BDNLS - BESSY DEVICE NAME LOCATION SERVICE

D. Engel, P. Laux, R. Müller, HZB, Berlin, Germany

Abstract

Initially the relational database (RDB) for control system configuration at BESSY has been built around the device concept [1]. Maintenance and consistency issues as well as complexity of scripts generating the configuration data, triggered the development of a novel, generic RDB structure based on hierarchies of named nodes with attribute/value pairs [2]. Unfortunately, it turned out that usability of this generic RDB structure for a comprehensive configuration management relies on sophisticated data maintenance tools. On this background BDNLS, a new database management tool, is currently under development using the framework of the Eclipse Rich Client Platform. It uses the Model View Controller (MVC) layer of JFace to cleanly separate retrieval processes, data path, data visualization and actualization. It is based on extensible configurations defined in XML allowing to chain SQL calls and compose profiles for various cases. It solves the problem of data key forwarding to the subsequent SQL statement. BDNLS has the potential to map various levels of complexity into the XML configurations. This provides usable, tailored database access to configuration maintainers for different underlying database structures. Based on Eclipse, the integration of BDNLS into Control System Studio [3] is straight forward.

HISTORY OF CONCEPT

Views – The First Approach

Database-views are predefined SQL-queries. They are like macros with precached results. Views are ideal for frequently used large and complex queries typically used in small database-applications. A big drawback of views is, that they are not always able to update their data sources. Views are able to hide the complexity of data structures, but you have to create one view for one specific problem. If you need to create views for all aspects of a facility, then you need hundreds of them. Complexity of data structures is pseudo-simplified in a confusing amount of views.

Generalization of the Data Structure

The obvious idea to maintain full flexibility, was to separate the data sets to the maximum, like atomic elements. For that purpose, the whole data set, for example device properties or I/O specifics, has to be to modeled in data and relations. The data is not grouped by table structures or specific column names. Only links and relations connect the data. This is a clean method to retain the data and avoid redundancy. This data structure has been called Gadgets [4]. This solution has the drawback, that you manage a very complex structure of stored data. Visualizing and managing the data is problematic because it is abstract and queries on this structure are very complex. It is not a general solution for storing data.

The Experiences from a Prototype

The first graphical application was implemented using Eclipse as a Rich Client Platform (RCP) application. RCP provides many solutions of graphical interfaces and a plugin concept to assemble numerous small plugins to a full-featured application. The application was designed as well as a standalone Rich Client Application and as plugin for other Rich Client Applications like the Control System Studio (CSS) [3] Framework. The application is able to provide the location of a device and to view their properties as requested. This solution used static SQL-queries and a special view that provided all information for a complete navigation tree, broken into columns. Updates of the data is only possible with hard coded update-dialogs. Another drawback of this approach was, that data could only be updated using the dialogs provided.

The Generalized Approach

The first graphical approach revealed with following shortcomings:

- Search, read, change and insert elements in the database without knowledge of SQL by the user.
- Importing and exporting huge data sets.
- The profile contains the navigation chain to a target element, the element properties, the wizards[1] and the popup-menus available for this target.
- The profiles should be able to change during operation.
- The complexity of data structures and SQL-queries is extracted from the source code and encapsulated to a configuration file. The source code is generic.
- The application should be independent of the given data structure. You can design your own SQL-queries in the configuration file.
- With the use of Java, a platform independent application is provided.
- The application uses existing databases and no hard coded data structures. It is also not designed to follow all data relations given by data keys hence it is no competitor of IRMIS [5].

[1] A Wizard is a user interface of dialogs to help the user to manage complex inputs

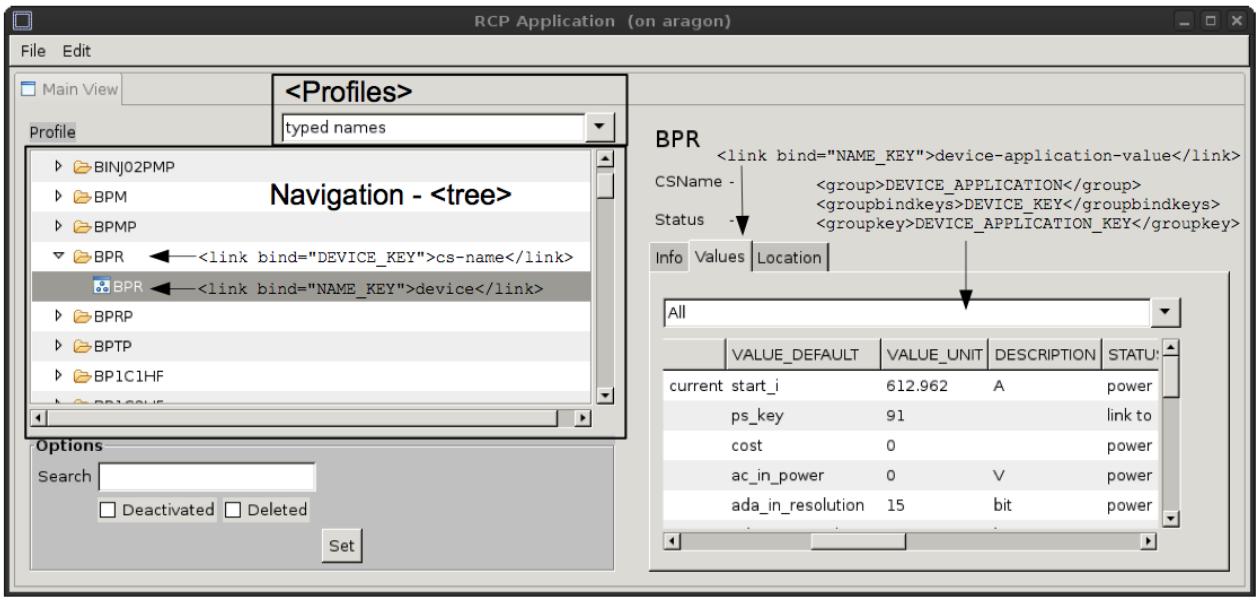

Figure 1: BDNLS prototype application; Profile selector (top), tree navigation (left), view section (right), the query section not visible.

XML CONFIGURATION CONCEPT

The whole configuration of the BDNLS-application is defined in an XML-file. The configuration file contains:

- How to access the database (URL, user name and the password).
- Definition of various profiles allowing several SQL-query sequences and different views.
- The displayed element and the data key provided by the SQL-Query.
- The definition of the navigation tree (-by profile-) and the data keys required for next tree element.
- The provided properties of a selected device, arrangement, grouping and visualization settings.
- The popup-menus and wizards provided for a device.

Profiles

A profile defines the navigation to a device and its specific properties. profile names are declared in the `<profile>`-part. The default profile used when the application starts.

```
<profiles default="1">
   <profile>located names</profile>
   <profile>typed names</profile>
   <profile>typed devices</profile>
   <profile>by facility and family</profile>
</profiles>
```

Navigation

The navigation section is divided in to different profiles. Every profile consists of the parts `<tree>` (shown as a tree elements) and `<tab>` (shown as tab elements). The tree-part is used to display and to navigate through device classes, domains, families and facilities. `<link>`-parts are called sequentially when expanding a tree element. `<link bind="xxx">` is used to narrow the results at the next hierarchy, using the keys from the last results. Clicking at the last element in the tree-part, initiates processing of the tab-part.

```
<navigation>
<profile name="located names">
   <tree>
      <link>facility</link>
      <link bind="FACILITY_KEY">cs-domain</link>
      <link bind="NAME_DOMAIN_KEY">cs-subdomain</link>
      <link bind="NAME_SUBDOMAIN_KEY">cs-name</link>
    </tree>
    <tab>
       <link bind="NAME_KEY">device-info</link>
       <link bind="NAME_KEY">device-location</link>
       <link bind="NAME_KEY">device--value</link>
    </tab>
</profile>
</navigation>
```

Query Section

The query section describes the name of the query, the corresponding statement `<statement>`, the provided key names `<key>` of the query and the column name to be shown as the element name`<label>`. The provided keys are stored and can be used to narrow the results of the next query. A statement may yield zero, one or many keys, they are separated by a comma.

```
<link name="cs-subdomain">
    <statement>NAME_SUBDOMAIN_DROPDOWN</statement>
    <key>NAME_SUBDOMAIN_KEY</key>
    <label>NAME</label>
</link>
```

View Section

Currently, the view section is only used for the properties of a device. The view part defines the name of

a tab <title>, whatether it shown as a form or a list <showtype> and which type of context menus the elements provides <showmenu>. Additionally, the output can be filtered by a group query. To group/filter the device properties, you have to define a group statement <group> and a grouping/filter key that is used to restrict the results of the device properties. To restrict the group statement itself, you can restrict the statement with another key provided by the device properties.<groupbindkeys> The tag <showmenu> links to the required popup-menus.

```
<view>
<tab name="device-value">
    <title>Values</title>
    <group>DEVICE_APPLICATION</group>
    <groupbindkeys>DEVICE_KEY</groupbindkeys>
    <groupkey>DEVICE_APPLICATION_KEY</groupkey>
    <showtype>List</showtype>
    <showmenu>device-value</showmenu>
</tab>
</view>
```

Popup and Wizard Block

Manipulation of the data, is enabled by an update or input wizard defined within the popup tag <popup>. The key expected by the wizard is defined with the tag <column>.

```
<popup name="device-value">
    <item name="create">
       <wizard>wizard-device-value</wizard>
       <column>device_key</column>
    </item>
    <item name="edit">
       <editor>editor-device-value</editor>
       <column>device_value_key</column>
    </item>
    <item name="remember">
       <history>history-device-value</history>
       <column>device_value_key</column>
    </item>
</popup>
```

IMPLEMENTATION

The Eclipse Rich Client Platform was chosen for implementation. By using and providing plugin extension points, it is possible to use other Eclipse based or third party plugins to extend the application. To connect to the database, a JDBC interface is used. The connection class can load many kinds of JDBC bridges dynamically, like PostgreSQL, MySQL, Oracle and many more. But at the moment only a Oracle OJDBC bridge is integrated in this project. The GUI of the first implementation of this project is shown in Figure 1.

CSS Integration

Adding the plugin-activator class allows to integrate this application as a plugin in CSS. Hence it is possible to exchange EPICS [6] process variables (PV) with CSS, to retrieve archived data, or to get all device information depending on a PV.

FUTURE DEVELOPMENT

- Implementation of drag & drop methods to improve the data exchange with CSS, other windows/forms and applications.
- Integration of visualization function, for example to show the location of a device.
- Integration of EPICS-PVs for fast visualization of the device state.
- Abstraction of the wizards to be fully configurable and provide plugin extension points for custom forms and visualization classes.
- Methods providing import and export formats for CSV, XML and XLS.
- Provide more JDBC Interfaces.

REFERENCES

[1] T. Birke et al., "Relational Database for Controls Configuration Management", IADBG Workshop 2001, San Jose, CA, USA.

[2] T. Birke et al., "Beyond Devices – An improved RDB Data-model for Configuration Management", PO1.078-7, ICALEPCS'05, Geneva, Switzerland.

[3] Jan Hatje, M. Clausen et al., "Control System Studio (CSS)", MOPB03, ICALEPCS'07, Knoxville, Tennessee, USA, http://cs-studio.sourceforge.net.

[4] T. Birke et al., "Introducing I/O Channels into the Device Database Open new Potentialities for Configuration Management", WEDT003, ICALEPCS'01, San Jose, CA, USA.

[5] D.A. Dohan, "The IRMIS Universal Component-Type Model", TOPA03, ICALEPCS'07, Knoxville, Tennessee, USA, http://aps.anl.gov/epics/irmis/index.php.

[6] http://www.aps.anl.gov/epics/