iSencia

EMPOWERING BUSINESS INFORMATION

iSencia, Passerelle

& EDNA

www.isencia.com

Agenda

- Who's iSencia?
- Software offerings
- Some Passerelle use cases
- EDNA and Passerelle

Who's iSencia?

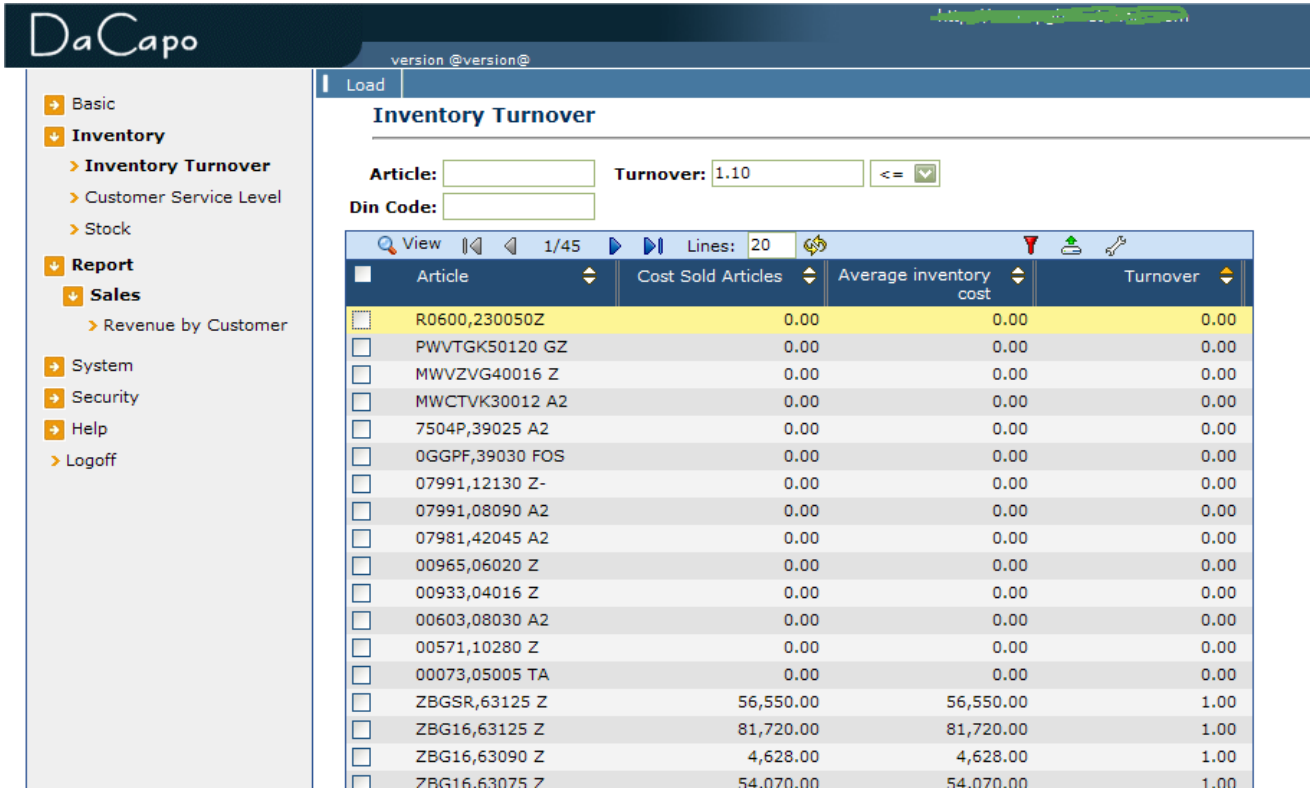
- Software solutions provider with offices in Ghent (Belgium) and Barcelona
- We build & maintain integrated solutions based on Java open source frameworks enhanced with own additions & improvements
  - Advanced web/intranet solutions
  - Middle tier, messaging & process engines
- Customers in different domains
  telecom, health-care, non-profit, science, logistics, ...

Software offerings

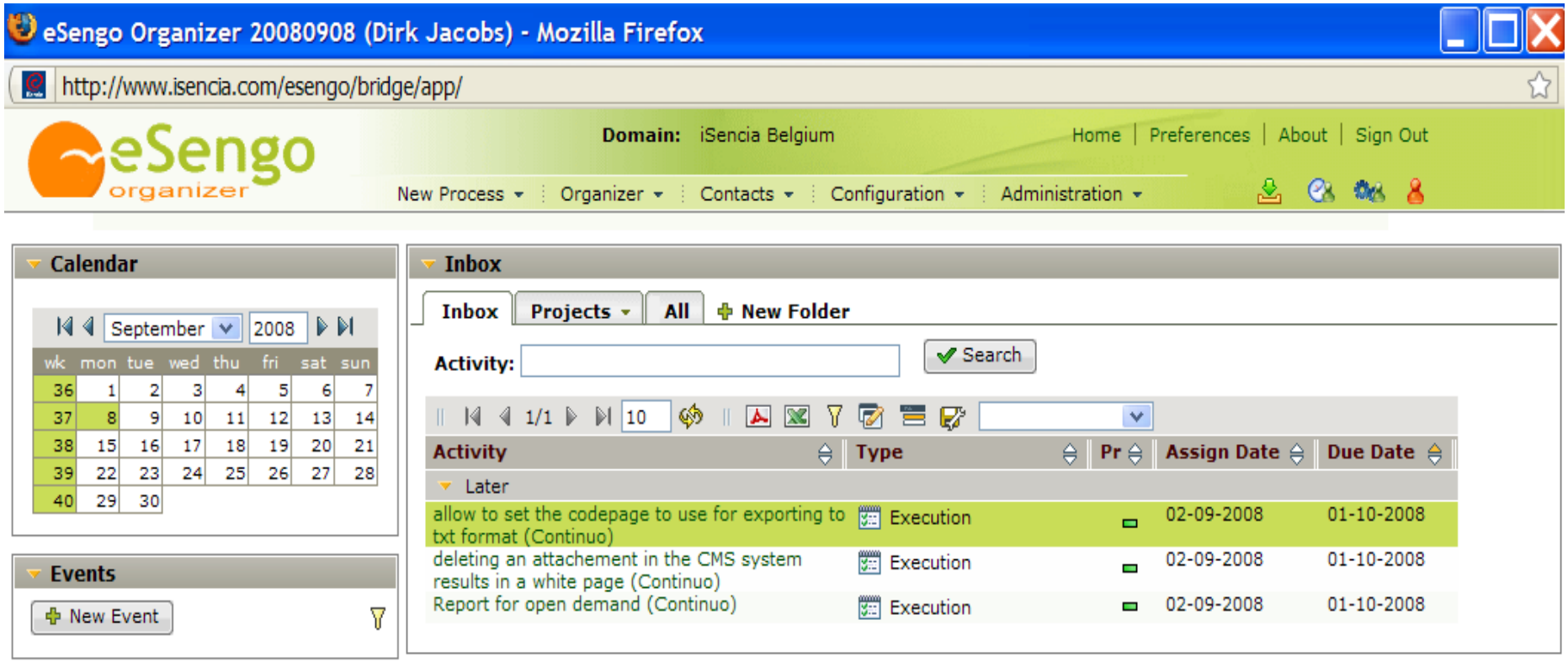
SherpaBeans

- modular, services-based Java-centric web framework
- avoids explosion of XML-configuration-files
- rich web UI with desktop-like components and interactivity
- integration and many enhancements of J2EE standards, Hibernate, Wicket, OSGi, ...

SherpaBeans samples



SherpaBeans samples - eSengo



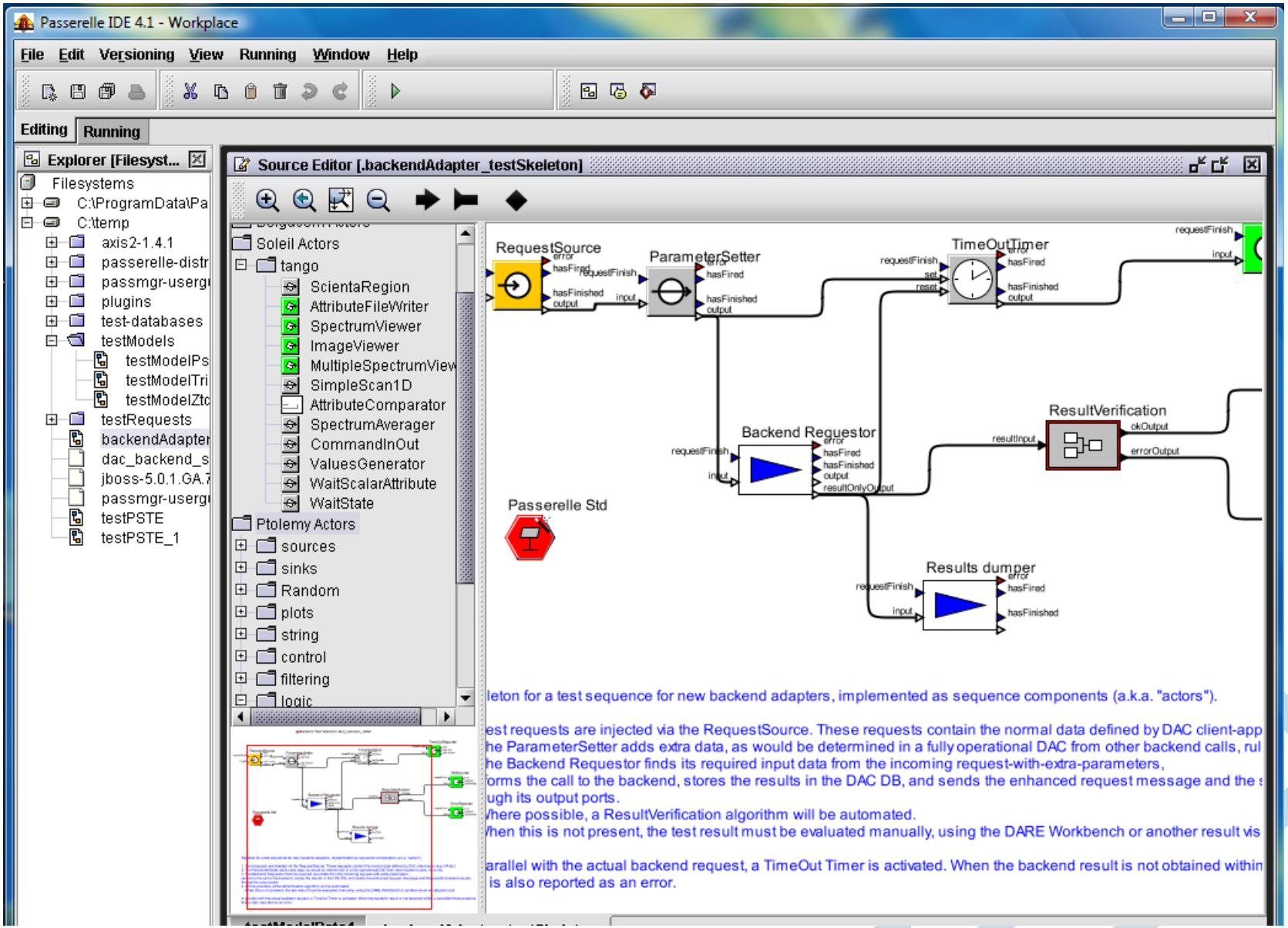
Passerelle

- Solutions suite for model-based solutions assembly
- Graphical model defines flow and execution semantics
- Based on picking, configuring and interconnecting reusable components (actors) from a library
- Support different roles in solution delivery & maintenance (designers, developers, users, administrators)
- Tools and infrastructure for
  - Maintaining component libraries
  - Assembling solutions
  - Testing & deployment

Some important concepts

- Flows / processes /sequences
  Support for all typical flow concepts (parallel branches, loops, …)
- Actors with multiple named input and output ports
  Each port has specific, identifiable function
  Actors have own execution thread
- Standardized internal messages
  System headers, App headers, App data
  Supporting wide variety of data content types
  Automated data content type conversion infrastructure

Passerelle



Benefits

- Designed for reuse
  - For solution assemblers (graphical solution assembly)
  - For component developers (actor dvp framework)
- Pipe-and-filter architecture
  - Promotes clear definition of component responsibilities
  - Promotes low coupling of component implementations
- Designed for extensibility
  - Well-defined, simple component development API
  - Ready for different application models

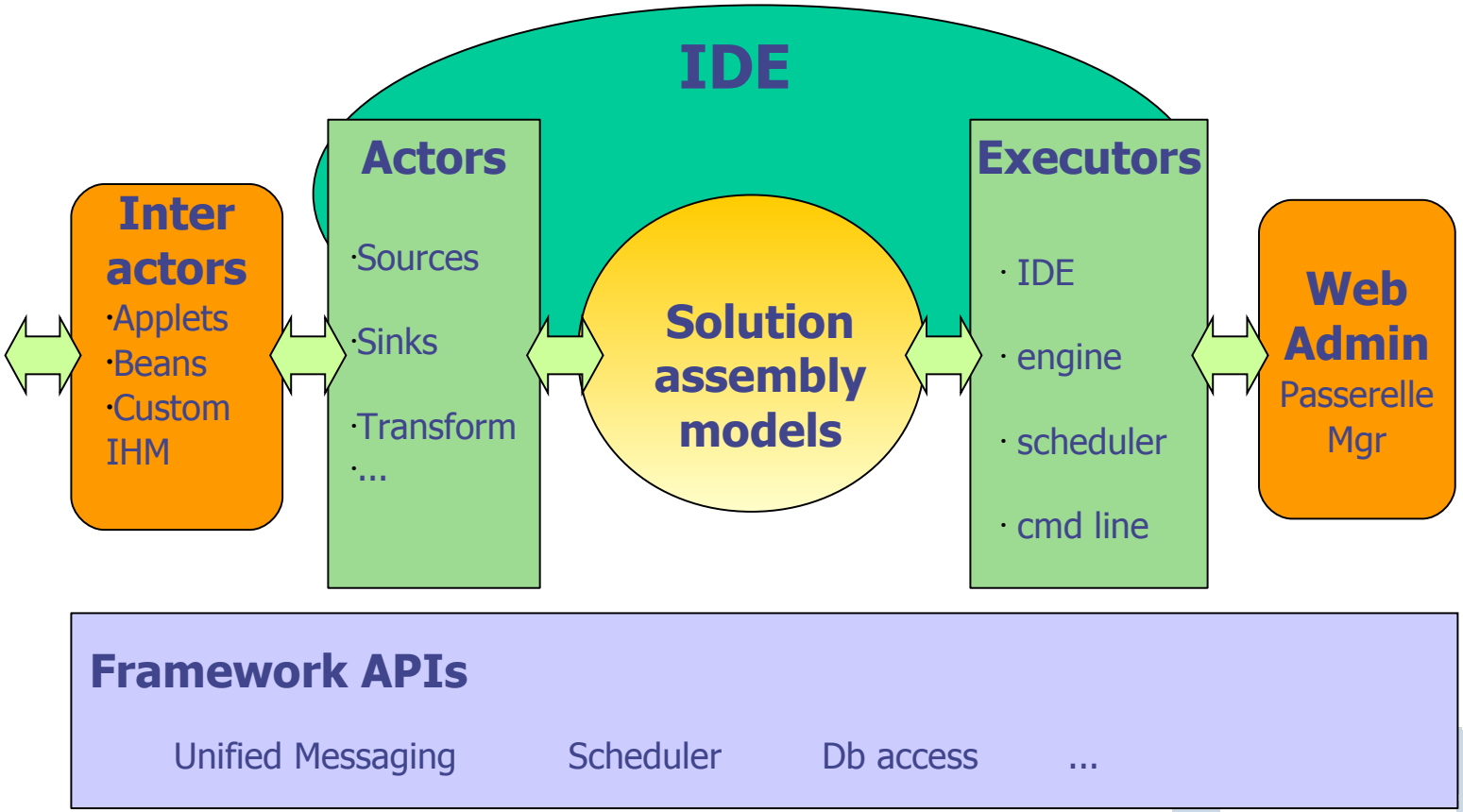
Benefits

- Take advantage of standard “good software engineering” architectural ideas (cfr above) and practices
  - Several test levels (unit test, system test, …)
  - Take advantage of test tools available in Java domain (Junit, Mock tests, continuous build & test, Ant/Maven build scripting, test reporting tools & build servers,...)
  - Version mgmt of workflows
- Support fort “auto-analysis”of executing workflows
  - Integration of rules engine component
- Workflow dynamic routing per request i.o. through “static”cfg changes

Actor-based solutions

- Based on actor-based paradigm of Ptolemy project
- Enriched with :
  - Usage & admin Tools : customized IDE, integration with Quartz scheduler, server run-time engine with DB persistence web admin UI, web services, ...
  - Dvp Tools : improved actor development API, standardized internal messaging model, automated data type conversions, improved error handling, run-time engine extensions, ...

Passerelle suite



Agenda

- Who's iSencia?
- Software offerings
- Some Passerelle use cases
- EDNA and Passerelle

Some Passerelle Use Cases

Light-weight communication engine

- Reuse of Passerelle UME
- Select components/libs matching communication needs
- Assemble communication layer once, embedded in app
- Passerelle completely hidden for end-users

Some Passerelle Use Cases

Batch job management

- Scheduling “expensive” actions out-of-business-hours (automated reporting, data replication & cleanup,...)
- Small assemblies, auto-terminating, limited number of executions per day
- Simple actors (DB access, running external scripts etc)
- Simple or no input data needed in the assemblies, only configuration data
- Using the Passerelle Manager
  - Secure environment to define and schedule jobs
  - Offers integrated job status and history views to follow/check execution status

Some Passerelle Use Cases

Process control and data acquisition

- Complex assemblies, auto-terminating, limited number of executions per day
- Adapted device control actors, combined with results analysis and reporting
- Simple or no input data needed in the assemblies, only configuration data
- Using the Passerelle Manager
  - Offers integrated job status and history views to follow/check execution status
  - Scheduler not often needed, plain “launch now”

Some Passerelle Use Cases

Personal automation tool

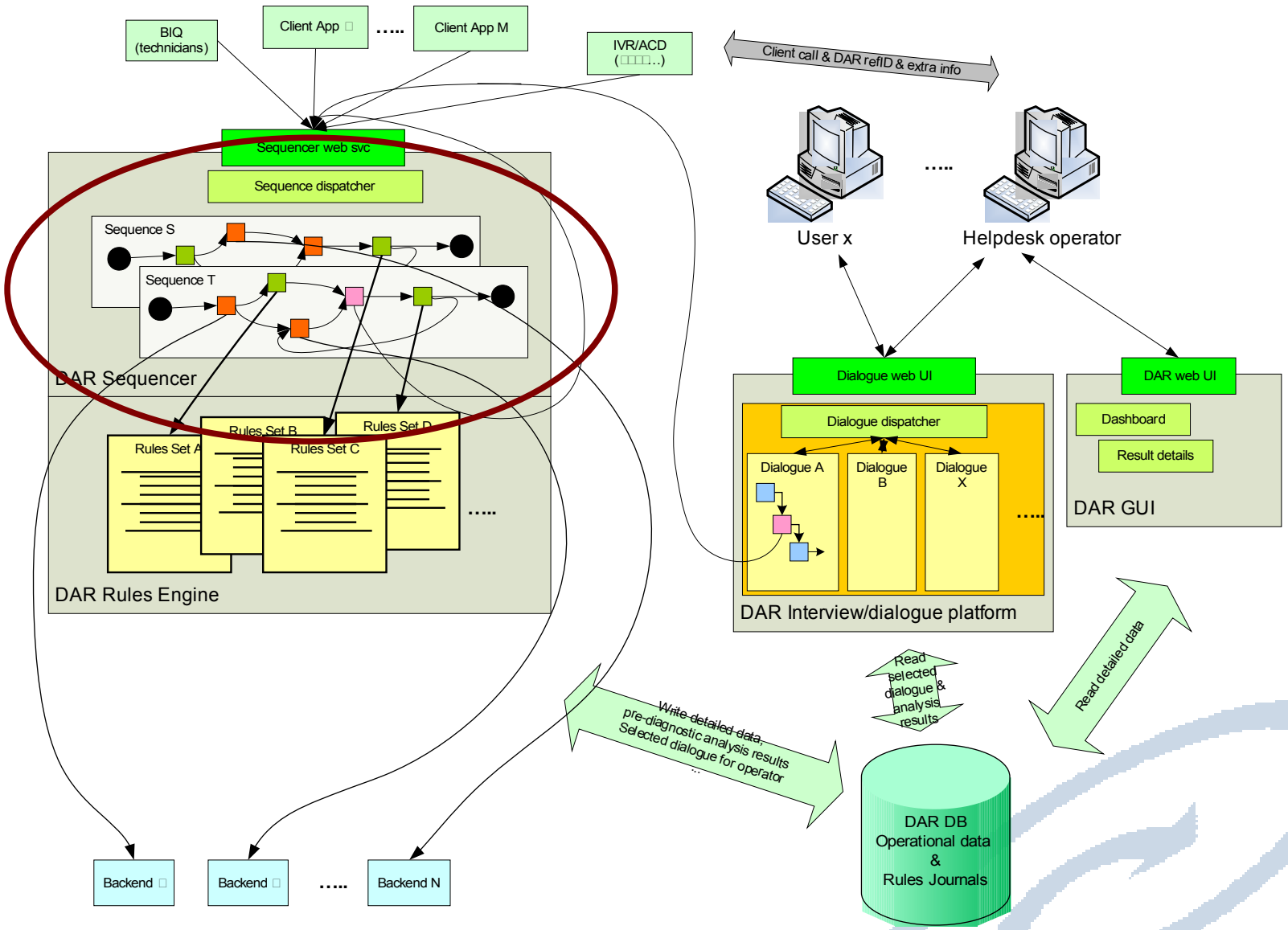
- Implement repetitive tasks via actors
  (personal backups, mailings, simple system monitoring etc)
- Replaces explosion of shell-scripts, macros, ...
- Auto-terminating assemblies
- Simple or no input data needed in the assemblies, only configuration data
- Plain Passerelle IDE is sufficient
  - Define personal “templates” in the User Library
  - Easy configuration of the personal tasks
  - Less reliability and tracing needed

Some Passerelle Use Cases

Diagnostic process engine

- Full-blown middle-tier processing engine
- Each diagnostic process represented as a graphical flow
- Limited number of complex assemblies, continuously running, non-auto-terminating
- Diagnostic is triggered by input messages
- Potentially high-throughput requirements
- Combines data collection tasks from approx 50 ≠ sources with rules-based data analysis tasks in integrated sequences/processes
- Part of a complete enterprise application environment

Diagnostic process engine

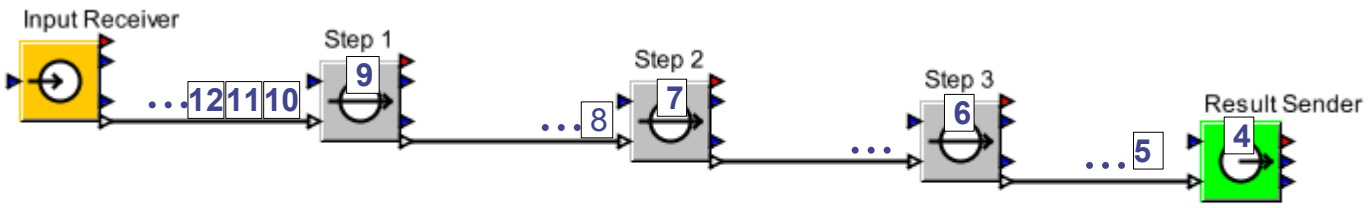


Diagnostic process engine

- Implemented on Passerelle Manager
  - Std job status & history not sufficient, need for persistence of extra execution trace details & data and analysis results via a generic data model
    (analysis to promote as std product feature is ongoing)
  - Needs integration in complete enterprise software infrastructure (clustered application servers, SOA, EJBs ...)

- Takes full advantage of Ptolemy's PN : multithreaded, internal buffering etc, leading to a high-throughput solution with a staged event driven execution model
  (cfr. automobile production/assembly line)

PN domain as “assembly line”



- Consecutive messages handled concurrently in different actors
- Each actor has own processing thread(s)
- Temporary congestion is handled by internal buffering on each input port

Leads to high-throughput with limited system resources, resilient for temporary peak loads

Agenda

- Who's iSencia?
- Software offerings
- Some Passerelle use cases
- EDNA and Passerelle

EDNA and Passerelle

EDNA :

*"The aim is to develop a **configurable application** able to **launch sequentially** the crystal **characterization** and the **data collection** strategy **steps** by executing **one or several parallel** external programs according to the user configuration"*

Passerelle :

- Graphical configuration and application assembly
- Flows with support for sequential & parallel steps
- Generic component model, support for automation, data collection and analysis etc

EDNA and Passerelle

EDNA :

Application components are “plugins” with well-defined lifecycle

Plugins invoke a limited number of external programs

Main development language Python

Passerelle :

Actor-based components with similar life-cycle

Already support for simple execution of external scripts

Basic Jython support

EDNA and Passerelle

Passerelle discussion topics :
(related to some EDNA issues discovered in docs/minutes)

- Java-based → platform-independent
- Integrated error reporting/handling mechanisms
- Actor mock-mode or external system simulators for testing
- Several levels of persistence
  (execution traces, result data model, …)
- Automated execution scheduling
- Administration via web application
- Support for GRID?
- Consulting result data via web UI
  (SherpaBeans app for ISPyB?
  → no need to create HTML, UI directly on DB schema)

EDNA and Passerelle

Possible approaches :

1. Invoke EDNA plugins from Passerelle
   - Develop Java wrapper API around Python Plugin API or use Jython actors
   - Develop limited set of actors (1?) on API
   - Exploit actor parameters for configuration (generate EDNA cfg file(s) for correct plugin functioning?)
   - Least-intrusive : EDNA-Python users can continue as now

EDNA and Passerelle

Possible approaches :

2. Directly invoke external programs from Passerelle
   - External programs executable as shell scripts?
     or reachable via inter-process communication?
   - Develop actor per program
   - Exploit actor parameters for configuration
     no need to generate EDNA cfg file(s)
   - Python-plugin development replaced by Java-actor development
   - Take full advantage of std Java / Ptolemy / Passerelle features
   - Large impact, both technological and psychological

EDNA and Passerelle

Possible approaches :

3. Mixed approach : option 1. for existing plugins, 2. for new ones
   - Valid as intermediate approach
   - Not as long-term solution → divided development and support

EDNA and Passerelle

Plugin ↔ Actor mapping :

setInputData : must be overridden! for casting XML into the XSDataInput type.

checkParameters : called befor config for assuring complete input data

config : called before preProcess, used for configuring the plugin

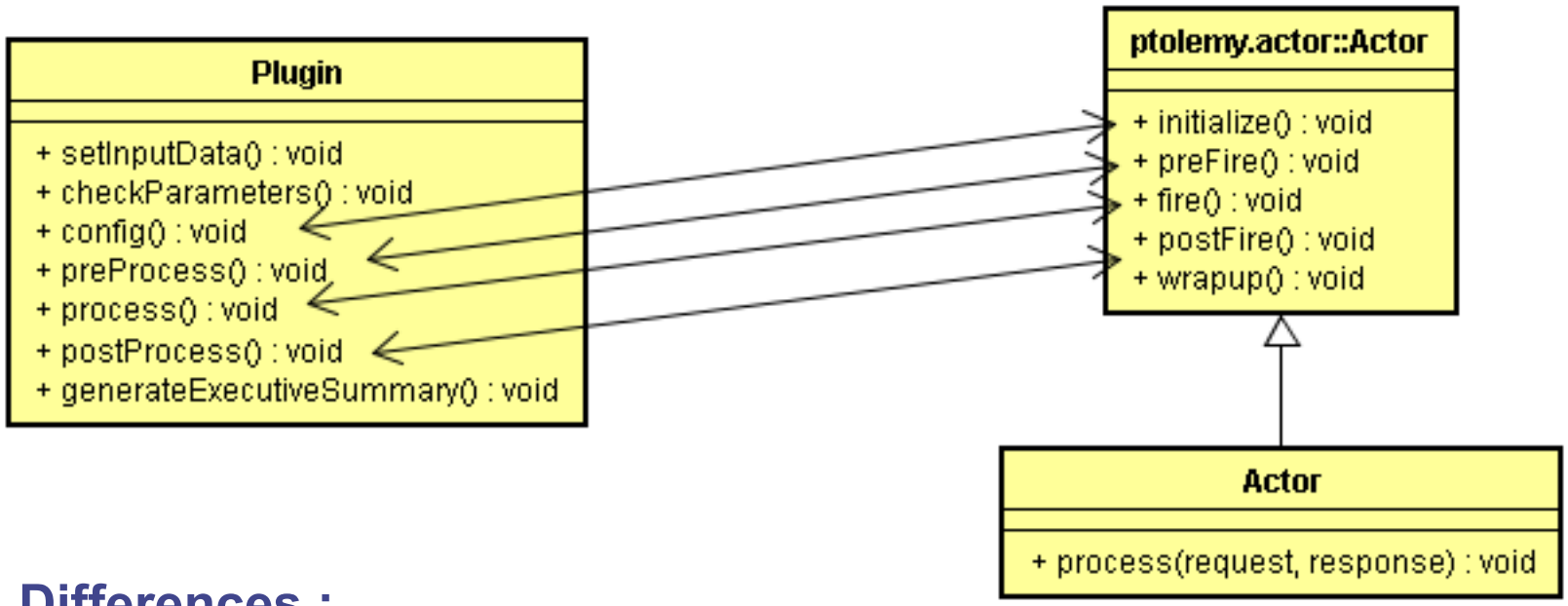
preProcess : called before process, used for setting up files etc. needed by the plugin

process : The main method. Does not need to be defined if derived from EDPluginExecProcess or EDPluginExecProcessScript

postProcess : called after the process, used for recuperating the data produced by e.g. external programs

generateExecutiveSummary : should write a human readable result summary after the execution of the plugin

EDNA and Passerelle



Differences :

- Actor executes fire-loop repetitively
- Several validation strategies can be plugged in at start of execution & during each actor fire-iteration
- generateExecutiveSummary could be in postFire(), wrapup() or even better in separate reporting tool accessing persisted actor results