Berger-Levrault - Leveraging the power of GWT rebinding to build massive applications

---

Author: André Morassut, R&D Engineer
Date: 27/07/2011
Mail contact - andre.morassut[at]gmail[dot]com / andre.morassut[at]berger-levrault[dot]com
LinkIn profile - http://fr.linkedin.com/pub/andr%C3%A9-morassut/38/351/6a7

The context

About Berger-Levrault

Berger-Levrault is a French private company and a major actor in the field of public administration software solutions. Starting in **1676**, Berger-Levrault has evolved towards software development in the late 1980's and is, today, the 6th software development company in France, delivering products to more than 56000 public administrations over 180000.

Early 2007, we started to use GWT/Hibernate/Spring (starting with GWT 1.3.1., as early adopters) to build our new major Financial and HR software products.

About the author

Having strong C++ and Java roots, I worked on this software architecture and the subsequent technical developments, with an emphasis on the GWT integration as I am the author of the systems described in this blog entry and main contributor to this success on GWT. These systems made the firm earn an innovation subvention from the French government and, so far on those products, we produced 6,000+ GWT classes for more than 685,000 LOC.

About Berger-Levrault R&D

André Morassut is a member of the **Berger-Levrault R&D team** managed by Mr Derras Mustapha, CTO of the group (http://fr.linkedin.com/in/derras). Working on fields ranging from Android® development to Talend® integration, this team represents the technological task force of Berger-Levrault.

The challenges

Berger-Levrault applications are heavyweights in terms of functionality and business complexity. They were also been developed as desktop applications in the past. By using GWT, we faced several challenges:

- ➔ **Handling massive applications:** A single Berger-Levrault application can represent 400+ different windows (i.e. a GUI for a specific functionality). Right before the GWT 2.0 release, we were shipping more than 10MB of obfuscated Javascript !

- ➔ **Handling modular and fast-changing structures:** Our modules (i.e. career management, payroll …) are not developed following a linear project management planning while belonging to a single application. We developed the ability of integrating functionalities one to each other's while some of them were not yet implemented, activating/deactivating functionalities at compile-time between shipping operations or at runtime according to licensing information.
- ➔ **Integrate non developer designers:** On one hand business rules are so complex that we cannot ask developers to design the functionalities and corresponding GUIs. On the other hand, the experts designing them are mainly unaware of software technical aspects.

The solutions

- ➔ **Functionality cartography:** We integrated a functionality management system (which is coincidentally close to the activity/intent management system of the Android framework) letting developers code functionalities as "self managed panels" and declare them to interact with other functionalities on a loose coupling scheme.

  Declaring functionalities is straightforward:

```
<phase functionalityCode="PHASE_AVECH_PARAM_ORGA"
habilitationCode="EAV_PORGA"
className="fr.bl.client.grh.ca.par.po.PhaseParamOrganisme"/>
```

  Interaction does not require knowing target implementation:

```
Workspace.getPhaseManager().displayPhase(ConstantsPhase.Util.get().PH
ASE_AVECH_PARAM_ORGA(), aBuinessObjectInstance,
AbstractDesk.TYPE_POPUP_MODAL);
```

  Moreover, our framework is nice enough to handle deactivation or disabling navigation links based on user license information and XML declaration.

- ➔ **Automatic module download:** to solve the script size issue, we used the capability of our framework to know each functionality and functionality interaction to integrate an automatic module download.

  Declaration of a module:

```
<module name="e.Carrière">
    <phase […] />
    <phase […] />
</module>
```

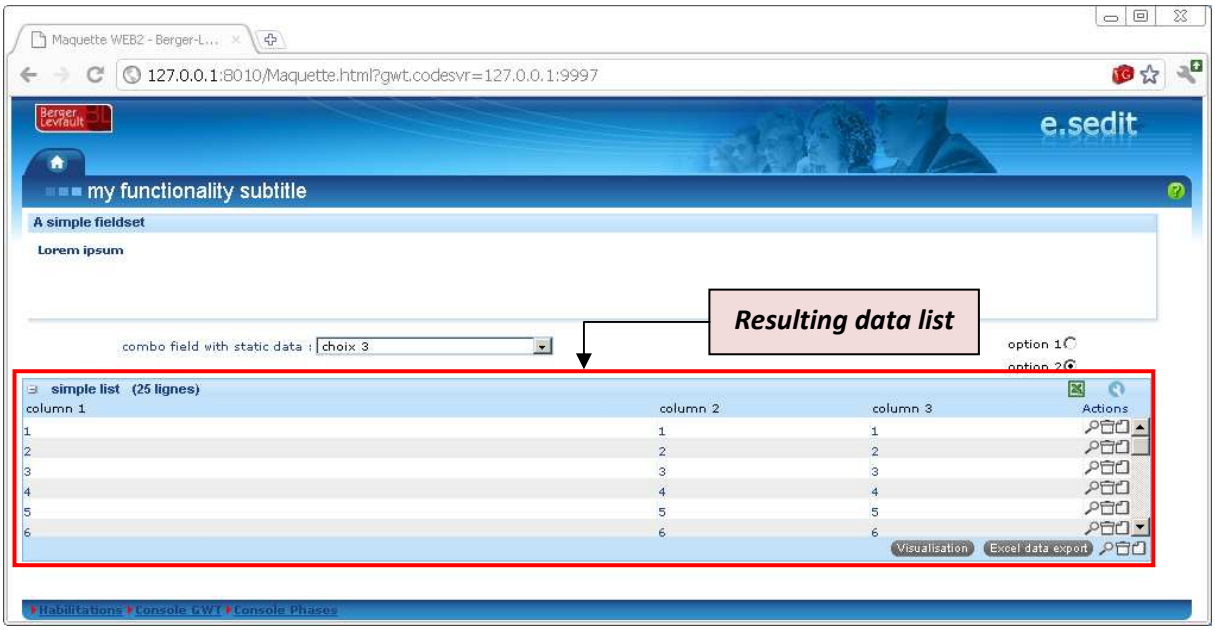
The first interaction using one of the functionalities inside the module will trigger the module download. This is achieved by using a rebinding feature to generate module loaders and insert split points at compile time.

➔ **XML GUI language:** Business product experts use a presentation program (like PowerPoint®) to build a draft of the expected GUI and developers get this input as part of the specifications. For the sake of process efficiency, we wanted the business product experts' work to be directly usable by the developer. All the static aspects are specified by the business expert and the developer adds the dynamic aspects (RPC calls, rules implementation …). We rely on the rebinding feature to establish an independent GUI high level specification syntax.

Example of a XML fragment for a data list:

```
<bl_list id="list1" title="simple list" nbrows="60" foldable="true"
filterable="false" sortable="false" xlexport="MyExportBean" columnchoice="false"
width="950px" height="100"
checkbox_on_rows="true" enabled="false" action_column="SUPPR, MODIF">
<bl_ list_col title="column 1" bean_attr="" visible="true"
largeur="60%" tri="NONE" filterable="false" />
<bl_ list_col title="column 2" bean_attr="" visible="true"
largeur="20%" tri="NONE" filterable="false" />
<bl_list_col title="column 3" bean_attr="" visible="true"
largeur="20%" tri="NONE" filterable="false" />
       <bl_list_footer colonneactions="VISU, SUPPR, MODIF">
            <bl_list_action type="VISU" titre="Visualisation" />
            <bl_list_action type="EXCEL" titre=" Excel data export" />
        </bl_list_footer>
</bl_list>
```

The resulting screen:



The benefits

- ➔ **"Users only pay for what they use":** when accessing and using the application, the user browser only downloads the code they need. Imagine a HR payroll manager only accessing the payroll module: no other application module will be downloaded. This leads to a faster, responsive application despite of its size.
- ➔ **Modularity and reorganization:** by altering the XML declaration of modules and functionalities, the download behavior and the module sizes can be completely rearranged without modifying one single line of java code.
- ➔ **Prototype – validate - wire :** introducing the XML language in the development process leads to a productivity benefit as the iterations on the GUI prototyping leads to a well formed set of classes directly usable by the development team. Once the GUI validated, all it takes is launching the rebinding process to get the classes and begin to wire RPC and business logic.