BEYOND DEVICES: AN IMPROVED RDB DATA-MODEL FOR CONFIGURATION MANAGEMENT

T. Birke, B. Franksen, B. Kuner, R. Lange, P. Laux, R. Müller, G. Pfeiffer, J. Rahn
BESSY, Berlin, Germany

ABSTRACT

At BESSY the initial approach of a device-oriented implementation of a RDB centred configuration management system revealed serious shortcomings. The resulting tight coupling of configuration parameters and corresponding application specifics requires changes in the data-model whenever new applications or modifications of existing applications are needed. Sharing of data between applications causes consistency problems and produces a high maintenance load.

A new data-model provides generic RDB-structures allowing to hold arbitrary configuration-parameter sets. The application-specific demands are no longer reflected by the structure of the RDB but the data stored in the generic RDB data-model. Main element is the possibility to store arbitrary hierarchies of named nodes with attribute/value-pairs. This paper describes the implementation status of the data-model and its API. An RDB-structure is given that allows representing acyclic directed graphs of nodes that carry the specific information. Inheritance, overloading and references are as well possible as arbitrary types of relations between nodes in this graph.

Since the novel RDB data-model by no means reflects the type of information stored, but provides the infrastructure to store any type of data, it is capable of storing the whole range of configuration parameters from hardware-access configuration to high-level-application meta-information.

INTRODUCTION, MOTIVATION

There are a couple of design factors, that make up the difference between a powerful control system and a less successful approach – architecture, performance, consistency, flexibility, maintainability. For complex systems, the management of configuration data was always a crucial task. The desire for a robust, flexible and transparent central data repository has a long history and is growing – especially with increasing availability of generic applications and the demand for flexible and frequently modified installations.

Nevertheless a major evolutionary break-through is not yet visible: natural starting point (and still frequently used approach) of storing configurations in named tables within file system trees or even hard-coded within different applications is easy to begin with and good for test-stands. By entering the maintenance phase of a control system in production uniqueness and interrelationships of the various data blocks become important. These requirements are best captured using a RDB allowing to store both data and their relations.

For most control systems the primary and quasi obvious model for configuration data is based on the device concept. Typical classification of devices is mostly expressed in similar naming-conventions – at BESSY this convention is:

$$DEVICE := \{member\}[\{index\}]\{family\}[\{counter\}][\{subdomain\}]\{domain\}\{facility\}$$

From the CDEV generalized data exchange model it is known, that the control system data exchange API can be mapped to components like *device*, *message* and *attribute*. For the EPICS-based control system at BESSY the corresponding communication entities, called process variables (PVs), are composed like:

$$DEVICE:signal[:subsignal][.FIELD]$$

At BESSY the relational database (RDB) as the primary data-source for control-system configuration management has also been built around the *device* concept. Configuration data are supplied by the RDB to the applications using instantiation of templates, retrieval scripts or direct RDB access. Applications range from I/O set-up description to modelling programs (see [1]). The DB part of the configuration management system consists of ~350 tables and ~200 views, split into sets of

tables and views per application or device-class. In summary the RDB approach has proven to be very reliable and generic. It is useful to have all configuration-parameters in a common, network-wide available and platform/system-independent data-store. The existing configuration-management system at BESSY is in operation since about 1996 and has proven to be an indispensable system.
Although the RDB data-model is very powerful and tailored to the needs of the corresponding applications, the whole system revealed a number of shortcomings that became increasingly hard to solve:

- Collision-free integration of new device-classes (consistent modifications of existing device-classes, composite systems) is getting difficult
- Complexity of programs/scripts converting the database-contents into actual configuration files is growing
- For PV's information in the RDB data-model is not complete, only the device name parts and some relations to specific signals are stored, i.e. a complete list of control-system signals cannot be produced by an RDB-query
- Applications that are not specific to any device-class (e.g. alarm-handler, archiver...) can hardly be configured using the existing RDB data-model

RELATED WORK

Within the EPICS-community there is development that started at APS/ANL to provide a *de*-scriptive ("what-do-we-have", re-active, as-is) system IRMIS[2]. IRMIS reads configuration-files for various applications (using *crawlers*) and feeds the configuration-data into its RDB data-model. Having done that, IRMIS provides an *as-is* snapshot of the actual productional control-system. The IRMIS/APS RDB configuration analysis of currently active equipment control is centred around a 3-fold hierarchy: *housed by*, *controlled by*, *powered by* and easily browsable with a Java client.

Meanwhile several institutes joined the IRMIS-collaboration (APS, SNS, SLAC/SPEAR, D0/FERMI, TRIUMF, DESY, BESSY, Diamond, TJNAF) helping to make IRMIS a portable system and trying to find a way to integrate existing site-specific RDB data-models used to generate configurations.

While IRMIS is a very useful toolkit to look at the installed and running control-system, the problem of a generic model to create all configurations using the RDB as the primary data-source is still unsolved ([3], [4] and [5]). Within the IRMIS-collaboration there's currently work in progress to isolate a common RDB data-model *(rdbCore*) that is site-independent and provides portability of IRMIS itself. *rdbCore* can then as well be used as a base component for a *pre*scriptive RDB data-model used to create configuration files.

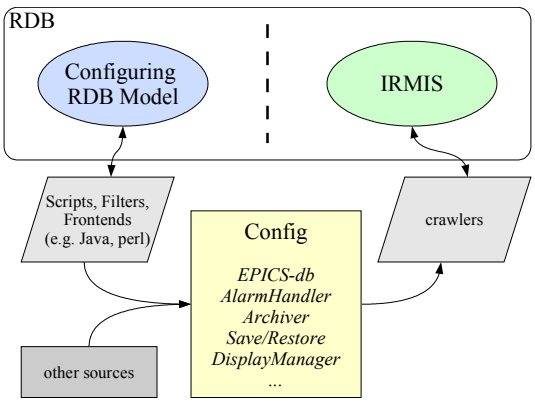

Figure 1: common installation using IRMIS

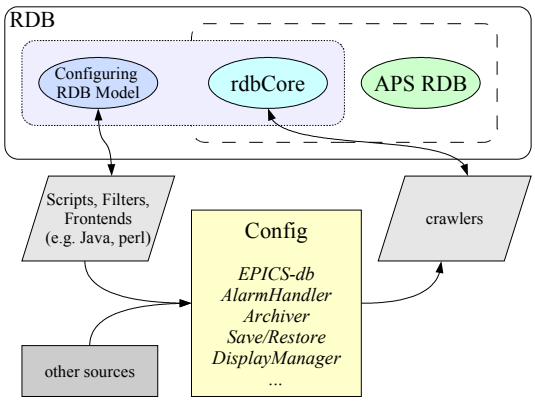

Figure 2: integrated config. system using rdbCore

MIGRATION PATH

A decision has been made, to develop a second generation RDB data-model capable of solving the shortcomings of the existing system, without losing the advantages. This RDB data-model is planned to be still a *pre*-scriptive ("what-do-we-want-to-have", pro-active, generating) system able to create all configuration files.

The new RDB data-model is designed to be appropriate to

- hold all parts of the PV-name.
- solve structural issues like device hierarchies, composition, inheritance

- address polymorphism of data and their meaning:
  - a PS may be replaced by another brand – same device, but differing I/O-specs
  - alarm-limits may depend on optics / machine operation status
  - correction-patterns may depend on operation mode
- move hard-coded information from generating scripts into data segments of the RDB
- transfer all existing data from the old RDB, homogenize structures

The proposed new RDB structure will be adequate to stream-line our *Configuring RDB Model* (Figure 1), adjust it to any future needs and accommodate any emerging *rdbCore* (Figure 2).

ATOMIC ENTITIES: GADGETS

The proposed RDB data-model was developed trying to catch the requirements of a controls-system based on the toolkit of EPICS-base and the commonly used contributed generic applications:

1. Support of the disparate subsystems of an EPICS control system context
   - EPICS .db, .template and .substitution files
   - Alarm-handler configurations
   - Archiver configurations
   - Save/Restore configurations
2. *Not* device oriented – enable to store entities of information, that are not necessarily dependant on a device
3. Represent hierarchies of arbitrary flavour

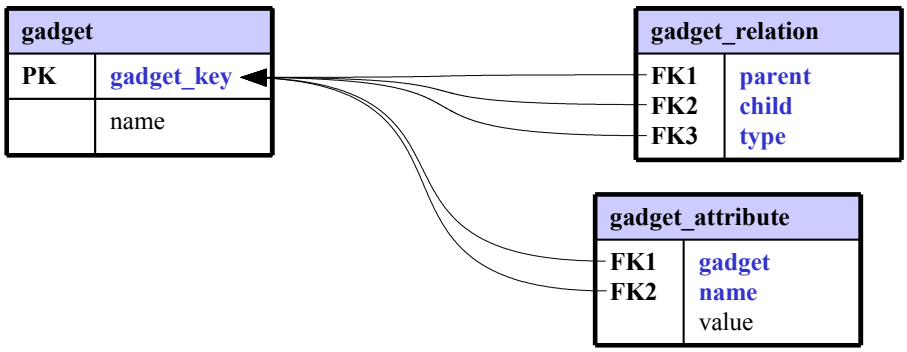

Figure 3: simplified ER-diagram of the atomic entites establishing the RDB data-model

The atomic elements of the resulting RDB data-model look very abstract and fairly simple, but cover those requirements. They are capable of representing directed acyclic graphs with named nodes and name/value-pairs attached to those nodes. The edges in these graphs are typed to distinguish between them. The RDB data-model consist of just three tables:

- *gadget* – a gadget represents a node in a graph. It carries no further information but a *name*.
- *gadget_relation* – a gadget-relation represents an edges in a graph. Identifying one gadget the *parent* and another gadget the *child* of the relationship models the direction. The third gadget used in a gadget-relation denotes the *type* of relation. Arbitrary types can be defined and used.
- *gadget_attribute* – a gadget-attribute is a *name/value*-pair connected to a gadget. For now, the *value* is a string that has no further restrictions. The *name* is again a gadget.

Restrictions

To unambiguously identify gadgets, there are some restrictions:

1. There can be at most one *gadget_relation* with the same *parent*, *child* and *type*
2. There can be at most one *gadget_attribute* with the same *gadget* and *name*
3. All root-gadgets (gadgets, that are not child in any relation) must have distinct names

where 1. and 2. are assured by unique-constraints in the RDB while 3. is checked by the API.

With these tables and following the restrictions it is possible to represent arbitrary directed graphs with named nodes and edges.

Advantages

- Relations enable construction of hierarchies
- Every gadget may be member of any number of hierarchies
- Hierarchies of gadgets with name/value-pairs connected to the gadgets open the possibility to implement overloading
- Linking hierarchies using relations enable creating re-usable hierarchies and thus make inheritance and prototyping possible

Additional Features

Every application, that wants to actually modify contents of the RDB is forced to provide an *application-name*. Any entity (*gadget*, *relation* or *attribute*) created or modified by an application is linked to that application-name. Only applications that provide the same application-name are permitted to modify or delete those entities.

An application can also use an application-name as a filter and will only "see" entities, that belong to this application in further queries.

COMPOSITION: THINK IN HIERARCHIES

The next organization level of the gadget based RDB are hierarchies:

- Hierarchies consist of *gadgets* with sets of name/value-pairs (*attributes*) attached
- Separate groups of information are modelled in separate hierarchies
  - these hierarchies get linked to combine information from different groups of information
- Hierarchies allow to inherit and/or overload default-values of attributes. Values of attributes are inherited and can be overwritten while descending or ascending the hierarchy (see Figure 4).
  Attributes and values will propagate from child → parent or from parent → child, depending on the way, the application retrieves the data
- Some hierarchies will serve as templates (re-usable class-definitions) for instantiation in other hierarchies to inherit a sub-hierarchy (see Figure 5).

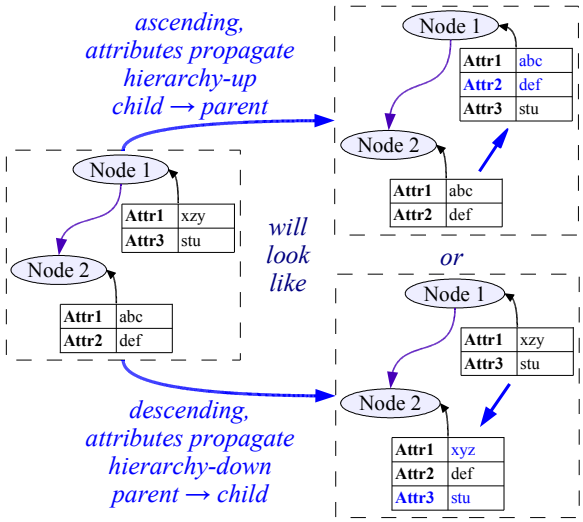

Figure 4: Overloading and Inheritance of Attributes

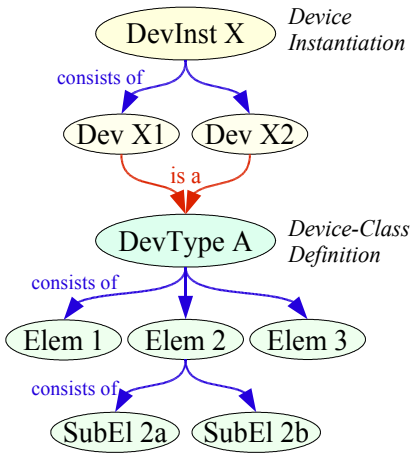

Figure 5: Template Instantiation

IMPLEMENTATION

The current implementation is done using Oracle 9i. The simple RDB data-model is accessed using an API that is currently implemented in PL/SQL and uses Oracle-specific features (hierarchical queries using *CONNECT BY* and *START WITH*, available in Oracle 9i and above). The API covers functions to

- Initialize the package
- Add/modify/delete gadgets, relations and attributes
- Navigate through hierarchies ("find" certain gadgets)
- Identify a gadget using a path, i.e. specifying the relations to parent and ancestors
- Find root-gadgets with regard to a certain relation-type
- Finding children/parents with regard to a certain relation-type
- Collect attributes on a given path

Examples

Given the hierarchy in Figure 6, **A**..**H** are gadgets representing nodes in the hierarchy and **T1** as well as **T2** are gadgets representing types of relations. Relations are represented by arrows and the direction is parent→child.

- *get_roots('T1')* returns the gadget *A* because *A* has children with relation-type *T1*, but no further parent with this relation-type
- *get_children('C', 'T1')* returns the gadget *D*, because it is the only child of *C* with relation-type *T1*
- *get_gadgets('/A/[T1]/%%/[T2]/%')* returns the list *{F, G, E}*. The path should be read from the end: *find all gadgets having a parent with relation 'T2', that again has any number of ancestors related with type 'T1', whose parent is the root-node 'A'.*
  In a path, *%* matches one gadget in the hierarchy while *%%* matches any chain of gadgets in the hierarchy.

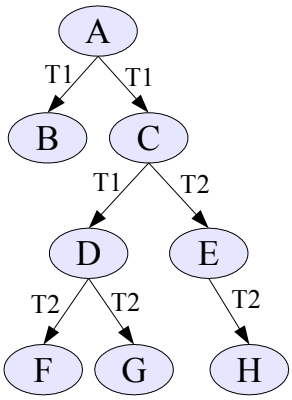

Figure 6: Sample Hierarchy

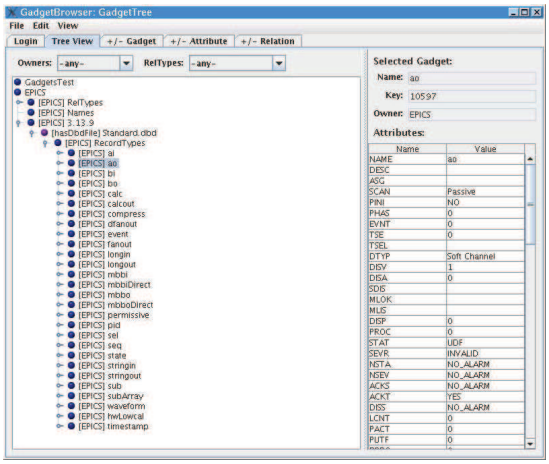

Figure 7: Java-based GadgetBrowser

CURRENT STATUS

While the DB-structure is simple by nature, the API is the mandatory way to get data into the RDB. The PL/SQL-based API has been finished and bindings for Java, perl and Tcl have been written. A simple browser has been written in Java (Figure 7). This browser is also capable of modifying values of attributes and will be extended to perform all basic functions provided by the API.

Furthermore, the very first applications, that will create EPICS-db files from the new RDB data-model, are currently being developed. During this process, the RDB data-model (including the PL/SQL-API) as well as the applications are improved and fine-tuned, based on the experiences made and the evolving requirements. An example use-case is shown in [6].

SUMMARY AND CONCLUSION

Powerful structural elements, navigation methods and modification tools have been developed defining a second generation RDB-based configuration management environment. From the microscopic view applicability, advantages and improvements of this approach are promising. Larger scale properties will become visible as the process of data migration, extension and integration proceeds.

REFERENCES

[1] T. Birke et al.: *Relational Database for Controls Configuration Management*
IADBG Workshop 2001, San Jose
[2] C. Saunders, D. A. Dohan: *The IRMIS Object Model and Services API*
ICALEPCS 2005, WE3A.1-6O
[3] N. Arnold: *Relational Database Collaboration @ APS & SNS*
EPICS-Meeting 2004, Tokai, Japan
[4] R. Chestnut: *RDB Issues @ SLAC*
EPICS-Meeting 2004, Tokai, Japan
[5] T. Birke: *BESSY Configuration Management, Plans & Wishes*
IRMIS Collaboration Meeting 2005, APS/ANL, Chicago
[6] T. Birke et al.: *Use Case - Configuration Management with a generic RDB Data-Model*
ICALEPCS 2005, PO1.079-7