EPICS Configuration with a Relational Database

T. Birke, B. Franksen, P. Laux, R. Müller
(BESSY)

Starting Point

- EPICS consists of a number of tools
- each of them needs configuration to do useful things
  - IOCs need their runtime databases configured
  - clients like medm, alh and archiver need configuration files that tell them which channels to monitor
- these configuration data are *not* independent
- this tends to lead to inconsistency

RDB: A Solution?

- at BESSY we use a relational database (Oracle) to store and manage EPICS configuration data
- currently, each type of device is handled by its own set of tables
- control system structure (hierarchy of devices) is represented by database structure, i.e. tables & their dependencies
- scripts generate substitution files for (usually simple) rtdb templates

Deficiencies

- new structures require new set of tables or changes in existing tables
- hard to factor out common configuration values for groups of devices
- many things interesting for client configuration are hidden inside db-template files
- devices usually consist of a number of *signals*, i.e. simple "molecular" building blocks, which we cannot easily reuse
  - typical signals are readback, setpoint, status bit, command word, ...

The Idea: Redesign

- produce a common generic framework for all devices and applications
- map *control system structure* not to database structure but to *database data*, to enhance flexibility & extendibility
- specify configuration data not only for individual devices but also defaults for whole groups (families) of devices, to remove redundancy

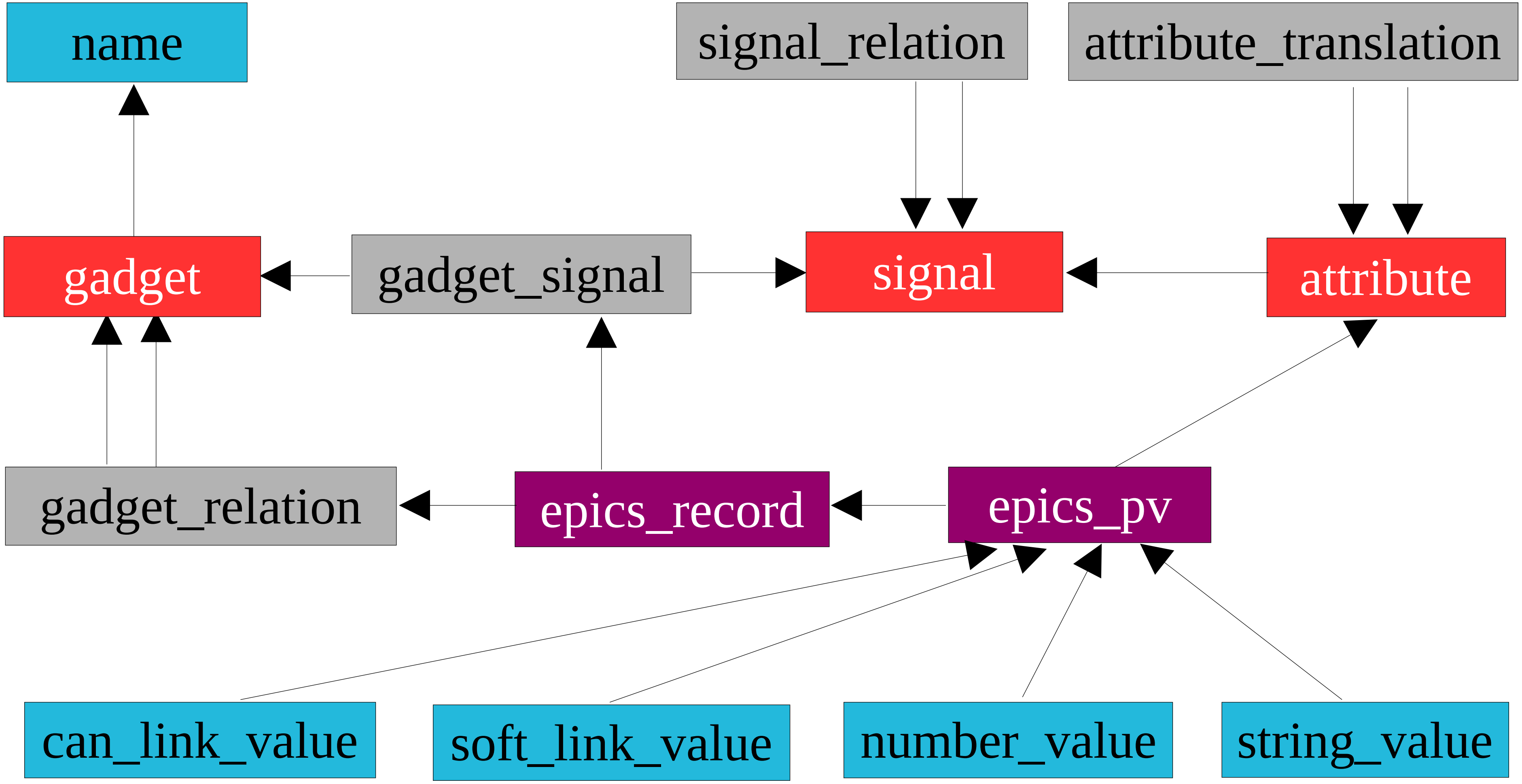
Further Goals

- consistent and complete model of the control system down to single channels, including
  - global name service
  - global repository for re-usable signal definitions
- extendable and re-configurable hierarchy of devices
- unified data source for all EPICS applications at every level of abstraction
  - e.g. high level (client) and low level (rtdb) data
- *all* configuration files generated from RDB

The New Concept

- ...is quite general and abstract
- centered around the notions of
  - gadget: either a device or a family / group of devices
  - signal: the building blocks of gadgets
  - attribute: signals have a number of attributes
- structure is defined by the content of relation tables
- the notions of record and pv are generalized
  - can be *virtual* if associated with abstract (group) gadget

Structural Overview



name

signal_relation

attribute_translation

gadget

gadget_signal

signal

attribute

gadget_relation

epics_record

epics_pv

can_link_value

soft_link_value

number_value

string_value

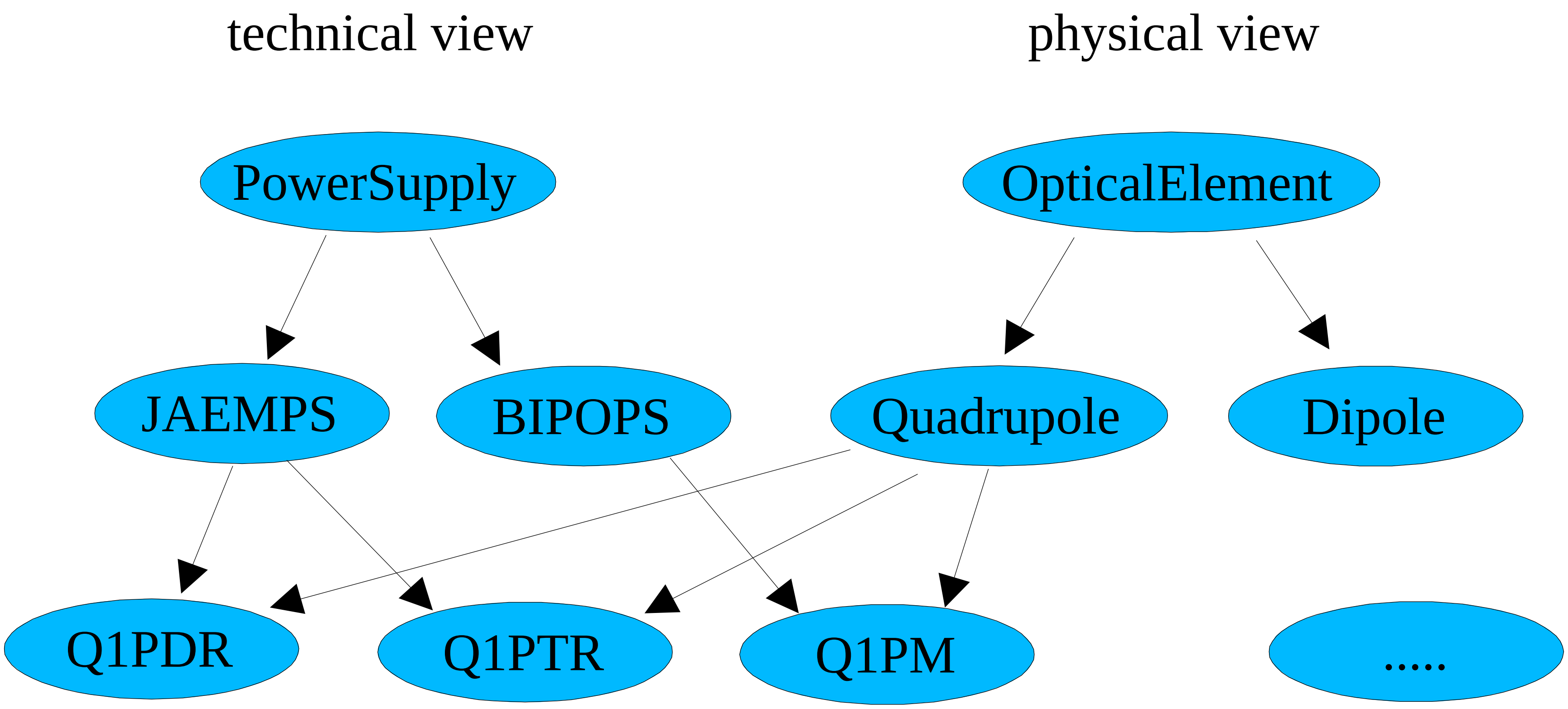
Names (Named Objects)

- each concrete device (or similar entity) has a *name* conforming to our internal naming convention
  - the name gives compressed information about position and type of object (this is not new)
- all names / named objects are held in one big flat table
- conformance to naming convention is checked by automatic db constraints

Gadgets

- gadgets are objects or object groups / families
- form a hierarchical tree
  - the leafs are the named (concrete) objects
  - the higher level nodes provide grouping / abstraction
- independent abstraction trees may coexist
  - restriction: gadget relation paths must be unique
- gadget relations are re-configurable
  - they are held as data in a relation table

Example: Power Supplies as technical vs. physical entities



technical view

physical view

PowerSupply

OpticalElement

JAEMPS

BIPOPS

Quadrupole

Dipole

Q1PDR

Q1PTR

Q1PM

.....

Signals

- the building blocks of which devices are composed
  - roughly correspond to database template files
  - usually contain one - or few tightly interacting - records
- form a hierarchical tree, similar to gadgets
- carry not only structural but also semantical information
  - e.g. "an analog readback from a can bus io-card"

Attributes

- signals have a number of atomic *attributes*
  - roughly correspond to fields of record types or - for higher level signals - to the substitution variables of a database template file
- attributes of high level signals are mapped to those of low level signals by an attribute translation

Attribute Values

- these are the actual configuration data
- they correspond to
  - configuration data of a concrete record instance or
  - a variable substitution into some database template
- values are of different types and thus are kept in different tables
  - numerical, string, soft link, various hardware links, ...
- they are assigned to (generalized) process variables

Record Instances

- are identified by a combination of
  - a gadget-gadget relation (parent-child):
    - child gives the instance name
    - parent gives the signal type
  - a gadget-signal relation (parent-gadget contains signal)
- may be virtual (non-existent as an EPICS record)
  - i.e. if either child gadget of first identifying relation or signal of second identifying relation is not a leaf
- name is <gadget_name>:<signal_name>

Process Variables (PVs)

- are identified by a combination of
  - a record
  - an attribute (of a signal that belongs to the record's gadget)
- like records, PVs may be virtual (if record or attribute are not bottom level in their hierarchies)
- otherwise they exist as real CA channels with name: <record_name>.<attribute_name>

The Price

- the advantages of this new structure do not come for free:
- DCTs can no longer be used directly
- the model is *very* (too?) abstract
  - tables no longer maintainable by manual sql hacking
  - thus we need high-level tools and scripts
    - to fill and update the tables
    - to convert existing applications from template files to rdb
  - the plan is to develop generic web browser frontends and stand-alone command line scripts

Project Status

- tables & general structure are implemented
- views, frontends, and generic scripts still missing
- next steps:
  - test system with a new application
  - develop frontends & scripts in parallel
- dream: a graphical configuration tool with SQL backend