ASDEX UPGRADE CONTROL PARAMETER ORGANISATION AND CONFIGURATION MANAGEMENT

G. Neu, K. Behler, R. Cole*, A. Lohs, K. Lüddecke*, G. Raupp, W. Treutterer, D. Zasche, Th. Zehetbauer, and ASDEX Upgrade Team
Max-Planck-Institut für Plasmaphysik, EURATOM Association, Boltzmannstrasse 2, D-85748 Garching, Germany
* Unlimited Computer Systems, Seeshaupterstrasse 15, D-82393 Iffeldorf, Germany

Abstract

The AUG tokamak's plasma control and data acquisition system runs a variety of realtime applications (APs) to feedback control and monitor plasma properties

AP behaviour is governed by parameters which have to be obtained from external data sources: files, relational and object databases, or streams.

APs and parameter data sources may evolve independently over time. We present a mapping mechanism based on well-established web technology that effectively decouples the two evolution paths.

INTRODUCTION

The new ASDEX Upgrade plasma control system is a network of realtime controllers interconnected through distributed shared memory. Possibly interdependent application processes (APs) programmed in C++ can be freely allocated onto controllers running under VxWorks and interact by exchanging realtime signals [1]. The set of all APs on the controller network forms a system release.

Currently the system release includes APs for:

- position & shape feedback
- density and impurity feedback
- coil load and current monitoring
- shape monitoring
- I/O

In a configuration phase prior to a discharge the APs have to be supplied with specific parameters derived from external data. Parameter data may have to be extracted from various sources (Fig 1):

- files: e.g. gain matrices for position & shape feedback obtained from modelling algorithms
- streams: e.g. limits for thyristor & coil currents from the power supply PLCs
- databases: e.g. scales & offsets from peripheral hardware devices (sensors, ADCs, …)

External data is organized in data sets whose structure reflects that of hardware devices, physics model algorithms, but not necessarily that of an AP. Data from data sets may therefore have to be transformed (filtered, sorted, aggregated) into the form required by the AP.

AP parameters may be numerous and complex. Ensuring the integrity and validity of AP parameters can be a demanding task, but is an essential for correct AP operation.

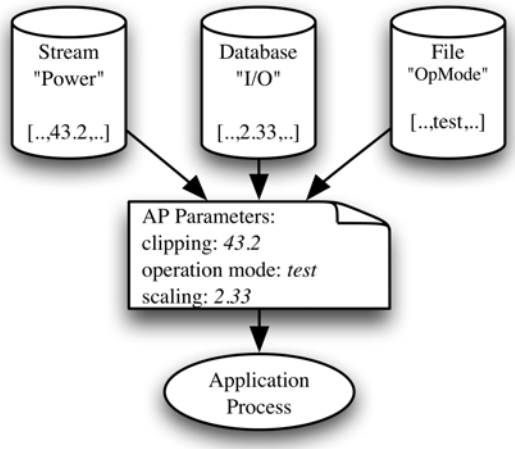

Figure 1: Assembling parameters from various sources

Moreover, we have to account for evolution of data sources and APs which do not necessarily occur in parallel.

We therefore need:

- a description language for AP parameters
- a mechanism for validating AP parameters
- a mapping of data from sets onto AP parameters
- a robust implementation of the mapping mechanism

SYSTEM EVOLUTION

System Release Evolution

A system release and thus the structure of parameters required by its application processes may change over time for many reasons:

- new APs are implemented to add functionality
- existing APs are replaced or enhanced for improved performance
- APs are coalesced to reduce communication overhead between them
- APs are split to allow parallel computation and reduce execution time

The driving forces behind these changes are new views on the control problem, the wish to implement new algorithms in the control system, and performance requirements.

Evolution of External Data Sources

Data sources and the data sets they provide also evolve:

- data previously available from a file is transferred to a database, served from a new streaming device, or vice-versa
- data sets are reorganized (divided, coalesced, …)
- new data sources are introduced
- new data sets adapted to changed experiment conditions are computed

The driving forces here are the availability of new devices, the reorganization of peripheral I/O hardware, new physical models, testing, or the availability of new media for data storage.

PARAMETER SERVER

Decoupling through Parameter Server:

Under the circumstances described above, keeping system releases and data sources in sync and guaranteeing the completeness an correctness of parameter data is far from trivial.

A good way of achieving the necessary decoupling is by introducing a parameter server which provides a uniform interface for the APs and performs the operations necessary to compile the AP's parameter sheets. (Figure. 2)

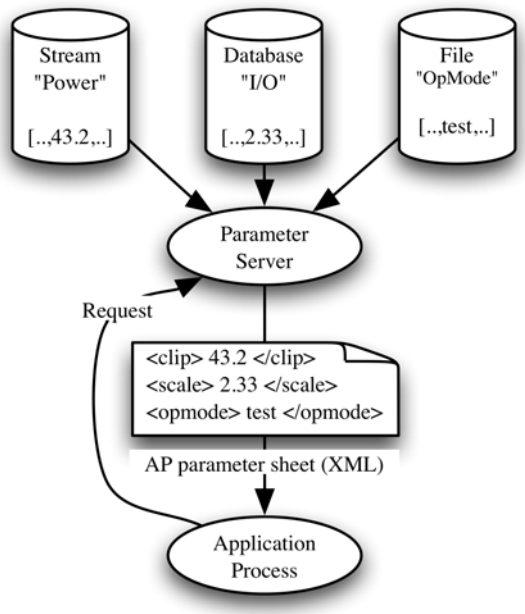

Figure 2: Parameter server assembles a parameter sheet

Switching between data sources then becomes completely transparent to the APs. The APs request their parameter data by identifying themselves to the server. The parameter server returns a parameter sheet of an AP-specific structure, which can be parsed, validated, and used to instantiate the parameter objects for the AP during the configuration phase.

XML

To implement our parameter server and specify the structure of parameter sheets we have opted for an XML-based solution.

The advantages of using XML to describe documents (such as AP parameter sheets) are obvious:

- standardised, human readable format
- possibility to formally describe document structure and constraints
- availability of parsers for a variety of platforms and programming languages
- availability of software for transforming into other document types
- availability of validators based on numerous description languages

We use XSD (XML schema definition [2]) to specify the content structure of AP parameter sheets. XSDs can be used to validate parameter sheets produced by the parameter server (e.g. by using SUN's multi schema validator) or generate specific editors which only allow the creation of valid parameter sheets (e.g. for creating test data sheets).

Apache Cocoon

A closer look reveals that the tasks of separating the parameter sheet structure from that of data sets are similar to those performed by web servers capable of delivering document content in different forms (HTML, WML, RTF, …)

From there it is a short step to attempt building the parameter server around a web publishing framework such as Apache's Cocoon [3].

Besides being freely available, Cocoon sums up numerous advantages:

- open source
- highly flexible
- easily testable
- modular
- extendable
- platform independent: based on Java and XML

Cocoon implements the concepts of separation of concerns (notably that of separating content from presentation) around components and pipelines. Components specialize on particular operations, and pipelines interconnect them to form a processing chain (Figure 3).

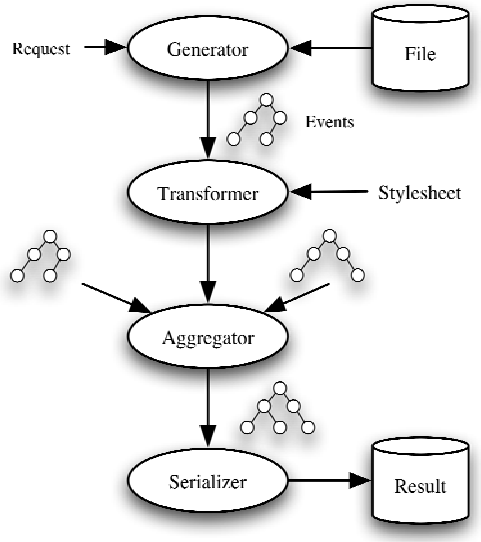

Figure 3: Cocoon pipeline

An XML document is pushed through a pipeline, and transformed in several steps. Every pipeline is identified by a matcher, begins with a generator, continues with zero or more transformers, and ends with a serializer:

Matcher:
- entry point for selection of a pipeline:
- associates the uniform resource identifier (URI) of a request with a specific pipeline of the sitemap

Generator:
- starting point for the pipeline
- delivers events, corresponding to XML elements and attributes, down the pipeline
- example: FileGenerator: reads a local XML file generates events and sends them down the pipeline

Transformers:
- get events and transform them to other events
- examples: SQL Transformers for database access, XSL [4] Transformers for adding information, filtering, sorting, renaming, etc.

Aggregator:
- gets results from several transformers and concatenates them into a new event stream

Serialiser
- end of the pipeline
- transforms events into binary or character streams for final client consumption

Cocoon comes with a set of ready-made matchers, generators, transformers, and serializers. Additional components can easily be defined and used in pipelines.

Components and pipelines are declared in an XML configuration description from which a web server is generated, the so-called sitemap. The sitemap allows to use a construction kit-like approach. Existing components can be hooked together without any required programming.

Using Cocoon to set up a parameter server

The following steps are necessary to generate a parameter server using Cocoon:

- Define a unique URI to identify each AP of a system release
- Write a schema (XSD) for each APs parameter sheet
- Write a matcher for each APs URIs in a system release
- identify sources for AP parameters and write generators and transformers to extract the relevant values from the corresponding data sets.
- aggregate the results from the transformations where needed
- use an XML serializer to create result parameter sheets
- insert one pipeline declaration with the above matchers generators, transformers, aggregators, and serializers for each AP
- check the correct operation by validating the result against the APs parameter sheet XSD and return the parameter sheet to the AP over HTTP.

CREATING PARAMETER OBJECTS FROM PARAMETER SHEETS

In our current implementation we have a specific parser for each AP XSD, which parses parameter sheets, and builds the parameter objects from them. This implies that modification of an AP's parameter structure requires manual adaptation of both the XSD against which parameter sheets are checked, and the parameter sheet parser itself.

In future, however, we are planning to use a tool, RogueWave's XML Object Link®, that generates validating parsers directly from XSDs.

SUMMARY

We use proven public domain web technology to solve the problem of creating a parameter server for an evolving distributed realtime control system. The server fully decouples evolution paths of application processes and datasources for the AP's parameters. Specific parsers instatiate parameter objects from the Cocoon-generated datasheets.

REFERENCES

[1] W. Treutterer et al., "Software Structure and Realtime Signal Exchange for the ASDEX Upgrade Tokamak Control System", this conference
[2] http://www.w3.org/TR/xmlschema-0/
[3] http://cocoon.apache.org/2.1/
[4] http://www.w3.org/TR/xslt20/