WEB-BASED EXECUTION OF GRAPHICAL WORKFLOWS: A MODULAR PLATFORM FOR MULTIFUNTIONAL SCIENTIFIC PROCESS AUTOMATION

E. De Ley, D. Jacobs, iSencia Belgium, Ghent, Belgium
M.Ounsy, Synchrotron Soleil, France

Abstract

The Passerelle process automation suite offers a fundamentally modular solution platform, based on a layered integration of several best-of-breed technologies. It has been successfully applied by Synchrotron Soleil as the sequencer for data acquisition and control processes on its beamlines, integrated with TANGO as a control bus and GlobalScreen™ as the SCADA package. Since last year it is being used as the graphical workflow component for the development of an eclipse-based Data Analysis Work Bench, at ESRF.

The top layer of Passerelle exposes an actor-based development paradigm, based on the Ptolemy framework (UC Berkeley). Actors provide explicit reusability and strong decoupling, combined with an inherently concurrent execution model. Actor libraries exist for TANGO integration, web-services, database operations, flow control, rules-based analysis, mathematical calculations, launching external scripts etc.

Passerelle's internal architecture is based on OSGi, the major Java framework for modular service-based applications. A large set of modules exist that can be recombined as desired to obtain different features and deployment models. Besides desktop versions of the Passerelle workflow workbench, there is also the Passerelle Manager. It is a secured web application including a graphical editor, for centralized design, execution, management and monitoring of process flows, integrating standard Java Enterprise services with OSGi.

We will present the internal technical architecture, some interesting application cases and the lessons learnt.

INTRODUCING PASSERELLE

Passerelle[1] is a component-based solution assembly suite, developed and distributed by iSencia Belgium since 2002. It is based on an integration and on extensions of Ptolemy II[2] (Univ. Berkeley), OSGi/Equinox & other OS libraries. Passerelle comes with an execution engine, enhanced actor development API, reusable actor libraries and several graphical model editors.

Passerelle is already used for several years in “production-mode” at different sites. Two important reference sites are :

- Belgacom, the main Belgian telecommunication provider, where it is used to automate diagnosis and repair processes for the customer help-desk operators, and for field technicians.
- Synchrotron Soleil in France, where Passerelle is used as sequencing engine for beamline experiments.

Since last year, the core Passerelle modules are provided in open source, together with an eclipse-based graphical editor, at the eclipse labs hosted by Google.

ACTOR-BASED DEVELOPMENT

Passerelle inherits many core concepts from Ptolemy II, the main ones being :

- actor-based development
- using graphical hierarchical models for defining executable solution assemblies
- separation between the “topology” of the actor assemblies and their runtime semantics by picking alternative *Director* components

Passerelle/Ptolemy actors and sequences are a direct implementation of the well-documented *pipes-and-filters* architecture pattern, see for example [3]. Such an architecture promotes component-based solution designs with strong decoupling. Each actor in a complete process has a single responsibility that must be fulfilled based on data in messages received on the actor's input port(s). Results must be sent via the actor's output port(s).

Runtime semantics, or “models of computation”, are determined through the usage of one of the “domains” provided by Ptolemy II. Passerelle is based on the *Process Networks* domain, which is a good basis for data-flow and process engines that internally require asynchronous behaviour and transparent actor-based concurrency.

Both actors and directors are implemented as Java classes, building on a rich API with base classes and utilities.

All of this results in a system that promotes designing and developing with reusability in mind. Actors are the basic level of reusable components and can be picked from actor libraries. There are actors for all basic flow control elements (branching, filtering, routing, loops, error handlers,...), but also libraries of domain-specific actors like Tango device control, database querying and updating etc.

Sub-processes can be easily stored in the library by end-users and can be reused between models.

Additionally, the clear split between actor development, solution assembly and execution environments supports different roles in solution delivery and maintenance for designers, developers, users and administrators.

ADVANCED CONCURRENCY FEATURES

Automation of non-trivial processes quickly confronts a solution developer with complex technical requirements. How can we control many simultaneous activities? How to assign system resources for a combination of long-running tasks and fast tasks? How to cater for peak loads in an optimal way?

The combination of asynchronous, event-driven internal processing, with the transparent concurrency that comes for free with the actor model means that neither the actor developer, nor the model designer, need to care too much about such technical complexities. Passerelle processes are automatically concurrent. Each actor can perform its work in an own thread that is managed internally by the engine. Each interaction between actors is buffered, so temporary overloads of an actor do not impact the others.

The philosophy behind the Passerelle engine, which is made possible thanks to the core Ptolemy design concepts, is that both an actor developer and a model designer should just focus on the desired functionalities. The complexities to ensure an optimal execution are hidden, and can be maintained and improved independently.

APPLICATION DOMAINS FOR SYNCHROTRONS

Passerelle actors and sequences can be used to automate all kinds of processes. By combining the right execution model with the right set of actors, the same process automation platform can handle e.g. :

- sequences for data acquisition and control
- workflows for data analysis
- autonomous monitoring and alarming
- scheduling batch processes
- …

A COMPLETELY MODULAR AND DYNAMIC PLATFORM

The Passerelle engine has been integrated in a complete platform for Java enterprise solutions with rich browser-based user interfaces. This has been designed from the ground up as a fundamentally modular system, through the usage of OSGi. This helps in several respects :

- strict control of intermodular dependencies with versioning
- OSGi bundles provide active modules with well-defined life-cycles
- support for dynamically updating/extending operational systems without downtime
- a very clean and performant service-based architecture

Just as for actors, OSGi promotes a model where each module has a limited responsibility, with a clean public interface and low coupling.

The Sherpabeans Java web framework integrates the following core technologies within an OSGi architecture :

- Apache Wicket for the web view layer
- JPA, Hibernate, eclipse link for persistence
- many smaller libraries

SherpaBeans itself is a collection of OSGi bundles that can be recombined at will. Some important modules are :

- scheduler for background jobs
- role-based security
- asset repository
- publish/subscribe notifications
- OSGi bundle upload and lifecycle management
- ...

This architecture makes it possible to deliver a basic Passerelle Manager platform with all core services, including a *web-based graphical model editor* (see Figure 1 below), an execution engine, some standard actor libraries and a complete web-based administration interface.

This system can then be extended in different ways by uploading extra modules as OSGi bundles (e.g. with new actors or new management features), designing and running new sequences etc.

ACTOR DEVELOPMENT WITH OSGI

In cases where the automated processes must be operational 24x7, even a short interruption to stop a sequence, and start a modified version, is problematic. Thanks to the dynamism offered by OSGi, many types of adaptations can be done without downtime.

A typical example is upgrading the logic of an actor implementation, e.g. for a bug fix or adapting to upgraded backend APIs etc.

To prepare for easy and dynamic maintainability, it is good practice to design actors in 2 layers :

- a service layer, containing the real logic behind a standardized task-based API that can be used by an actor
- a simplified actor, that basically feeds received data to the right service, collects its results and generates output messages from them

This results in actors that act as “binding” between a control layer (defined by the process model) and the service layer.

In an OSGi-based environment, each service can be offered by a dedicated bundle. When the bundle becomes active, the service is registered and is available for usage. When the bundle is deactivated or uninstalled, the service disappears. On the “client”-side, the actor can lookup the available service implementation(s), is notified about any changes in the status of the service, and can even pick the best one when multiple implementation versions would be available.

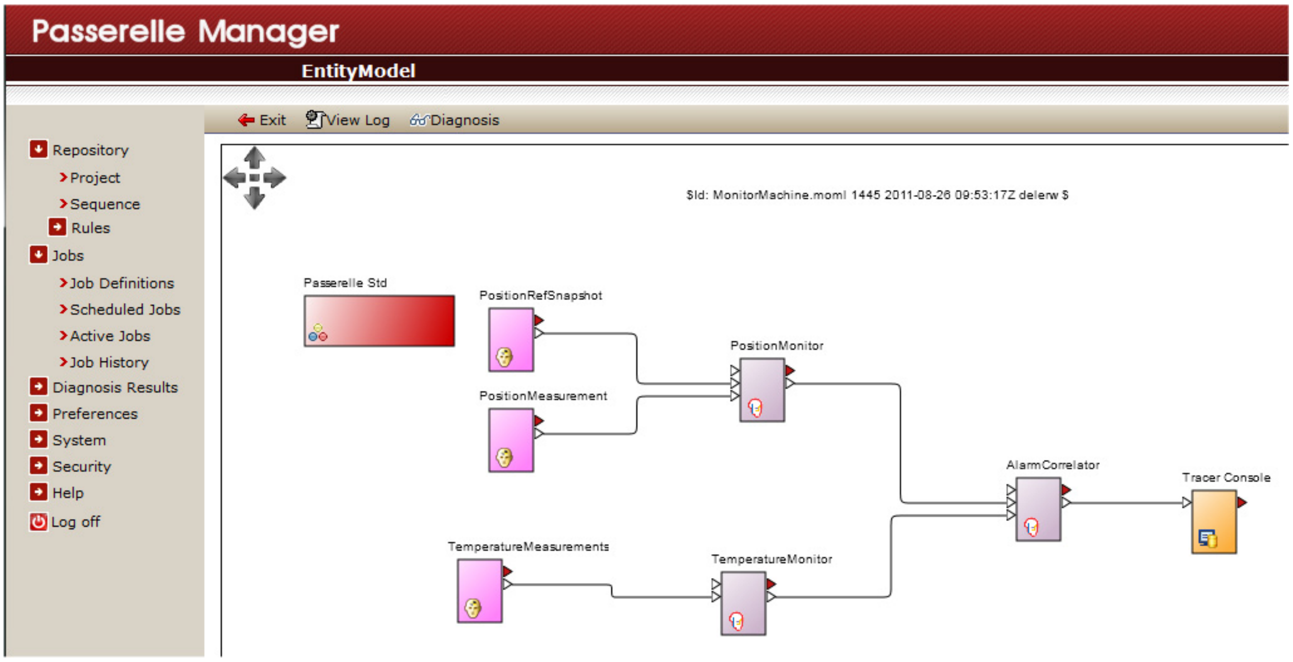

Figure 1: Screenshot from the web-based model editor in Passerelle Manager.

THE RESULT : EXTREME FLEXIBILITY FOR PROCESS AUTOMATION, WITHOUT DOWNTIME

Via the Passerelle Manager's web UI, it is possible to manage the lifecycle of each service bundle, to upload new versions etc.

The end result is a fully secured and dynamic platform to design new or updated processes via the browser, to upload new or updated actor and service implementations, to schedule process execution, consult execution traces etc.

COMBINING MODULES IN DIFFERENT PACKAGING OPTIONS

Through picking and recombining subsets of the available OSGi bundles, several solution packages can be obtained, with almost complete reuse of the core Passerelle bundles.

By using OSGi's service-based architecture, with a clear split between interface bundles and implementation bundles, it is also possible to provide several replaceable implementations of core service interfaces. For example, the asset repository has two implementations :

- the “enterprise” version, with persistence in a relational database, to be used inside the Passerelle Manager
- a simple file-based version, to be used with desktop workbench/IDE

Besides the Passerelle Manager, the following packages are possible :

- a standalone eclipse based Passerelle workbench
- the Passerelle eclipse workbench as plugins in larger workbench undertakings, like ESRF's Data Analysis WorkBench (DAWB) [4].
- an OSGi version with a Swing HMI (Figure 2 below)
- a plain Java-main Swing HMI, as OSGi bundles are finally also useable as plain jars...

EXAMPLE : ACTORS FOR DATA ACQUISITION VIA WEB SERVICES

In many enterprise environments, both scientific or non-scientific, the usage of web services is common, e.g. via SOAP or REST. Automated processes often include steps to obtain data from other software systems (often denoted as “backend systems”) via such web services.

The preparation of a web-service request, and the interpretation of the received response typically involves tedious XML generation/parsing work. In many cases it's preferred to use frameworks like Apache Axis, JAXB etc to generate a Java binding.

In such a context, it is good practice to create an OSGi bundle for each required backend service. This bundle encapsulates the following responsibilities :

- accept request data from the client/actor
- prepare the outgoing request to the backend
- send the request, potentially with some safety measures to prevent overloading the backend
- collect the response or errors
- store the results and timing information etc in a database
- return the results to the client/actor

The actor layer becomes quite simple does not need to care about web service protocols, data persistence etc.

Such a design improves testability and maintainability.

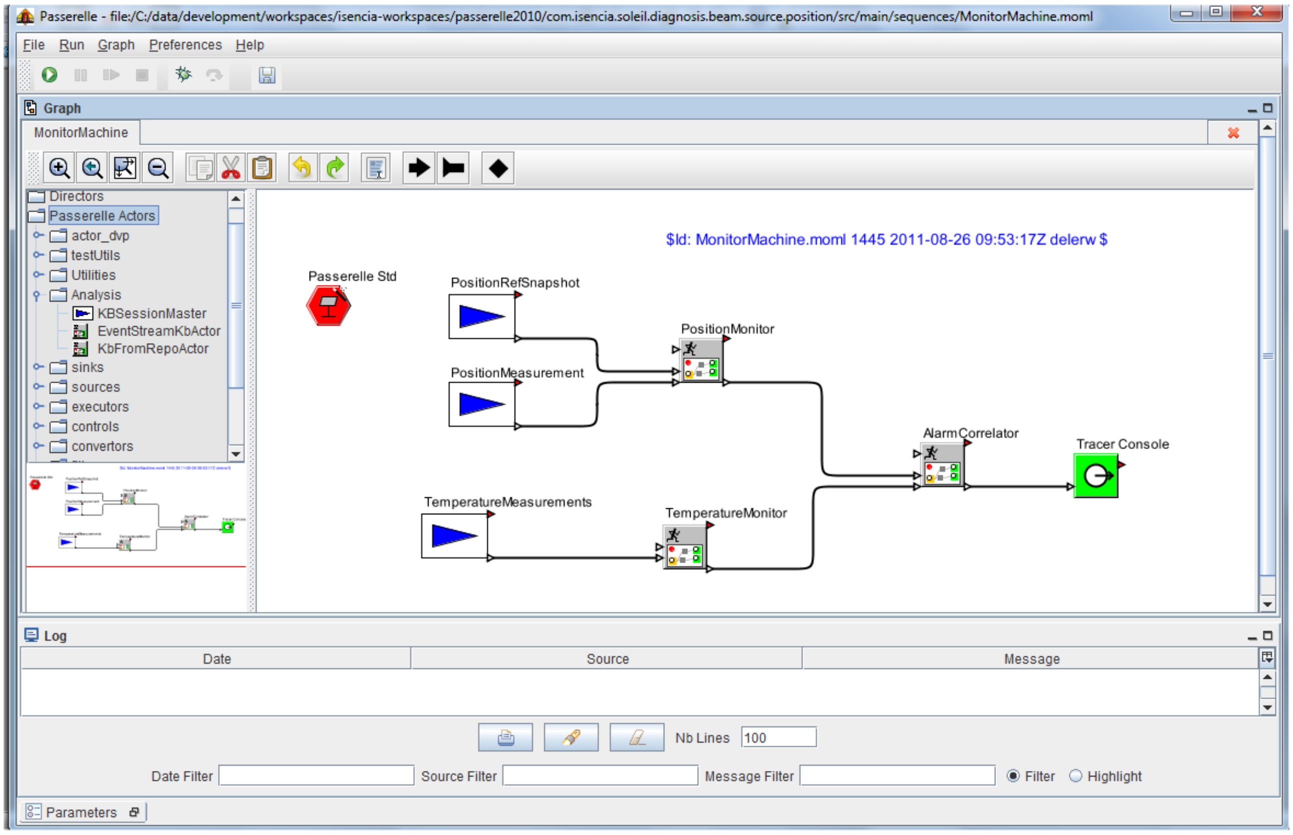

Figure 2: Screenshot from the Swing-based HMI/editor.

When the backend needs to change its web service, or the protocol changes for any other reason, or extra technical features like timeout management etc need to be added, this can all be done via an upgrade of the service bundle(s), without requiring a downtime.

CONCLUSIONS

Through a fundamental modular and componentized approach, it is possible to obtain advanced platforms for process automation.

By rigorously following good service-based design approaches in the context of an OSGi-based platform, a generic platform providing a complete tools set for designing, running and maintaining process models, consulting execution traces etc, becomes the starting point for a robust system that can be extended and updated without requiring downtime.

REFERENCES

[1] Passerelle project information : http://www.isencia.be/services/passerelle and http://code.google.com/a/eclipselabs.org/p/passerelle/

[2] Ptolemy II project information : http://ptolemy.berkeley.edu/ptolemyII/

[3] F. Buschmann et al., Pattern-oriented software architecture – Volume 1

[4] ESRF's DAWB project information : http://www.dawb.org