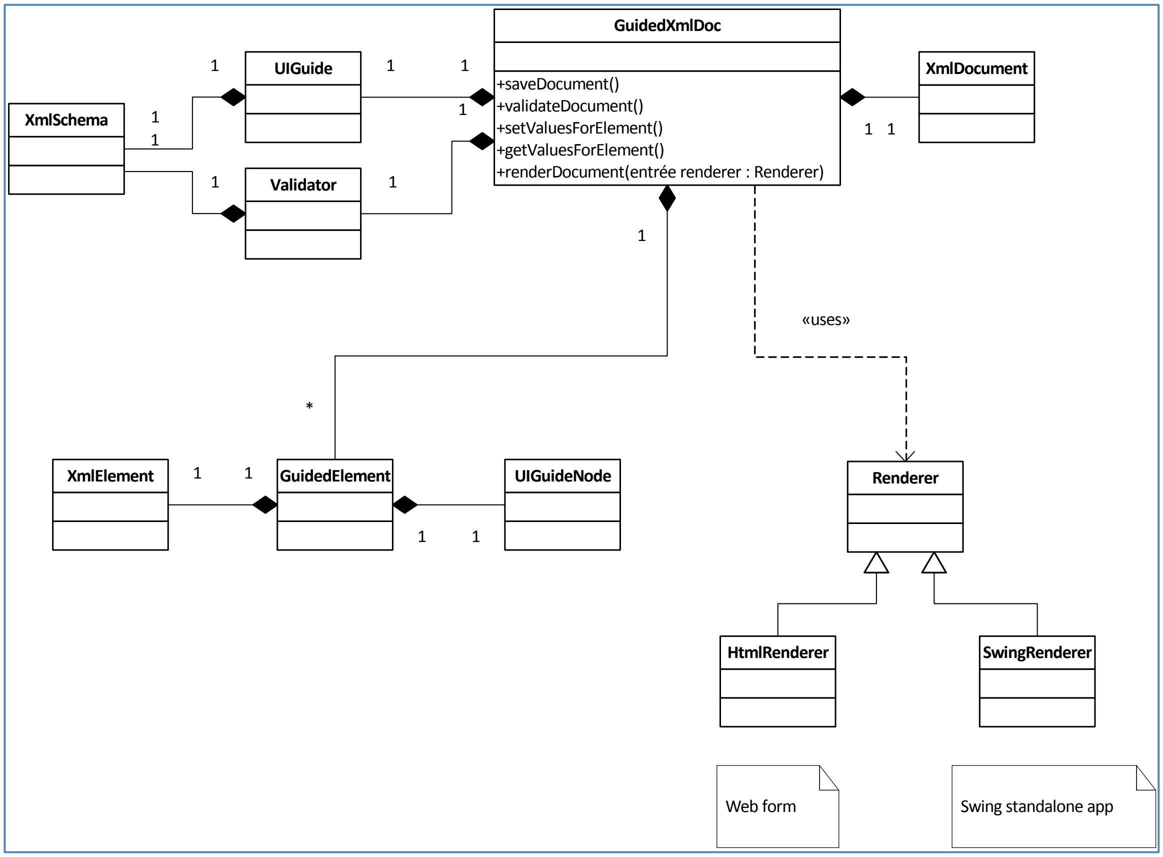
`demo`	The web server, and all related classes necessary for the application (demo) to run correctly.

The web server is implemented using Jetty[7], a lightweight web server that obviates the need for heavier solutions, like Apache and Tomcat. Additionally, since most data is written in plain text files and XSD files, there is no need for MySQL or other database systems.

The main executable for the demo is `ca.umontreal.rali.xsdguide.demo.GuideServer`. When launched, it creates a Jetty web server and listens for incoming commands emanating from the HTML page. It responds in JSON.

4.2.2 Architecture

The general architecture of the library is illustrated in the figure below.



At the heart of the running prototype is a `GuidedXmlDoc` object that allows the assisted creation of an XML file (the SAR report in our case). This object is created from an XML schema (specified in XSD format). The latter serves two roles: It drives the creation of an interface guide (`UIGuide`) and the creation of an XML validator (`Validator`). XML elements forming the XML document are each associated with an interface guide. This association is maintained in multiple `GuidedElement` objects.

[7] http://eclipse.org/jetty/

Finally, an object of type `Renderer` allows the rendering of an element in the desired format. In our case, only the production of HTML fragments is fully implemented (`ca.umontreal.rali.xsdguide.gui.HtmlFragmentRenderer`), and allows the interface to load the desired HTML fragment when it needs to render the corresponding XML element.

4.2.3 Limitations of the current implementation

Please note that the prototype has its limits regarding certain aspects of its features.

We concentrated our efforts on covering as many features of the Xml schema standard as possible, but we fell short of implementing all possibilities. Here are unimplemented XSD features:

- Elements with `mixed="true"`
- Elements of type `xsd:any`
- Simple types whose "variety" is `union` or `absent`
- Subtle distinctions between text types `string, ncname, nmtoken, token`
- Data type `uri`
- Facet constraints of type `whitespace, length, maxlength, minlength, totaldigits`
- The regexp language used by XSD
- Multiple regexp constraints
- Cardinalities other than 1 for elements of type `sequence`, `choice` ou `all` (it is nonetheless noteworthy that, within these elements, all cardinalities are supported, including the "unbounded" cardinality).
- Schema components of type "wildcard"
- Groups of type `all` or `choice` are treated like `sequence`

These limitations will result in error messages being printed on the console *on the backend side only*. The user will not see those messages.

It should also be noted that it is not possible, at this point, to *load* an existing XML SAR report into the application. It is possible to save a report from the application to the user's disk, but not the other way around.

4.3 HTML frontend

The frontend contains the following files, in the subdirectory `xsdguide/bin/content`,

`index.html`	Web page, containing all HTML code.
`css/`	Directory containing the CSS for the project (`sar-rali.css`) as well as the Bootstrap[8] and JQuery[9] CSS plugins.
`fonts/`	Fonts required by Bootstrap.
`images/`	Logos
`js/`	JavaScript logic, including JQuery and boostrap parts.
`js/sar-rali.js`	The main JavaScript file for this project, containing all frontend logic written by RALI.

[8] http://getbootstrap.com/

[9] http://jquery.com/

5 SAR-RALI schema proposition

In order to focus our efforts on the creation of the interface, we used a custom-made SAR schema, called SAR-RALI. This schema is available in directory `xsdguide/sarrali`. The files for this proposition are as follows:

`sar-rali.xsd`	SAR-RALI schema proposition
`ScenariosSar-v2.docx`	Description of 4 fictitious SAR scenarios, and associated documentation
`ui-001.xml`	Encoding of scenario ui-001 (taken from the above-mentioned report), encoded in an XML file conforming to schema `sar-rali.xsd`
`niem-encoding/`	For comparison, encoding of scenario ui-001 using NIEM-SAR standard.

Please note that the `sar-rali.xsd` schema comes pre-loaded with XSDGuide and constitutes the default schema the user creates reports from.