EasySOA : Light approaches of entreprise integration

Analysis document

Version	Public Deliverable, V1.0
Date	January 31st, 2012
Authors	EasySOA partners: Open Wide, Inria
Contributors	EasySOA partners: Bull, Talend, Nuxeo

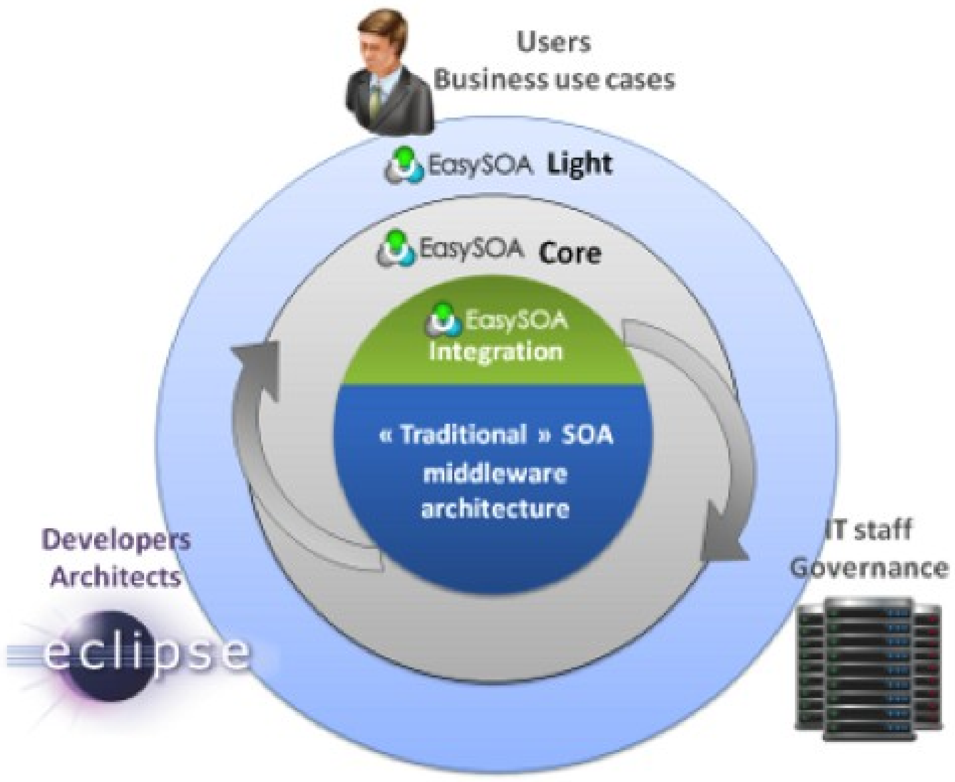
Summary

Index des illustrations

1. Context

This document is produced on behalf of the EasySOA consortium, which collaborates at building the EasySOA Open Source platform, whose goals and results are shown at http://www.easysoa.org.

As shown in the Figure below, main goal of EasySOA is to add a light and agile layer around the "traditional" SOA middleware architecture.



Dessin 1: How EasySOA works

In short, this document has to answer to the question of "what implementations of the SOA model, especially on the topic of collaboration / exchanges / "service" in the technical sense, should be supported (from model to tooling and even runtime plugins) in Light, but also in Integration ?"

This document first lists the criteria defining an entreprise integration approach as "light" and how it differs from "traditional" ones, which we'll both simply name Light and Integration.

Then it provides an overview of existing entreprise integration approaches (some being already Light), focusing on their main use case (what they're good for), integration paradigm (how they do it) and tooling public (who are their users), in order to make appear which ones are the most interesting to provide a Light alternative for in EasySOA, and what's required for that.

It also provides an overview of Light technology trends and how they're interesting for the topic at hand.

The following process has been shared by the authors of this document : analysis of the existing entreprise integration solutions, tools and standards, especially service-oriented ones, description of their specific business needs and use cases, design and prototyping of various Light alternatives.

2. Defining Light versus Integration

The difference between Light and Integration goes beyond technology (such as REST versus SOAP) and tools (online versus desktop-based), to being two different visions of entreprise information technology (IT), depending on where the cursor is placed between : agility, business-driven, short-term on one side ; and efficiency, scalability, long term on the other.

2.1. Light

“Light”:

- Uses agile development workflow and methodology, light technologies, bottom-up, fast prototyping and evaluation of business requirements
- But is harder to maintain and scale and requiring sandboxing & protecting consumed services by enforcing constraints on Quality of Service (QoS).

Light approaches can have one or more targets among :

- first and foremost, empowering "people close to business"
  - any user with some technical knowledge, typically project owner, but also Q / A & functional testers
  - web / mashups developers, including UI / mockup designers,
  - business data analysts i.e. spreadsheet / Extract Transform Load (ETL) / reporting users
  - business process analysts i.e. diagrams / business process designer users
- but as well, easing integrating "testing" for developers (web / online but also reusable in Integration / standalone IDE & SOA solution development)
- Light integration can also be a natural, valuable complement to light web application development with frameworks like Ruby on Rails, Django or Play!
- finally, Light can be used as a swissknife easing integration of the EasySOA platform itself in an enterprise's Information System (IS), and especiallly with it's “traditional” SOA solution, i.e. it's Integration side.

2.2. Integration

“Integration”:

- is "traditional" IT, comprising the full range of solutions on the market, able to answer any service level requirement,
- at the cost of IT human and financial resources (design, development, licenses, support, training).

We aim to accommodate, among those approaches, the main & most useful ones for (Easy)SOA.

We will support :

- at the root, support for classical SOAP / HTTP ESBs
- with additional support in Java development platforms (Maven, Jenkins / Sonar, Eclipse)

And branching out :

- a model-first approach of business & requirements
- allowing for business orchestrations (BPM)
- with flexible technical & business-oriented monitoring
- a first level of dedicated, exemplar integration with Talend ESB as integration platform
- and with Nuxeo as application platform

3. Overview of (service) integration approaches

3.1. Aspects of integration

Main aspects of integration (and its narrower service-oriented part) are :

- routing (service addressing), up to (service) composition including orchestration
- protocol connectors, including serialization / data binding and transport
- transformation, which when it has no business goals (like data mapping) is actually content manipulation i.e. what helps bridge the delta between protocol implementation details or versions (the "assembly of XML")

All along, we take here a user-centric approach, in that what matters is not what it allows to do, but who it empowers to do that. So everywhere the word "business" appear, there is also the question of identifying who it is the business of.

Technical aspects of Light integration can be addressed, hidden from the user, at the level of the Light platform. For instance, a SOAP service requiring WS-Security UserName Token extension to handle login requires configuration by the administrator at Light platform-level in order to map it with the platform's authentication facilities. This approach is however scalable if the same configuration can be reused for as many as possible other users, and even for as many as possible other services.

So, taking technical & non-business aspects out of the equation, we'll retain :

- Composition & its (visual) design tools, including the business parts of routing (like business proxy or chaining), business process, choreography and up to workflow.
- How it maps to and from data. More explicitly : business transformation & its tools, and its opposite subfields data mapping (field extraction) & template rendering. Note : in model transformation, data mapping and rendering are model-driven but exist nonetheless.
- End user UI & typical audience : who is comfortable using it, and will be using something that would look the same way ?
- What kind of code-level business logic is possible

While still keeping an eye on :

- How is handled integration with the entreprise information system and what it allows for, especially compared to the standpoint of SOA (Service Oriented Architecture)

3.2. Service Oriented Architecture (SOA) as integration pattern

Since all our study is done with the SOA architectural pattern in mind, here are a few reminders about it, so we won't need to say it again in all later parts of our analysis, especially when sizing up how each integration solution handles (SOA) integration with the entreprise information system.

3.2.1. Definition & value offer

In a Service-Oriented Architecture (SOA), integration between architectural components is done in a decoupled manner thanks to "services" provided by one and consumed by others who obey its contract. It allows consumers to develop with only the contract of consumed services, independently of their implementation ; and providers to improve their service implementation as long as it still respects the same contract.

3.2.2. Composition features

Besides service consumption, there are no composition patterns per se in SOA. As said the only SOA-level patterns are :

- service : call a service without knowing how he does its work. Allows to have, besides the "real" implementation, additional implementations just as compatible to the service contract. Such additional implementations could be mock implementation & simulation, proxy handling non-functional behaviour such as authentication or monitoring, intermediary layer ("facade") hiding technical "plumbing" such as handling version or business compatibility, instance with alternate implementation or configuration.
- contract between provider and consumer

However, by adding into the mix an all-purpose notion of architectural "component", like the SCA (Service Component Architecture) standard does, designing an SOA becomes

- specifying services provided and consumed by each architectural component,
- wiring them together,
- and wrapping all of them in turn within bigger components.

Such a component can be

- a business application,
- its atomic (code / modules) constituents,
- on the opposite a whole (information) system,

or only an incomplete part of it

- used as high-level model
- or published as composable part to components that are upwards (information-system level)
- or outside (consumer's or even customer's).

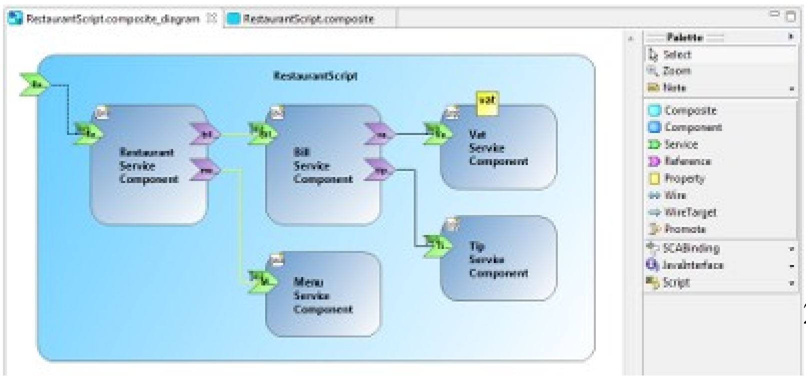


Illustration 1: Service Component Architecture (SCA) Diagram in Eclipse SCA Editor

Such black-box component allow for a lot integration patterns in it, including most routing ("Entreprise Integration Pattern") and orchestration patterns (see later), as well as bare code.

SOA is an integration pattern rather than an integration platform, and therefore not limited : any service-oriented integration platform like an ESB but also a web service-enabled BPM solution can fit in it. The SCA standard is standard and platform agnostic and can be further extended, however it is limited because it does not have much semantics (ex. Which is an application versus a module, which provides business A rather than B - which is rather good since whichever one can be put on it) and it does not handle well dynamic(ally composed) services (but generation can take care of that).

A well architectured and governed set of services will avoid duplicate services providing the same business but using incompatible service protocol or data format. The first can be solved by ESB as universal protocol connector, the second can be helped by MDM (Master Data Management, see ETL), but both belong to SOA governance.

Again, SOA is an integration pattern rather than an integration platform, so we won't study here code, information system integration or UI features. See rather in the most appropriate solutions later, like ESB (& REST).

3.3. Entreprise Service Bus (ESB)'s Entreprise Integration Patterns (EIP)

3.3.1. Definition & value offer

An Entreprise Service Bus (ESB) is a service-oriented integration middleware, providing an XML oriented toolbox (routing, protocol, transformation) helping develop and monitor service integrations.

Note that ESBs come from a long-lived history of entreprise integration : EAIs, messaging & object-oriented middlewares.

3.3.2. Composition features

Gregor Hohpe's Entreprise Integration Pattern (EIP) constitute ESBs' composition palette.

We won't list here EIP patterns mentioned in their own part in this document, such as business process orchestration, workflow engine & service, rule-driven engine & service (actually, an ESB could integrate any service engine), since ESB adds no value to them and rather often subtracts some, such as taking them out of their own tools and users. So among business-aimed ones, providing various level of composition abilities, let's cite :

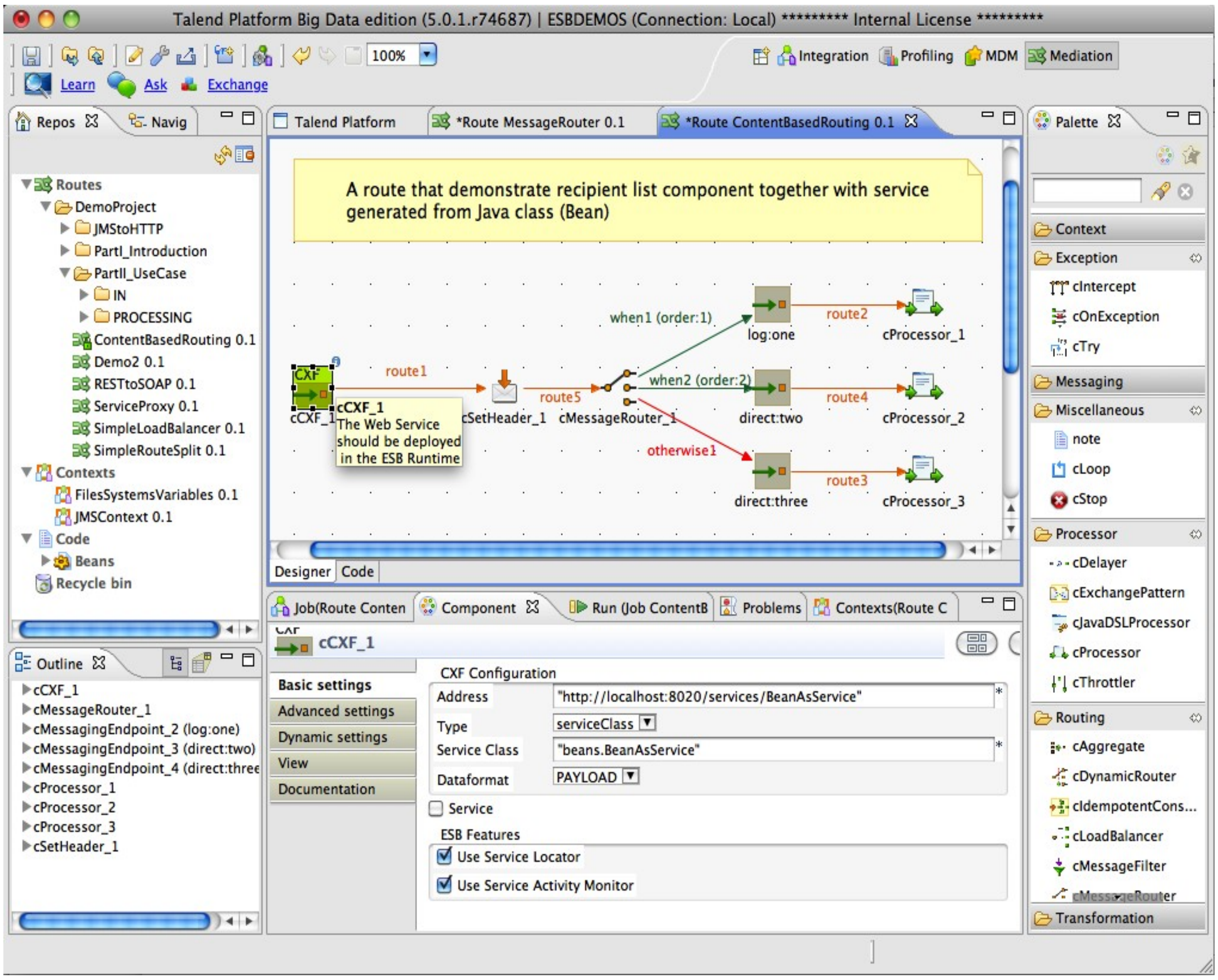
- Service proxy : provides benefits of the basic service pattern in ESB.
    - Allows to change the implementation from a simulation (mock) to the real one, to hide

routing details such as between the various versions of implementations, and to inject required plumbing between two slightly incompatible implementations, one-case patches ("detour"), non-functional behaviour such as authentication or monitoring ("wiretap"), additional business logic, hooks for any other custom behaviour such as by emitting events, to provide dynamically composed ("virtual", negotiated, instantiated) services, to provide the service to other consumers which otherwise wouldn't be able to access it (being restricted to other protocols or networks).

  - Strictly spoken, definitions of the service exposed by the proxy and of its proxied service have to be the same. However, extrapolating from said abilities (providing the same service in another flavour), and considering that boundaries between features of the service versus of the platform vary, "proxy" can by extension also refer to any component allowing to inject additional behaviour, even if both definitions are not the same. Example 1 : implementing a workflow service with platform-integrated authentication on top of an existing one requiring to pass user id or role explicitly as parameters. Example 2 : reducing complexity by hiding several services behind a single "facade".

- Routing & orchestration : which service(s) to actually call or mashup. Target services can be set in configuration (like in SCA standard), chosen among all (ESB registry-known) services conditionally according to content, or extracted from content (content-based routing). Its simplest patterns are, in order, chaining service calls ("routing slip"), dispatching them (parallel), aggregating them, with more powerful variants allowing to do so conditionally and based on request content. A complex version of configuration-based routing is process-based service orchestration, which is achieved by an embedded or external orchestration engine. A complex version of conditional routing is rule-based routing, which is achieved by an embedded or external rule engine.
- event : subscribe to an "event" and get notified through correlation. An event definition is basically correlation configuration that says whether a given message is eligible to it or not. A simple correlation pattern is topic-based correlation (publish / subscribe). Complex correlation can be achieved using languages, either targeting message content (XSL / Xpath) or model (script & code), or targeting more dedicated paradigms such as decision rules and its alternative more suited to ESBs : complex stream event processing (CEP / ESP). Producing events is therefore as simple as calling the engine that dispatches event messages according to these correlations, and can be easily hooked in existing integration through proxies, or even emitted ex nihilo using polling.

See more about EIP at http://www.eaipatterns.com/ .



cCXF_1

CXF Configuration

Address	"http://localhost:8020/services/BeanAsService"
Type	serviceClass
Service Class	"beans.BeanAsService"
Dataformat	PAYLOAD

☐ Service

ESB Features

☑ Use Service Locator

☑ Use Service Activity Monitor

Dessin 2: Entreprise Integration Pattern composition in Talend Open Studio's Camel Editor

3.3.3. Code and data

ESB business routing and service composition is possible thanks to its service registry.

EIPs are usually implemented in ESBs as dedicated service engines driven by specific configuration, usually XML files, and best designed in vendor-provided tools, which are often visual ones but nonetheless targeted to developers.

ESB native message & service format is usually XML & WSDL. Since they target developers, data mapping & rendering are often restricted to XPath & XSL.

Scripted or coded logic is allowed and can work on message content (XML) or its model (thanks to data binding) equivalent, and therefore quite clean, but ESBs are far too deep to put much business logic in there.

3.3.4. In the Information System

ESBs' power to address non-functional requirements rely on the flexibility of their service stack when they are tied to protocol standards (such as WS-*), and otherwise to the power of their own peripheral components and solutions (such as on distributed routing, monitoring).

Integration to information system is done using service-oriented connectors for standards or specific solutions.

3.3.5. UI & audience

ESBs provide monitoring UI, albeit heavily technical administrator-focused.

3.3.6. Solutions available on the market

All big IT editors have their own :

- IBM (in the WebSphere offer),
- Oracle

Players from the integration field usually have one :

- Sonic,
- Tibco,
- Progress,
- Informatica (comes from the data integration field).

Main open source commercially supported ones are :

- Mule (proxy-centric),
- Talend (good integration with ETL & MDM),
- Red Hat (JBoss, open source),
- WSO2 (modular architecture),
- Petals (natively distributed).

See more at http://en.wikipedia.org/wiki/Comparison_of_business_integration_software .

3.4. *Business Process integration*

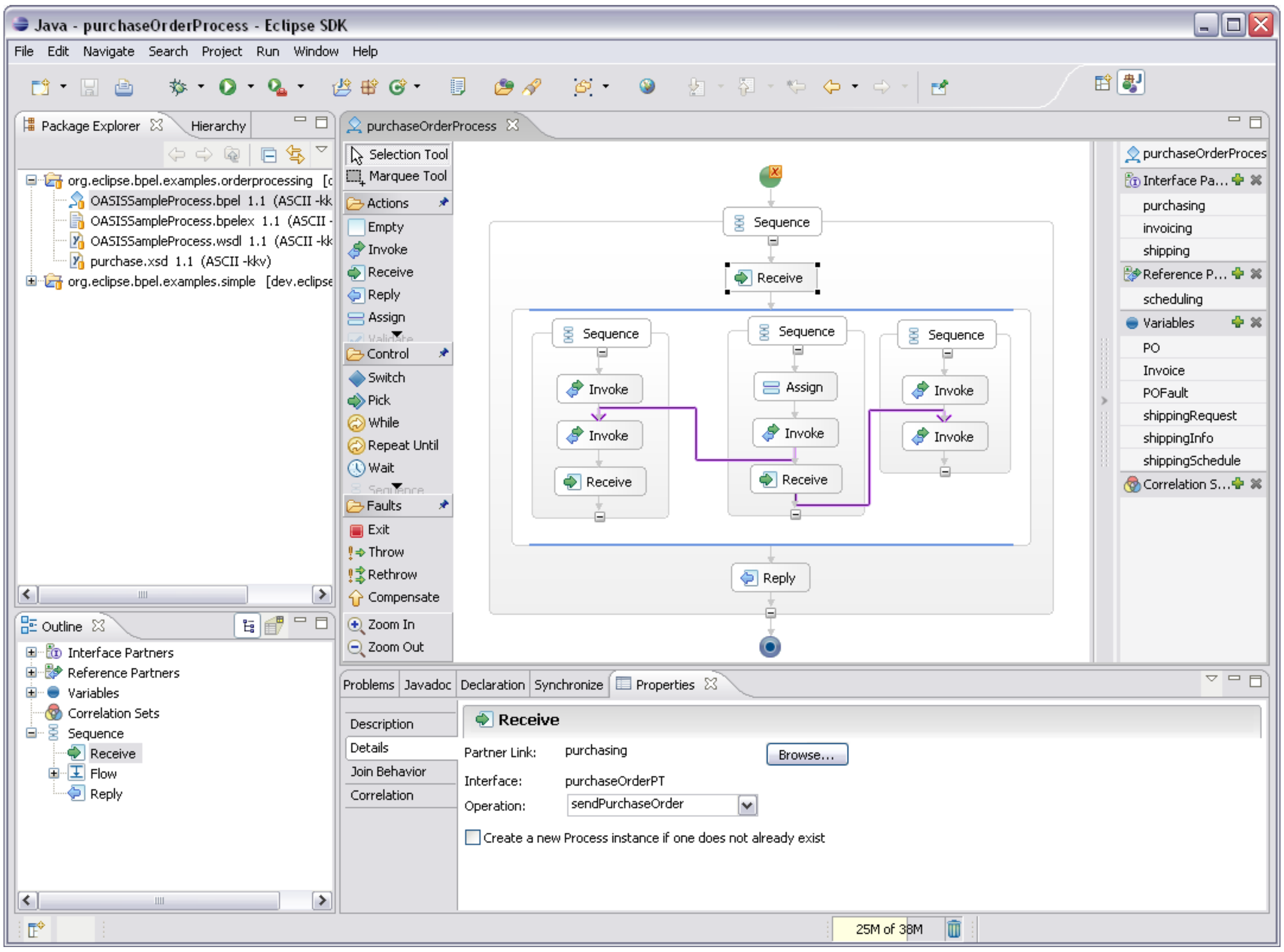
3.4.1. Definition & value offer

Business Process Management (BPM) aims at modeling the entreprise's business processes to capitalize on their knowledge, manage their execution and optimize them (through simulation, monitoring and a posteriori analysis). To ease it up, BPM solutions provide, to users with knowledge about business processes up to dedicated business analysts, visual design, integration with the Information System, interaction (such as through generated forms), a posteriori analysis (such as reporting).

3.4.2. Composition features

BPM's composition palette comprises the various avatars of business processes. Let's cite :

- orchestration of components / services, BPEL (Business Process Execution Language) being the most prominent model standard. Actually consists in block-based languages dedicated to flow control & mashup of services calls. Simple orchestration patterns are first chaining service calls, then aggregation and conditional behaviour.



Partner Link: purchasing

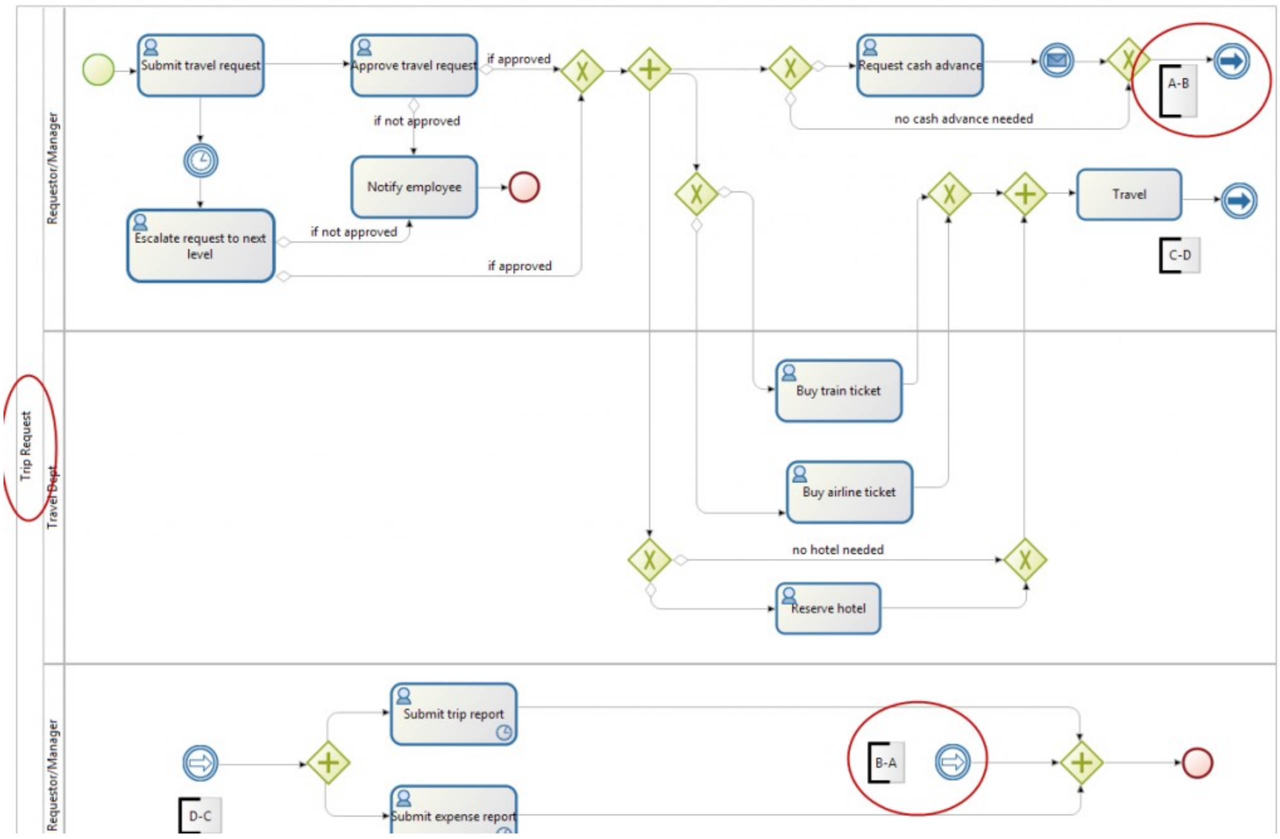
Interface: purchaseOrderPT

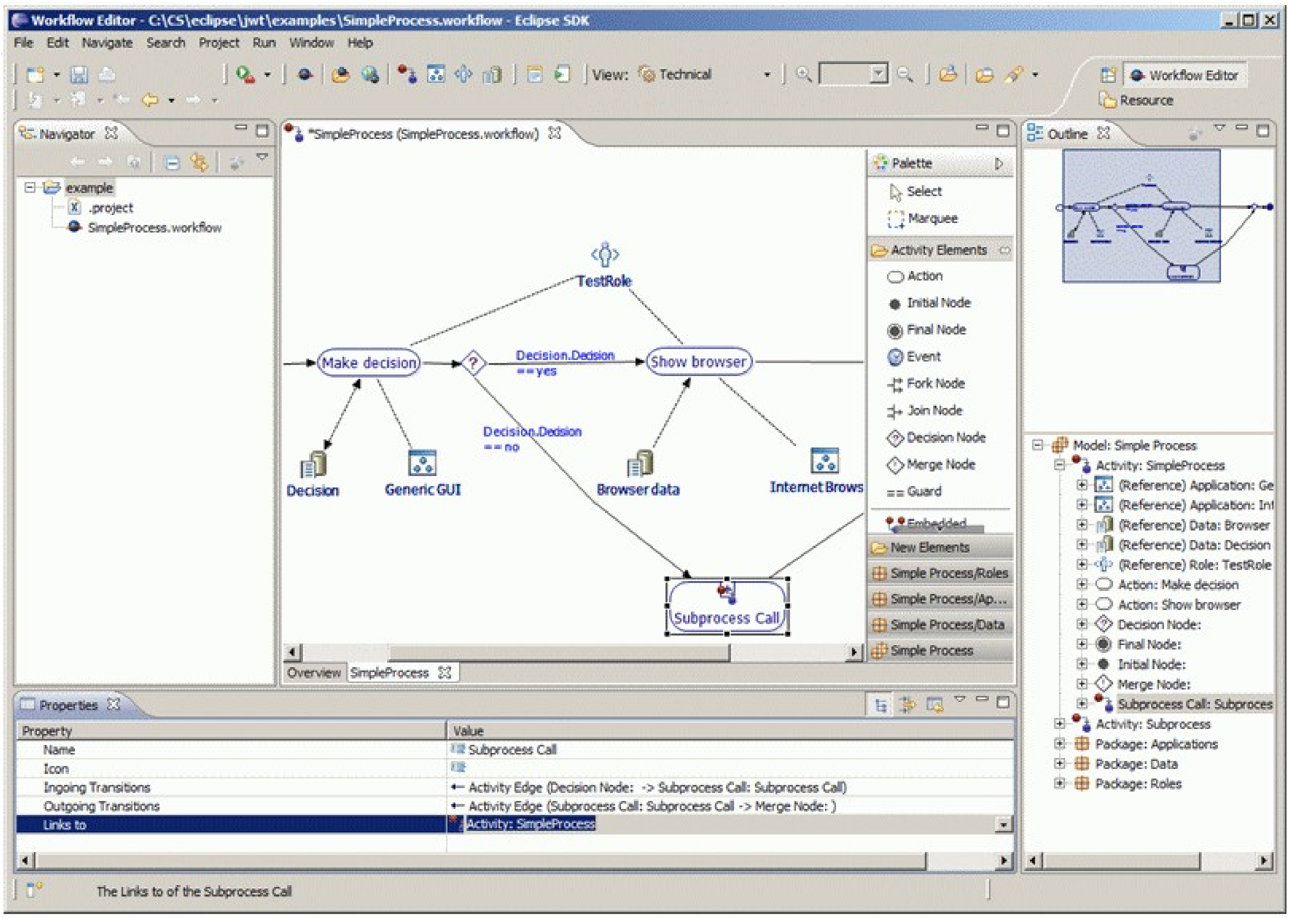
Operation: sendPurchaseOrder

☐ Create a new Process instance if one does not already exist

Illustration 2: Business Process Execution Language (BPEL) in Eclipse BPEL Editor

- workflow, or role-oriented orchestration, a.k.a. the human user's point of view, BPMN2 (Business Process Modeling Notation) and XPDL (XML Process Definition Language) being widespread model standards in the BPM field, while beyond UML (Unified Modeling Notation) Activity Diagram benefits of UML's widespread recognition in business modeling. Actually consists in declarative definition of flow control & task nodes and conditional transitions between them. Simple workflow patterns are first sequential and parallel transitions, then conditional ones.





Property	Value
Name	Subprocess Call
Icon	
Ingoing Transitions	Activity Edge (Decision Node: -> Subprocess Call: Subprocess Call)
Outgoing Transitions	Activity Edge (Subprocess Call: Subprocess Call -> Merge Node:)
Links to	Activity: SimpleProcess

Illustration 4: Unified Modeling Language (UML) Activity Diagram symbols used in Eclipse Java Workflow Tooling's Editor

- choreography, which is actually several services (or processes) working together along a shared understanding of the process,
- rule-driven task management (or task basket-oriented management), allowing for more power (up to costless live process migration) than all previously cited model-driven task management alternatives, at the cost of being more brittle.

3.4.3. Code and data

BPM engines focus on controlling the flow of work. BPM solutions make this approach scale up by helping share the processes, thanks to the process repository.

In all cases, the oriented graph is by far the most widespread and useful graphical representation and design tool.

Coded business logic is allowed as an inevitable "glue" but restricted to context-focused scripting.

Mapping of business data is just as inevitable and adequately visually tooled but neither central.

3.4.4. In the Information System

Integration with Information System is often provided out-of-the box by BPM solution using a Connector-based approach. Indeed, BPM focuses on sharing the processes, but less so the integrations. This is no scalable integration solution, and BPM is therefore complementary to SOA, and BPM solutions can let business logic, data mapping and IS integration to entreprise integration middlewares such as ESBs – actually, if they don't work with one, they often come with embedded parts of it to do the job (such as handling SOAP service calls).

3.4.5. UI & audience

Finally, day-to-day interactions with business processes are made palatable through generated & customizable form UI, and execution can usually be monitored in a UI reusing the process' oriented graph, or at worse its task list. Overall user audience is comfortable with forms and oriented graphs ("Visio people").

3.4.6. Solutions available on the market

All big IT editors have their own BPM solution :

- IBM (formerly Lombardi),
- Oracle,
- SAP

Players from the integration field sometimes have one :

- Tibco,
- Progress,
- Informatica (comes from the data integration field).

Note that some offers focus address rather management than execution of business processes :

- Aris (focuses on business architecture)
- Mega (focuses on entreprise architecture)

See more at
http://en.wikipedia.org/wiki/Comparison_of_Business_Process_Modeling_Notation_tools

Main open source commercially supported ones are :

- BonitaSoft (proxy-centric),
- Intalio
- ProcessMaker
- Activiti (started by the Alfresco ECM vendor with jBoss jBPM's author)

See more at http://www.softwareforenterprise.us/2009/03/13/list-of-top-open-source-bpm-workflow-solution/

3.5. Data integration's ETL

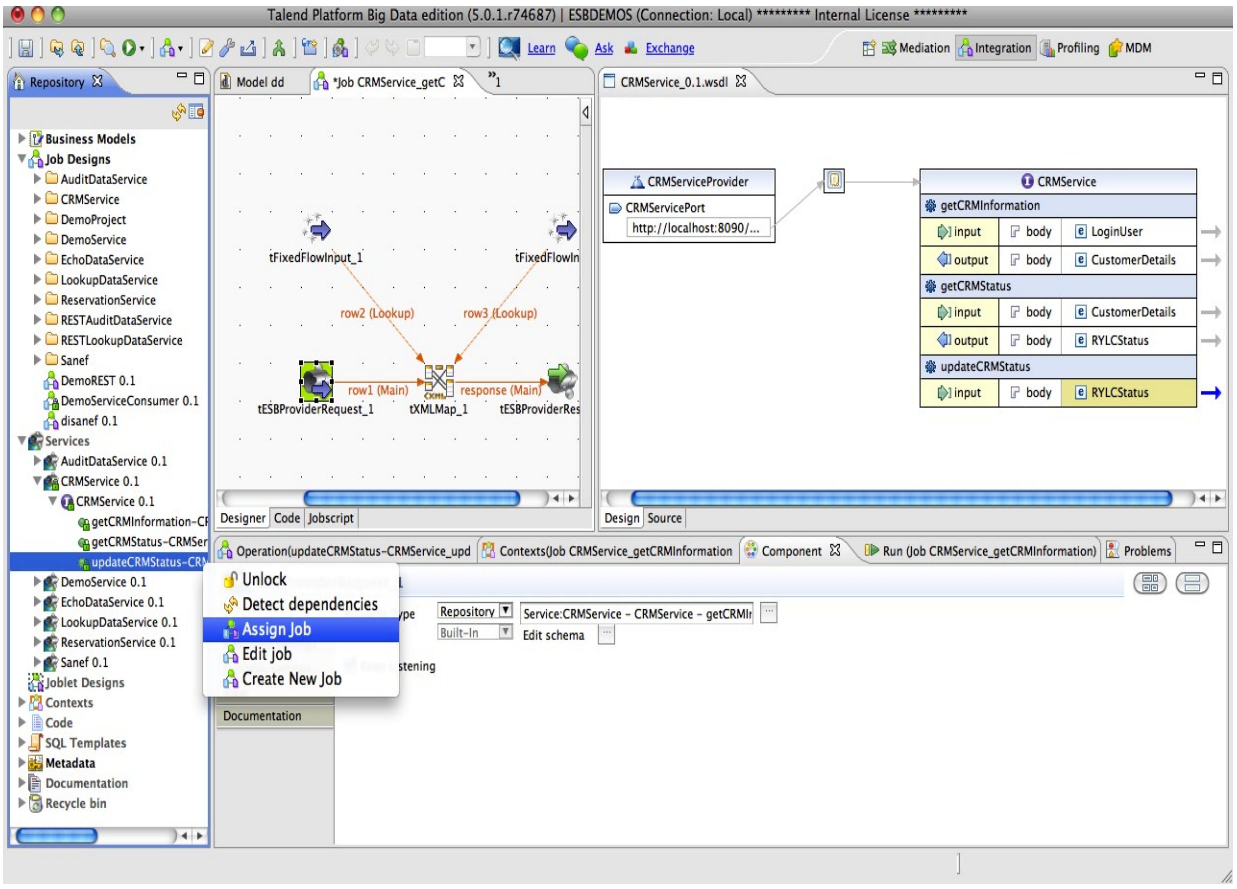
3.5.1. Definition & value offer

Data integration can be required only once (migration to another data storage solution), by batch (notably for business data analysis, which includes querying, reporting, dimensional analysis), or realtime (access by business application, however usually through application frameworks like Object Relational Mapping rather than data integration, or business services).

ETL (Extract, Transform, Load) tools are the common swissknife for users with business knowledge about data when providing a solution to the above use cases. Master Data Management is the pivotal reference for data formats that makes all of them possible together and scalable.

3.5.2. Composition features

In ETL, composition is done by controlling the bare data flow and the consecutive steps it takes. Some steps are similar to ESB's EIP such as XOR routing, others are more specific to data handling such as doing joins on several datasources.



Dessin 3: Extract Transform Load (ETL) tool Talend Open Studio

3.5.3. Code and data

In ETL, coded logic is allowed, but obviously limited and not well suited to business logic.

In ETL, mapping of business data is central, and therefore very well visually tooled, allowing for intuitive field selection, extraction and aggregation and supporting all data formats.

3.5.4. In the Information System

Integration with the information system is done using data connectors for standards or solution specific. Thanks to MDM, ETL can scale up by sharing data formats, but is less appropriate to share integrations. That's why SOA is complementary to data integration, which can in turn perfectly handle SOA's data format impedance issues.

3.5.5. UI & audience

Monitoring UI are rather technical administrator-focused. Overall user audience is comfortable with spreadsheet data (“Excel people”).

3.5.6. Solutions available on the market

Main commercial ETLs are :

- IBM DataStage
- Oracle Data Integrator
- SAP Business Objects' Data Integrator
- Informatica PowerCenter

Main Open Source ones are :

- Talend
- Pentaho's Kettle

See more at http://en.wikipedia.org/wiki/Extract,_transform,_load

3.6. Business Intelligence

3.6.1. Definition & value offer

Business intelligence is about getting insight on business data (querying, reporting, realtime dashboards of Key Performance Indicators and further data analysis and visualisation), its evolution (dimensional analysis), up to business process execution (Business Audit and Monitoring).

3.6.2. Composition features

BI is about seeing the data you want, how you want.

The first is the core of business data querying, at the root of reporting, and used in indicators. It uses patterns like query / join / aggregate, pipes & filters, sort and allow to design them visually using a unified business, graph-oriented model of data relations. Languages are used for query (SQL or business-level equivalent).

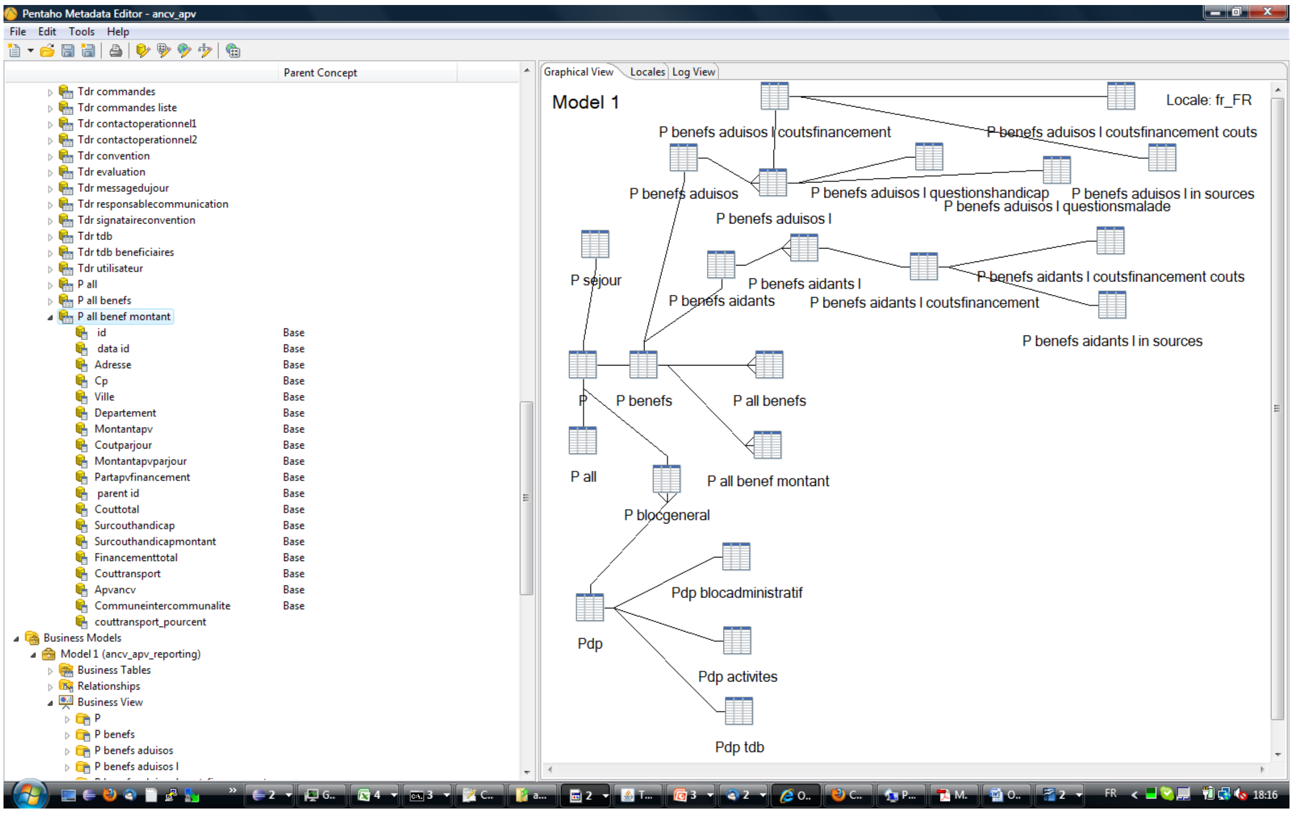
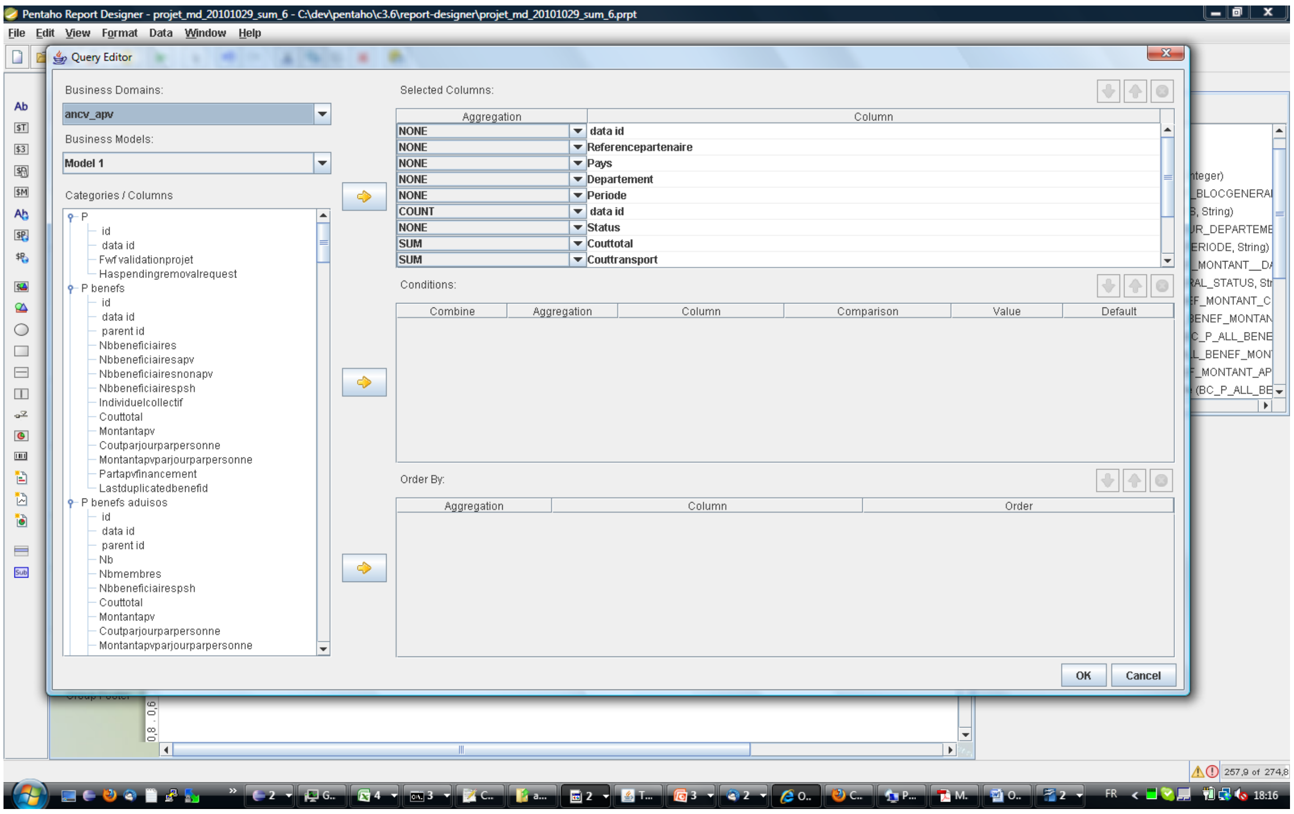


Illustration 5: Business model of data relations in Pentaho's Metadata Editor


Pentaho Report Designer - projet_md_20101029_sum_6 - C:\dev\pentaho\c3.6\report-designer\projet_md_20101029_sum_6.prpt
File Edit View Format Data Window Help
Query Editor
Conditions:
Combine Aggregation Column Comparison Value Default
Order By:
Aggregation Column Order
OK
Cancel

Business Domains: ancv_apv

Business Models: Model 1

Categories / Columns

- P
  - id
  - data id
  - Fwf validationprojet
  - Haspendingremovalrequest
- P benefs
  - id
  - data id
  - parent id
  - Nbbeneficiaires
  - Nbbeneficiairesapv
  - Nbbeneficiairesnonapv
  - Nbbeneficiairespsh
  - Individuelcollectif
  - Couttotal
  - Montantapv
  - Coutparjourparpersonne
  - Montantapvparjourparpersonne
  - Partapvfinancement
  - Lastduplicatedbenefid
- P benefs aduisos
  - id
  - data id
  - parent id
  - Nb
  - Nbmembres
  - Nbbeneficiairespsh
  - Couttotal
  - Montantapv
  - Coutparjourparpersonne
  - Montantapvparjourparpersonne

Selected Columns:

Aggregation	Column
NONE	data id
NONE	Referencepartenaire
NONE	Pays
NONE	Departement
NONE	Periode
COUNT	data id
NONE	Status
SUM	Couttotal
SUM	Couttransport

Illustration 6: Filter, Group & Sort in Pentaho's Query Editor

The second is the core of business data visualization, and a main part of report and dashboard layout. It relies respectively on visual mashups (see later entreprise mashup solutions), widget-oriented web portals (facebook-like) and visual layout composition (usually using a desktop editor tool).

3.6.3. Code and data

There is no logic in BI (save possibly UI logic).

Data mapping is usually handled automatically thanks to said unified business-level visual model of relations. Designing it requires going down to said query languages.

3.6.4. In the Information System

Integration with the information system is done using data source connectors for standards (JDBC / ODBC) or solution specific. Note that BI often relies on its own data store, which is designed to provide a specific view (and sometimes relies on specific technology such as OLAP), and which is fed by batch using ETL. Realtime data analysis relies more on actual data services (again, see mashups later).

3.6.5. UI & audience

UIs for business users are paramount, see above. Overall user audience is comfortable with data analysis (filters and facetted, contextual, drill-down search), reporting and lately web portals.

3.6.6. Solutions available on the market

Main commercial BI solutions are :

- SAP Business Objects
- Microsoft
- Cognos

Main Open Source ones are :

- Pentaho
- SpagoBI
- A lot of independent tools are also available, especially for reporting (JasperReport, Eclipse BIRT)

3.7. Entreprise mashups

3.7.1. Definition & value offer

Entreprise mashups are a web-age offshoot of Business Intelligence's realtime dashboard & data visualisation. Just like BI's two roots of querying and visualisation, in entreprise mashups, mashupped web contents (services) are exposed to users through mashupped web UIs (visual mashup / apps / widgets). Like BPM, mashups want to be SOA's killer application.

3.7.2. Composition features

Web service mashups allow composition of web services, that is mostly XML or JSON (JavaScript Object Notation) content on HTTP, by taking advantage of web content standards (XSL, Xpath, Xquery, Javascript). The recent EMML (Entreprise Mashup Markup Language) standard by the Open Mashup Alliance classifies mashup composition patterns according to the following items :

- filter, group, sort
- split / parallel, join / merge, condition
- last but not least : scripting, web scraping & clipping

See more about EMML at the Open Mashup Alliance (gathers HP, Intel, Adobe, CapGemini, JackBe, Convertigo) at http://www.openmashup.org/ .

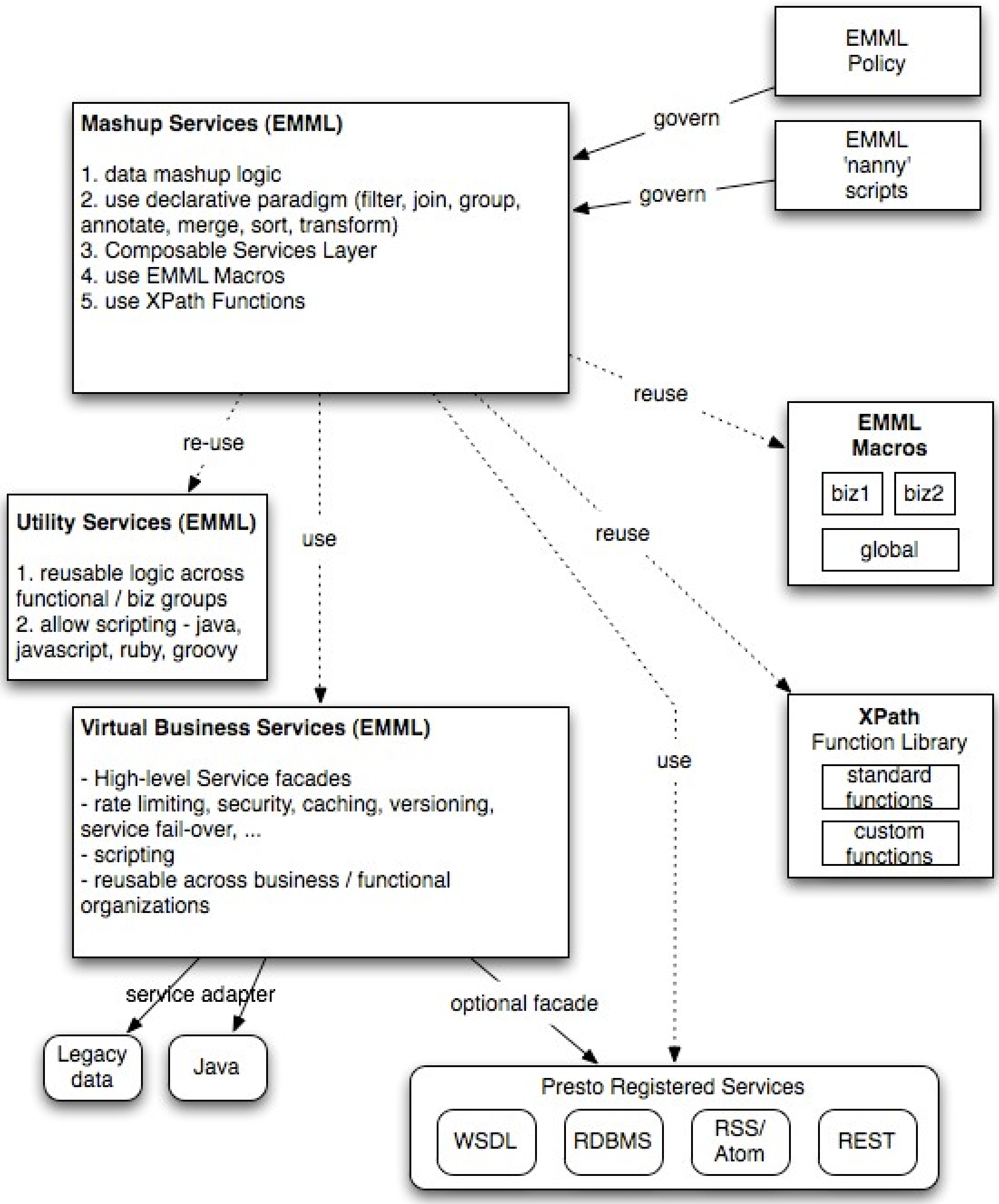

EMML
Policy
EMML
'nanny'
scripts
govern
govern
Mashup Services (EMML)
1. data mashup logic
2. use declarative paradigm (filter, join, group, annotate, merge, sort, transform)
3. Composable Services Layer
4. use EMML Macros
5. use XPath Functions
reuse
EMML
Macros
biz1
biz2
global
re-use
use
reuse
Utility Services (EMML)
1. reusable logic across functional / biz groups
2. allow scripting - java, javascript, ruby, groovy
XPath
Function Library
standard functions
custom functions
use
Virtual Business Services (EMML)
- High-level Service facades
- rate limiting, security, caching, versioning, service fail-over, ...
- scripting
- reusable across business / functional organizations
service adapter
optional facade
Legacy data
Java
Presto Registered Services
WSDL
RDBMS
RSS/ Atom
REST

Illustration 7: How is Entreprise Markup Language (EMML) implemented in JackBe according to Chief Architect Rajmohan Krishnamurthy, see http://raj-rajmohan.blogspot.com/2009/04/emml-best-practices.html

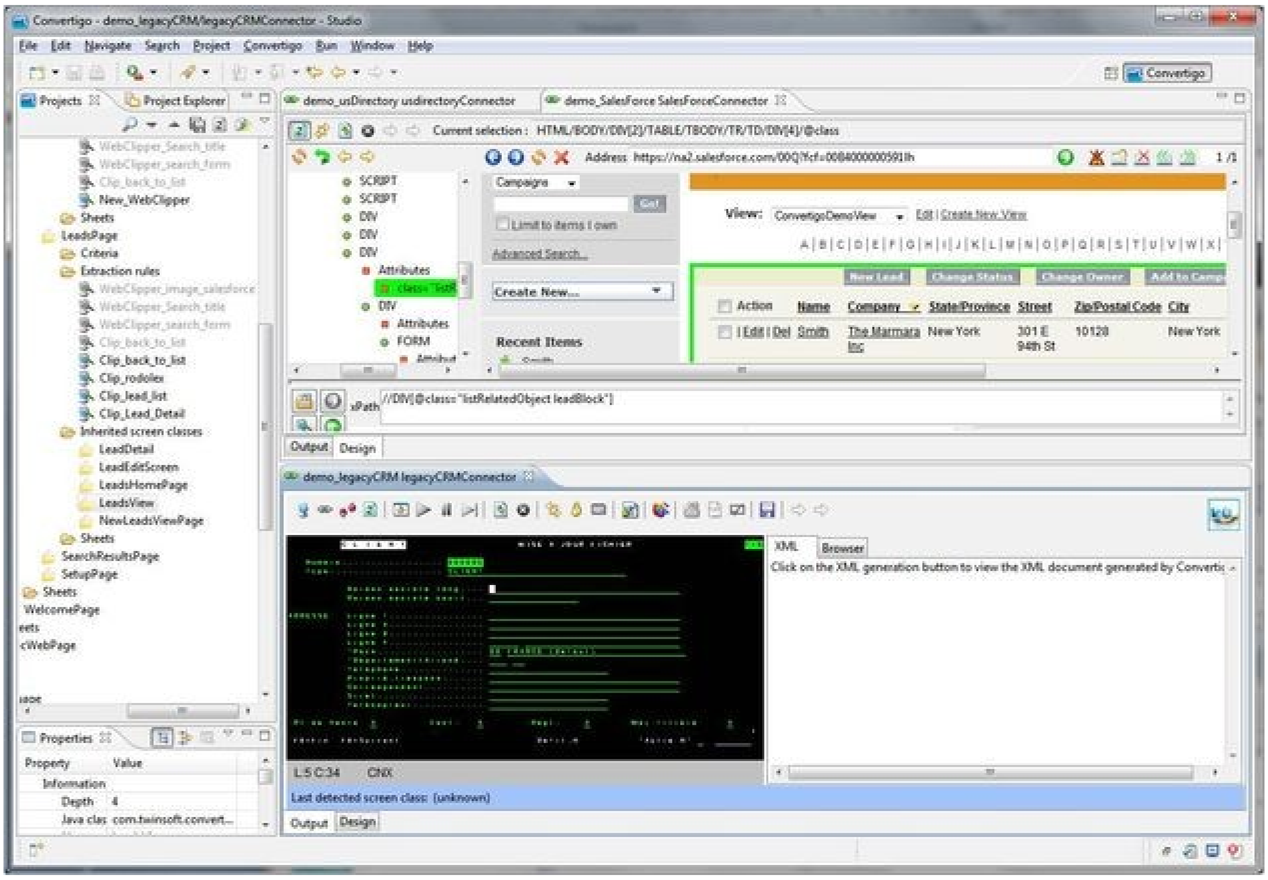

Convertigo - demo_legacyCRM/legacyCRMConnector - Studio
File Edit Navigate Search Project Convertigo Run Window Help
Projects
Project Explorer
demo_usDirectory usdirectoryConnector
demo_SalesForce SalesForceConnector
Current selection : HTML/BODY/DIV[2]/TABLE/TBODY/TR/TD/DIV[4]/@class
xPath //DIV[@class="listRelatedObject leadBlock"]
demo_legacyCRM legacyCRMConnector
Output
Design
XML
Browser
Click on the XML generation button to view the XML document generated by Convertigo
Last detected screen class: (unknown)
Properties
Property
Value

Illustration 8: Convertigo Mashup Editor

3.7.3. Code and data

Web scripting (Javascript) is commonly used, including to manipulate data.

Data mapping is helped for visually but web services are harder to map than e.g. ETL's data flow. JSON (JavaScript Object Notation) is a less typed but more readable and far more efficient (because of script interpreter software) alternative to XML as a data textual format. Therefore scripting, given how powerful & widespread it is among web cognoscenti, is a common solution.

3.7.4. In the Information System

Integration with the information system is done using data service connectors and mainly target web standards : WSDL, REST (see next chapter), the RSS (Really Simple Syndication) & Atom syndication protocols, in addition to databases. Mashup servers focus on sharing web service or visual mashups, and connecting them to information system data in an uniform manner. Thus they can benefit from MDM's sharing of data formats, and from SOA's shared knowledge about services.

3.7.5. UI & audience

As in BI, UIs for business users are paramount, see above. Overall user audience is comfortable with web applications and technologies (“web people” - from HTML to Javascript and further).

3.7.6. Solutions available on the market

Main commercial BI solutions are :

- Serena Business Mashups
- JackBe's Presto. Allows to reuse “mashables” to compose mashups and provide them in “apps”, see http://www.jackbe.com/enterprise-mashup/content/about-presto
- IBM Websphere Smash. Targets fast creation of situational applications using preexisting assets, see http://publib.boulder.ibm.com/infocenter/wsmashin/v1r0/index.jsp?topic=/com.ibm.websphere.sMash.doc/overview/zero.overview.doc/docs/en/Overview.html
- Presto
- Yahoo Pipes (online, pure player)

Main Open Source ones are :

- Convertigo (just became Open Source, see http://www.journaldunet.com/developpeur/client-web/convertigo-enterprise-mashup-server-en-open-source-1211.shtml)
- WSO2 Mashup Server (open source, tested)

See more about mashups at http://en.wikipedia.org/wiki/Mashup_%28web_application_hybrid%29 .

3.8. Entreprise Content Management

3.8.1. Definition & value offer

Allows for a document-oriented integration style where all participants get from or put in their data as documents (files tagged with business metadata) in the same ECM server thanks to its easy document interoperability, and then take advantage of ECM's document rights and workflows to let users manage them, (web) scripting and templating to mashup and render them.

3.8.2. Composition features

There's no particular composition pattern.

3.8.3. Code and data

Custom business logic can be scripted on document lifecycle event or user action.

There is no help to data mapping per se, but there are specific data management abilities thanks to being stored according to a customizable meta model, which make it particularly suited to store high-level, business, classifying or semantic information.

3.8.4. In the Information System

Integration to the information system in the document-oriented style is fostered as much as possible, using document interoperability standards such as CMIS, integration standards such as web services, or other bulk import & export features (that can be eased by ETL).

3.8.5. UI & audience

UI is either generically document-oriented and powerful but complex, or simple custom static or dynamic web site publishing ECM information specifically towards targeted consumers. Overall user audience is comfortable with documents (“Word people”) and web sites.

3.8.6. Solutions available on the market

Main commercial ECMs are

- EMC Documentum,
- IBM FileNet,
- Microsoft Sharepoint.

Main Open Source ones are

- Alfresco,
- Nuxeo.

3.9. Others integration patterns

3.9.1. Big Data

A web-age offshoot of grid computing for scaled up BI data analysis (see Business Intelligence).

Introduces new platforms and patterns, of which the best known is map / reduce (similar to custom scripted indexes).

3.9.2. Business rules engine

Decision engine based on “condition to be evaluated” – “action to be taken as a consequence” (see ESB).

Typical audience is comfortable with spreadsheet decision tables (“Excel people”) and sometimes scripting.

3.9.3. Complex / Stream Event Processing

Realtime business rules-driven processing (see ESB).

3.9.4. Alignment on Business Requirements & functional testing

Business requirements & functional testing are the role of the project owner, rather than its (contracted) implementor, and certainly not an integration pattern. However, just as IT alignment on business is an entreprise-level pattern, alignment of implementation on initial business requirements and their final validation through functional testing along an initially specified testing plan is an IT pattern.

Tools dedicated to mockup design can be used to design non-functional mockups, or standard tools and platforms used to develop simulations that are working “up to a point” in a defined business context.

4. Light and agile besides integration solutions

As a complement to previously studied main integration solutions, here are a few more broader IT solutions but participating to the current light, agile, web 2.0 trend.

4.1. Light web integration standard : REST

4.1.1. Definition & value offer

Resource State Transfer (REST) models the world using “resources” identified by and made available at a unique address (URL), and allows to get and modify their state.

The properties of REST are what allow the Web to scale, thanks to service operations having consistent semantics (“uniform interface”). For instance, a GET operation on a given resource URL is known not to affect the state of its server, therefore its result can be cached by the client.

See the original thesis of Roy Fielding defining REST at http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm .

4.1.2. Composition features

REST has a workflow style called HATEOAS (Hypertext As The Engine of Application State). It consists in each workflow task being a resource on its own available behind its own URL, and returning in its state the transitions that are available to the user from this task to a next one. It's actually merely an application-level extension of the properties of hypermedia. See an introduction to HATEOAS at http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven .

4.1.3. Code and data

REST has no logic per se besides using HTTP methods, which can be scripted in any language. However Javascript is omnipresent on the client side and appearing even on the server side, with servers such as node.js.

REST has no notion of service contract or data format per se, but the closest thing is the media type. At high level, a resource state may have different representation, with media type being used to negotiate the one the client wants, allowing for instance a web browser to get an HTML web page while a program can get a JSON or XML file, all representing the same state. At finer level, media type of “application/javascript” would ensure JSON format, but not enforce any kind of business-level typing, while a media type of “application/json+nxrequest” implies further semantics of a Nuxeo Request, all to be custom checked on client and server state. Further, media type negotiation allows for adaptable clients : an XML-only client can require an XML representation of a resource rather than a JSON one (representation format), an older client can require the older representation format that was the norm when it was developed rather than the latest version (representation version). Small business impedance could even be handled using this avatar of routing.

4.1.4. In the Information System

With all due caveats, REST / HTTP could be described like SOAP without envelope (but using

HTTP headers instead), with WSDL contracts having only operations named POST, GET, PUT, DELETE with typical CRUD (Create, Read, Update, Delete)-like semantics and typed by media types, and addressable sets and atoms of data ("browsable API"). Various standards-to-be, WADL (Web Application Description Language, see http://www.w3.org/Submission/wadl/) being the most prominent, try to fit in here. Most service platforms now provide a REST alternative to their SOAP services, and Java has even server-side standards for that (JAX-RS : Java API for RESTful Web Services, see http://en.wikipedia.org/wiki/Java_API_for_RESTful_Web_Services).

However, since REST is merely an architectural style – some call it Web Oriented Architecture (WOA) and not than a standard, there is no automatic way to assess whether a platform is REST or not ; we'll rather say that it is "RESTful", meaning it bears some REST properties. In fact, a lot of platforms and uses called "REST" are actually only RESTful, which, as long as it's clear, can be a good thing if the REST properties kept and those dropped are appropriate in the context of a given architecture - for instance, using POST instead of GET because of its limitations (GET have fairly limited parameter length because of HTTP URL's maximum size of 1024).

REST is actually interesting for SOAP, though not at all as a drop-in replacement to SOAP, but because it reminds that all that SOAP allows have not to be coupled to a single enabler like WSDL. Some are only of interest to clients, like generation of client stub, of test application or UI, client-side validation, client-side service enablement and authentication. Some are only of interest to servers, like generation of server skeleton, of test simulation, server-side validation, provider and authentication configuration. SPoRE (Specification for Portable REST Environment, see https://github.com/SPORE) try to fit only the client-side features, whatever the server platform, for "free" for several script languages once a declarative "spec" has been written describing the service. Of related interest is RxSchema (see http://rx.codesimply.com/), as an attempt to define REST & JSON data format in such a decoupled way.

4.1.5. Versus SOA

REST does not by default adequately provide SOA benefits, which come with having a well-defined service contract. REST media type can be used to allow for service negotiation, but in itself are no contract, because it is not enforced by default. However, with such a contract (WADL comes to mind, but anything that can be easily enforced could do, including bare XML schema, JAXRS on server side or SpoRE on client side), a REST platform sure can allow for SOA – the point being that using a REST platform has to be grounded on other good reasons.

And there indeed are good reasons to choose REST. Some can be deduced from what's already said, some are listed below for our own case.

In addition to those, and outside SOA altogether, REST is very interesting to pure players whose business is to provide online APIs ; not only is "using the web as platform" a very good common denominator, but REST allows them to deploy evolutions without breaking clients (in that existing clients would either not know about the newly deployed parts of the API, or be generic enough to handle them well) in ways SOA / WSDL wouldn't. This is helped by "browsable" (discoverable) APIs, "adaptable client" (thanks to media type negotiation) and "code on demand" (downloadable Javascript) properties of REST web architecture.

4.1.6. For Light

For Light, REST is interesting in how it makes integration easier :

- "Browsable" (discoverable) APIs : making a service "browsable" is using "divide & conquer" to make it simpler ; there is a single root entry, all data can be browsed / discovered by drill-down introspection, and when coming down to a data atom it is simple enough (no complexity in and of itself, simple manipulation). Discoverable APIs and data, both being at some point indistinguishable since representations of data resource can return which operations are available on them, are the consequence of using hypermedia, e.g. linked data.
- Uniform semantics on operations : HTTP methods are very similar to CRUD services, and therefore to data access services ; it just so happens that CRUD & data management semantics are well known of a lot of people, including "data people", and easy to understand otherwise. This is because of REST's "uniform interface" property.
- Good documentation, examples, testing : since there is usually no service contract (or in any way it is not the first way to use a REST service), good documentation can't be avoided by service providers, as well as good examples, all online, and often with a live test "playground" UI as well.
- Lighter data handling, through a format (JSON) that is simpler, less typed, efficient and easy to code on (Javascript), easy to render (various template engines).
- Web technologies are well known : HTML pages & forms, CSS styling, HTTP, AJAX, Javascript & JSON...
- Caching on client side : not OK for everything, but for things like BI & mashups at least yes.

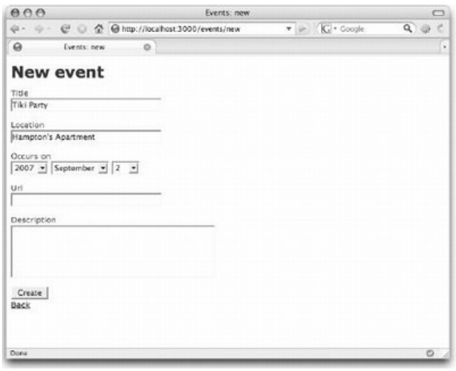
4.2. Light web application development : RoR-likes

The Ruby on Rails framework, as Django (in Python) & Play! (in Java) alternatives provide light & agile development of web applications. It is appropriate to business prototyping, short-term & "throwable" developments and therefore situational applications, but not for long-term maintenance developments.

It does so thanks to :

- script & templates rather than languages "slow" to compile or requiring "heavy" tooling
- light setup through "convention over configuration" mindset, using established design patterns
- continuous deployment
- "free" back office UI using dynamic or generated (scaffolding) web pages
- embedded development environment, including development framework command line shell and (optional) code editor

For Light, RoR-like frameworks are interesting because it provides for applications a development style similar to the one Light promotes for integration, and therefore a natural companion as soon as ones develops not only a light integration anymore, but a full, business-specific light application.



New event

Title: Tiki Party

Location: Hampton's Apartment

Occurs on: 2007 September 2

Url:

Description:

Illustration 9: scaffolding helps rapid prototyping

4.3. Light data storage : noSQL

Lately, the noSQL trend has gained momentum. What's “light” in it is :

- content-oriented data stores, from key-value tables to hierarchical typed structures (the descendants of “XML databases”)
- often no or “lighter” ACID & transactional properties
- easy access from web UI & APIs (REST e.g. JSON / HTTP)
- easy mashupping, scripting, templating
- often “potential” web-level scalability

5. Putting it all together

5.1. More than the sum of products

First of all, one could say, based on all these studies, that to get EasySOA Light, it's just a matter of putting all together a mashup solution for its audience and value offer, an ESB to handle interoperability with existing SOAs, ETL to help it handle data impedance ; and to get EasySOA Integration, it's just a matter to integrate a BPM solution and be able to play well with live external ESBs ; and in order to manage impedance between service models, a shared meta model was in order, as ECM are able to provide. At the root, this is true. But mashups, BPM, ESB, ETL, ECM are all different solutions, so not only is doing them all far too big a work for EasySOA alone, it does neither make a product, nor truly integrated features or actual use cases. However, it makes for an opportunity at their core : an agile SOA registry, which will hopefully open up other opportunities around. So EasySOA is not about making a mashup solution, a BPM solution etc. but rather about making a shared model able to handle them all and seeing how it allows them to benefit to each other.

5.2. The case for an open, flexible, interoperable Service Registry

Service registry solutions have until now been only a secondary consequence of ESBs' runtime registry being in need of operational administration, and therefore made by ESB vendors, not interoperable and useless without their own ESB. In EasySOA, we make the bet that it will change, under pressure from current trends of virtualization, the Cloud and mobile devices. On one side current IT is scaling up by an order and complexity will make it unmanageable without all-around openness, interoperability with solutions and standards, and flexibility not only in technical configuration but also in methodology and IT management workflows. And on the other side even on-premise IT will be handled like “internal outsourcing” and be under pressure to provide clear ROI (Return on Investment), which will make possible SOA's real goal : long-hoped-for IT alignment on business.

Managing non-formal, social collaboration on a shared, flexible model : modern collaborative content management platforms i.e. ECMs such as Nuxeo are the ideal solution for that, and offer much more around content : publishing, workflows, previews, back office & custom UIs.

5.3. EasySOA Light proxy & bridge ESB

ESBs are useful to bridge EasySOA Light with existing SOAs, as well at protocol as at QoS levels, but also a component central to many real world SOAs and therefore in Integration.

For this bridge, a full ESB is not required but embeddability (e.g. in Nuxeo ECM) and modularity are rather the focus, as are web technologies-friendliness, proximity to an SOA model, live reconfiguration – all which FraSCAti provides. The main EIP pattern is obviously here “proxy”, which is very simple to understand but can stretch to answering a lot of needs.

In 2011, embedding FraSCAti in Nuxeo has been studied at https://github.com/easysoa/EasySOA/wiki/Frascati-in-Nuxeo-Alternatives , then an advanced prototype has been built using a Nuxeo-provided "wrapping bridge". Besides, proxy use cases that are interesting for EasySOA Light have been studied and are described at https://github.com/easysoa/EasySOA/wiki/Easysoa-proxy-use-cases .

5.4. EasySOA Integration support levels

In EasySOA Integration, given the heterogeneity of information systems, it would not make sense to set in stone the choice of a given ESB, or any other existing component for that matter, be it BPM, business analysis or further BAM (Business Audit and Monitoring). So EasySOA Integration will provide,

- beyond a bare but full-coverage support using HTTP web services,
- and simple but useful tools helping generic Java platforms,
- an open, on-demand offer based on robust and renowned Open Source components, with exemplar integration of chosen best-of-breed ones such as
    - Talend ESB including its CXF and Camel constituents,
    - the Bonita workflow engine,
    - and Eclipse SOA tooling.

The new generation of BPM has same relation of mutual benefits with SOA than Business Mashups, but on the side of EasySOA Integration rather than Light. ETL is the lever needed to let business users handle data format impedance in the ESB bridge, but also as a Light alternative to web mashup, and up to MDM in Integration.

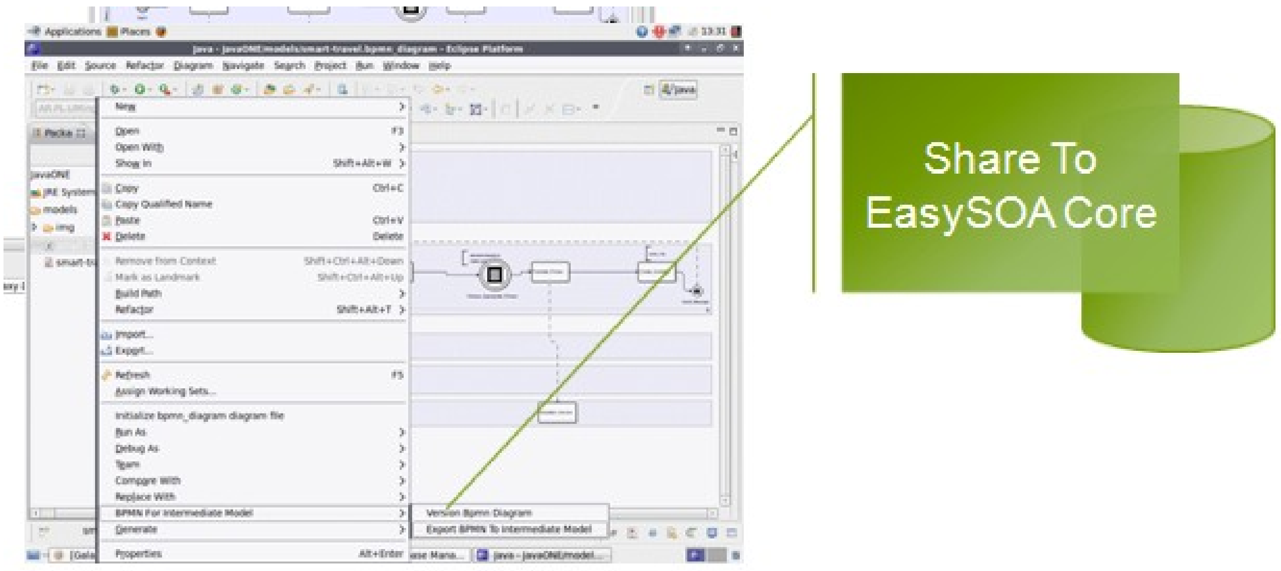


Illustration 10: Eclipse SOA tooling relationship to EasySOA Core

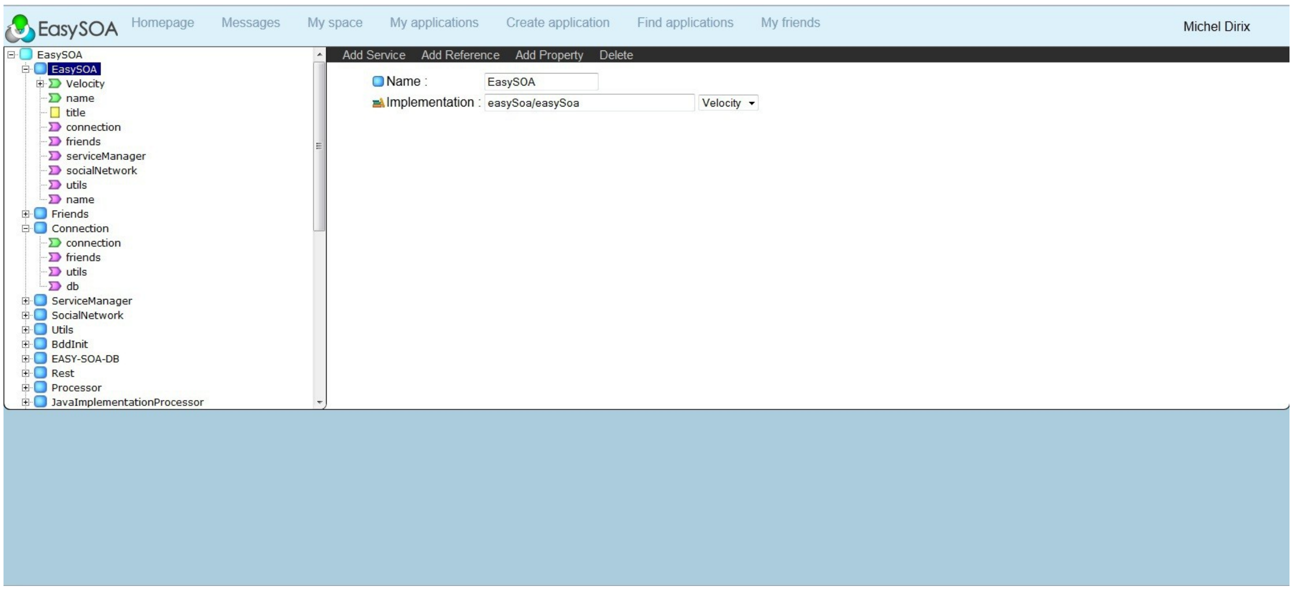
5.5. EasySOA Light's businesses

Business Mashups are a great inspiration to EasySOA Light, because they leverage democratized web UI & integration technologies to give some IT power back in the hand of business users. This same spirit translates to architectural style with REST, to application development with Ruby-on-Rail-likes, to data storage with noSQL and to methodology with agile ones such as Scrum. However, compared to business mashup solutions, which allow to create light applications using existing assets but stop there, EasySOA aims Light creations to help creating them again on Integration side by reusing documentation, business requirements, mockups, working samples, unit tests, business design or even converting & translating further technical artifacts. So it doesn't make sense to create “yet another business mashup server” ; rather, we're looking for the best ways to target business users while providing useful features built on this “transition back to EasySOA Integration” in our platform. Here are the main ones :

- FraSCAti Studio : online web scripting of simple, proxy-like applications running on FraSCAti, in a “social” (shareable) and Cloud-enabled way. Targets “web 2.0 people”.
- HTTP Miner : online tools to test and simplify existing HTTP services, by getting working samples, deriving others from there, automatic or interactive design of test drivers, assertions and simulations. Integrated with FraSCAti Studio, targets any application user of HTTP exchanges, with advanced features (scripting) for developers.
- Automation Service Playground & Chaining : Nuxeo's Content Automation services is the REST platform with the best use case among EasySOA partners, so an online “playground” for these and further REST services, as a complement to their documentation to help Nuxeo developers, allowing to test them in the manner of Google's OAuth playground, would be appropriate and very useful. Besides, there is another opportunity of a Light feature with Nuxeo Studio, which allows online, visual composition of Content Automation Operations in the simplest orchestration pattern of all : the chain. Both target at the same time REST, ECM & Nuxeo users.

We're also looking to :

- target “data & Excel people” and address data format transformation in the Cloud using ETL tools, by deploying Talend (ESB) jobs in FraSCAti Studio
- target “Ruby on Rails & agile people” by integrating an external RoR-like framework on the Light side of the EasySOA development workflow.



Name : EasySOA
Implementation : easySoa/easySoa Velocity

Illustration 11: FraSCAti Studio prototype