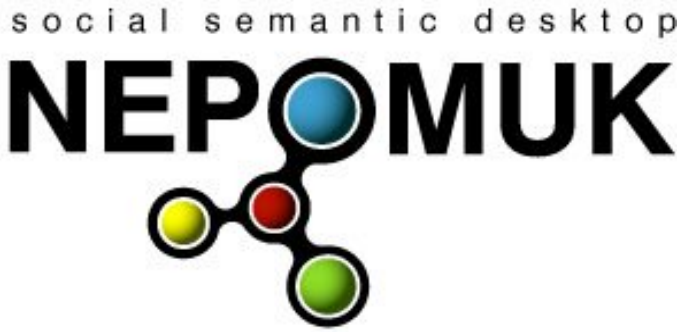


Integrated Project no. 27705

Priority 2.4.7

Semantic based knowledge systems

Creating Rich Semantic Web GUIs

Leo Sauermann

DFKI GmbH, 10.7.2006

Some preliminaries about a Rich Client

Theory:

- Nepomuk at minimum has to be as comfortable as Microsoft Outlook
- and the semantic features are the rock and roll on top

How to implement a Semantic Web Rich Client

- Any modern GUI application uses **frameworks** to generate features,
- for MS-Outlook or SQL-database-driven applications these frameworks are **mature**
- For RDF, frameworks to build GUI applications are scarce, and that's also why Semantic Web GUIs look bad.
- Some have tried before us to make frameworks to build rich clients for the Semantic Web, the approaches and building blocks will be shown here

Example of expected user interface components needed by end-users and developers

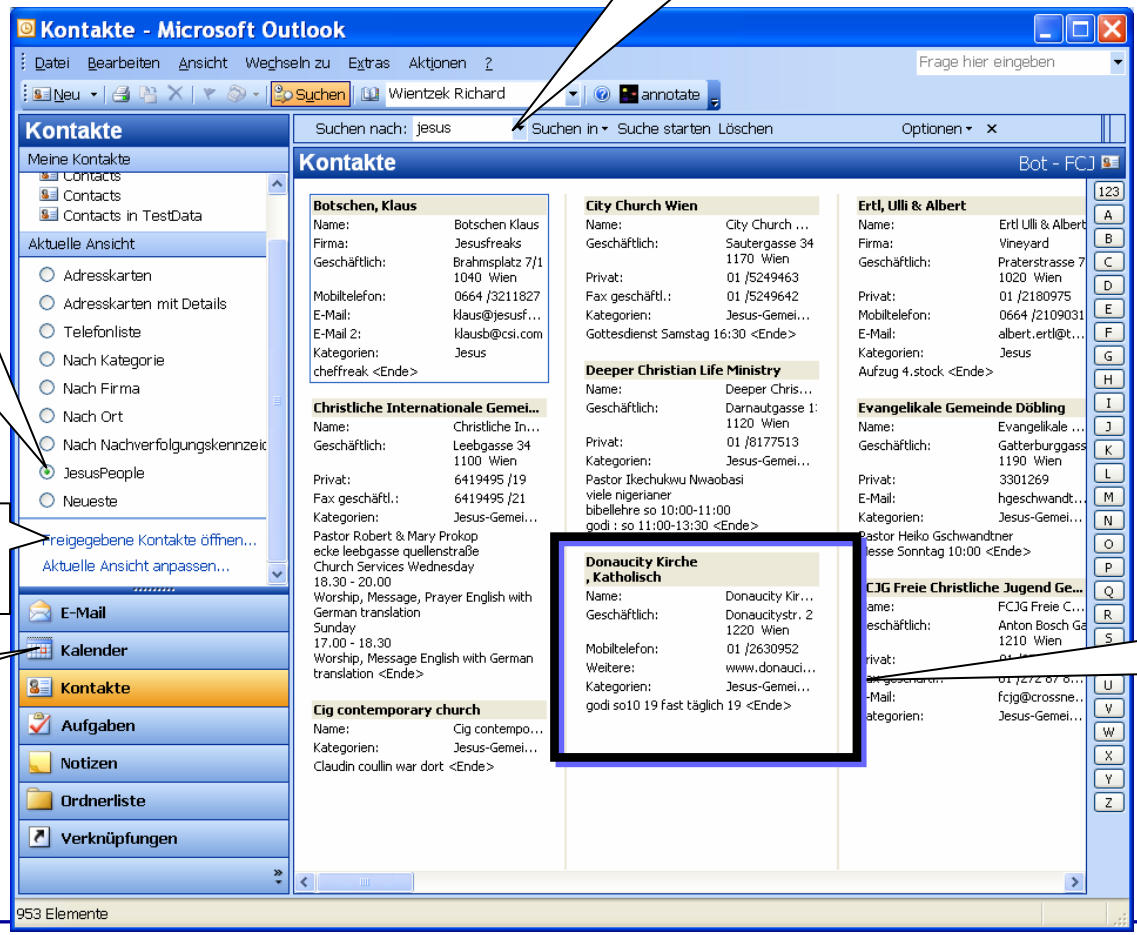
search field

select view metaphor, changeable to list view, more or less details

gui personalization options

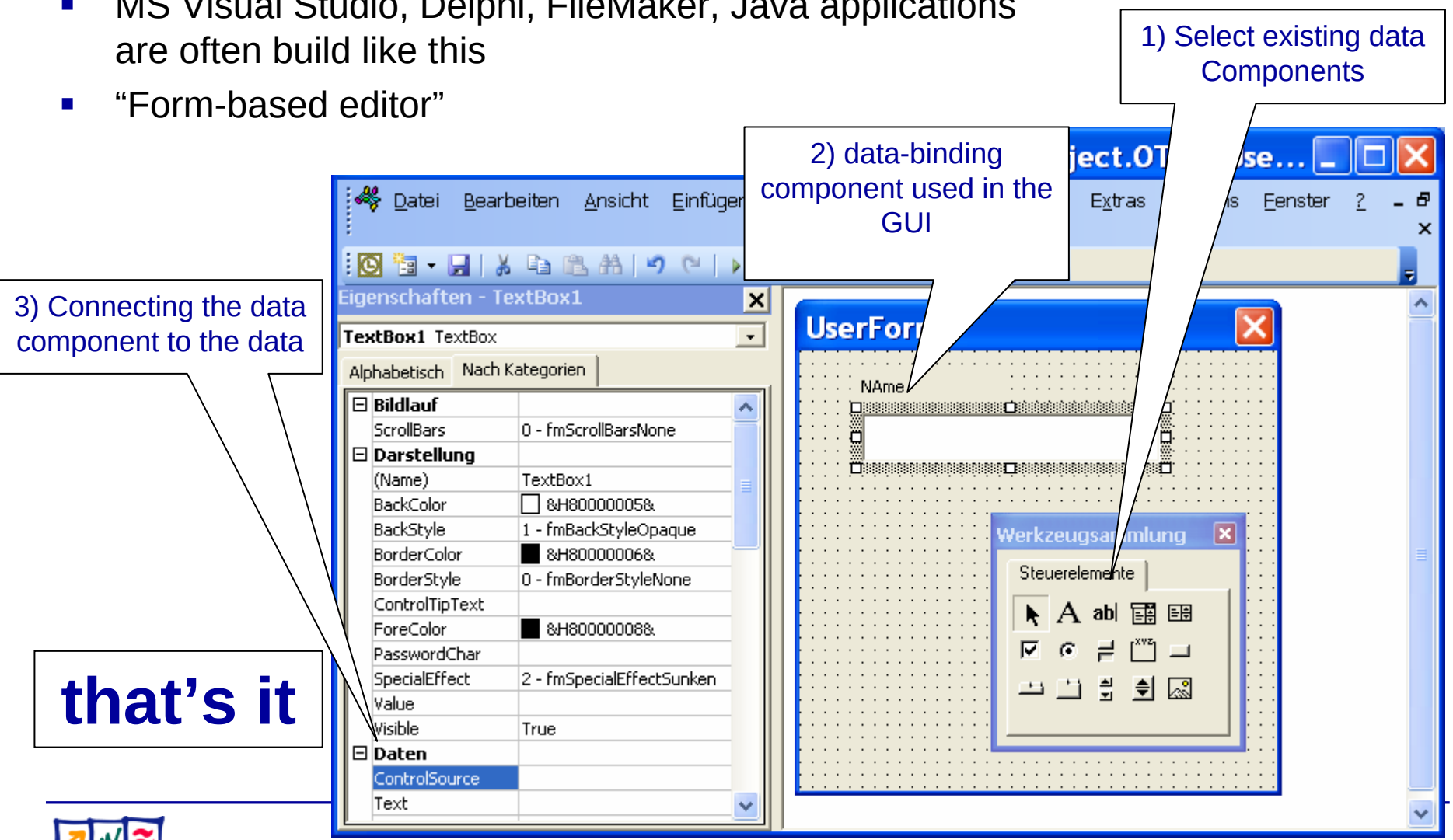
different application plugins for different tasks

Visualisation of one Hit, class “Person”



The Framework to Build such a GUI is a Forms Editor with Data Binding options

- MS Visual Studio, Delphi, FileMaker, Java applications are often build like this
- “Form-based editor”



How some Java or Web Programmers develop, which is not-efficient, hard to maintain, stupid and a waste of precious lifetime

- this is crap and makes IT peoples lives a living hell

```
Form myForm = new Form();
myTextEdit = new TextEdit();

myForm.add(myTextEdit, 10,10, 50,10);



method readData() {
  Database myData = getDatabaseconnection(mytable);
  myTextEdit.setText(myData.getValue("name");
}

method writeData() {
  String newName = myTextEdit.getText();
  if (name == "")
    throw new Exception("you have to enter a name");
  myData.setValue("name", newName);
}
```

The **look and feel** of the application should be designed visually

The connection to the database should be configured, **displaying data and storing changes should be automated.**

Application logic should be expressed in Data Definition languages like SQL or Ontologies

We don’t want to have bad looking user interfaces, and our peers neither

- avoid a naïve approach of “we will just write a gui”
- ontologies and RDF don’t help building GUIs, they make things more complicated. with these complexities, a well-thought approach is needed
- lets look at some people who did really invest in **usability** and **system design** and learn from them

Example: better looking Semantic Web user interfaces

- These **“Browsers For the Semantic Web”** are State-Of-The-Art where we can learn from
  - Haystack / Hayloft
  - Dbin

Haystack user interface

social semantic desktop
NEPOMUK

A Table

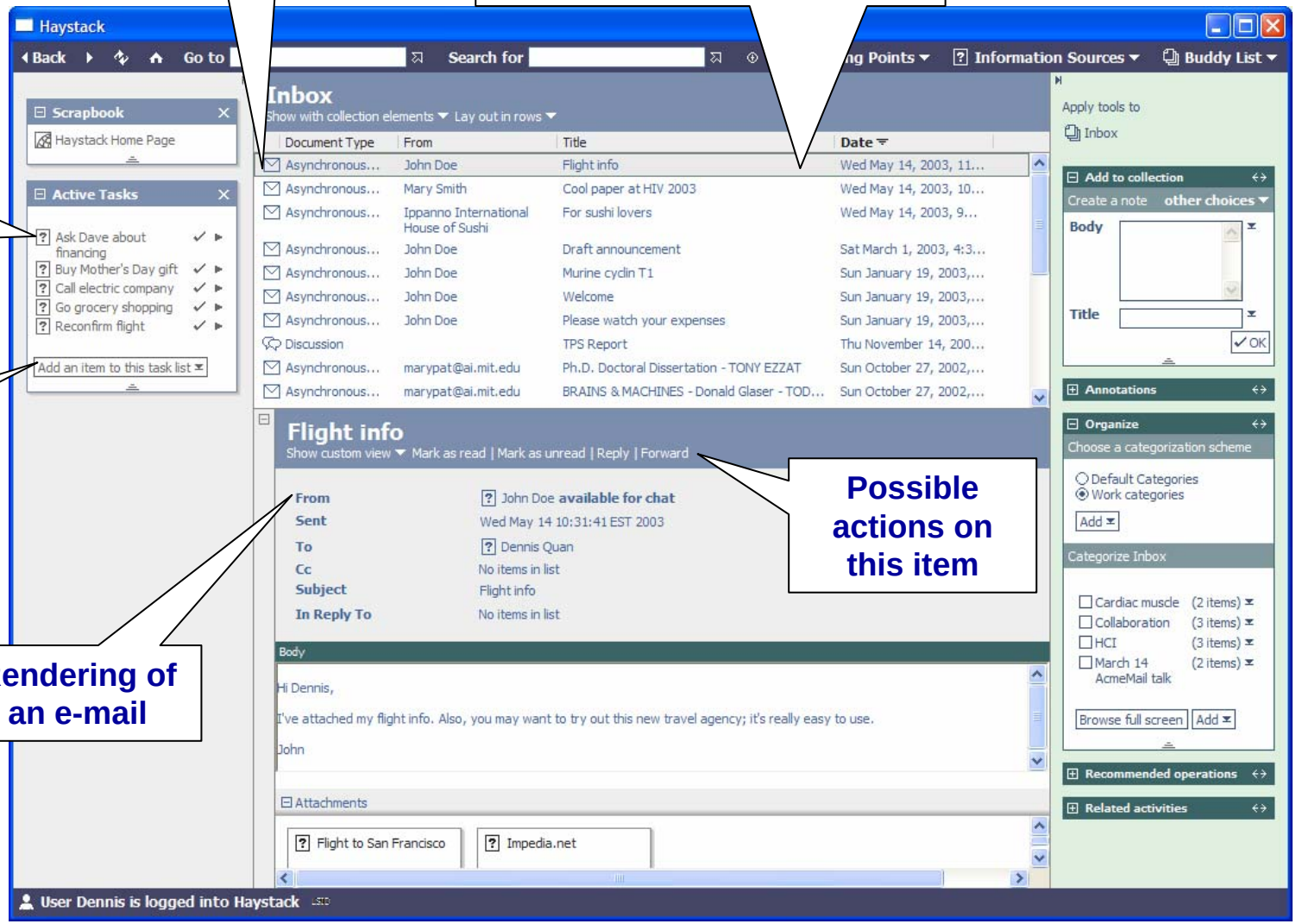
Selection Model:
a thing selected here will trigger the other components to render information about the thing

A list with icons

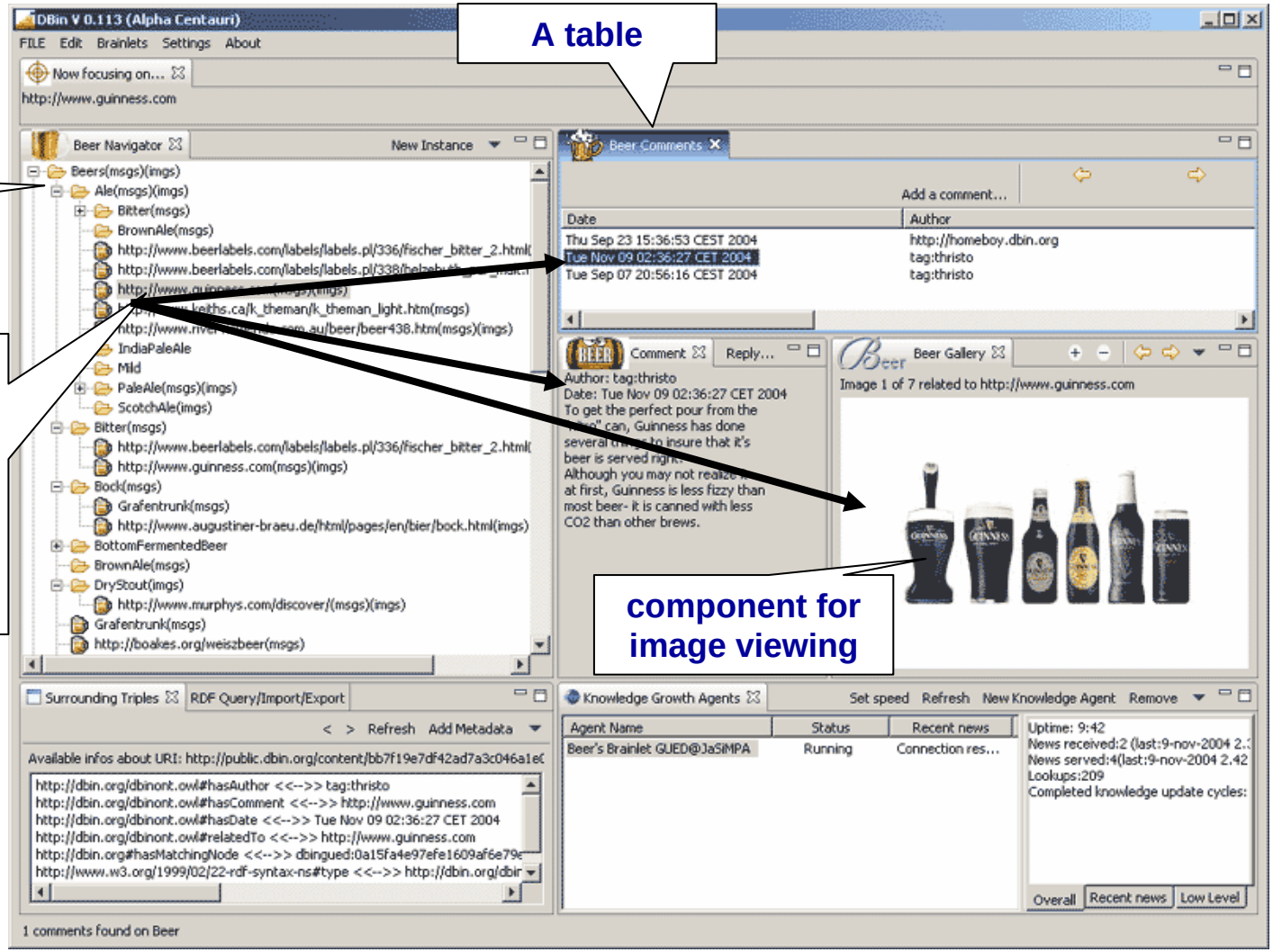
action "add item"

Rendering of an e-mail

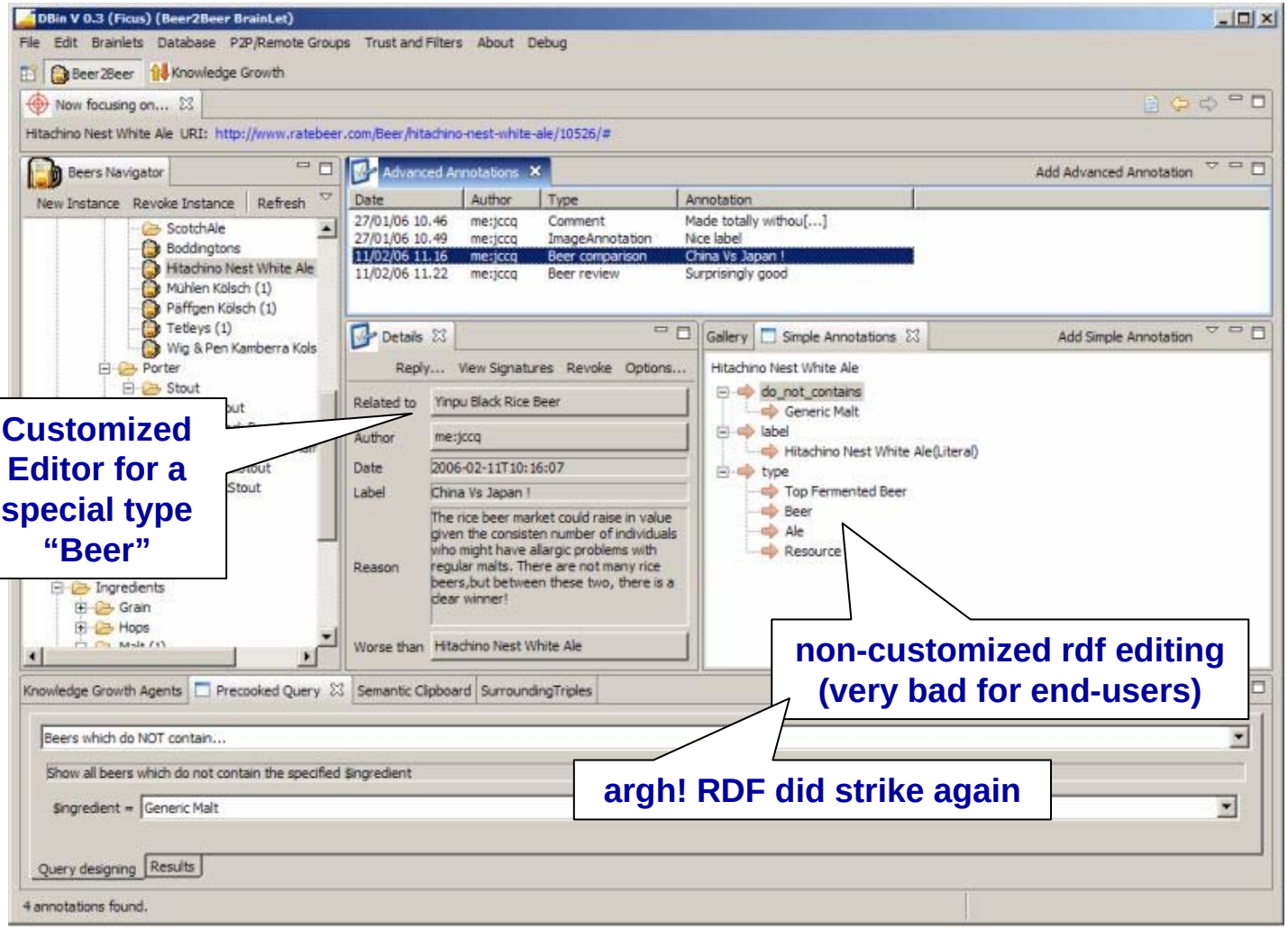
Possible actions on this item



DBIN User Interface



DBIN User Interface



The goal

- A Semantic Web Rich Client that allows easy creation of Semantic Web Desktop user interface
  - using the Rich Client, application development should mainly consist of re-using existing plugins and components
  - a new component added to the system adds value to the existing components
  - Basic Components should be there to build on:
    data interaction, copy-pasting, drag-drop, lists, trees, rendering of any RDF data, defining actions that can be done on resources, etc
- hope: a good rich client basis lowers implementation cost for the Visual Knowledge Workbench, the task manager, miniquire and the case studies.

A middle layer for application logic defines the behavior of the user interface

- Application logic should be defined using a definition language and not code:
  - what is to be displayed: The tree to the left shows people
  - what can be done: double-clicking a person opens another view
  - defined domain specific editors: a thing of type person is edited by showing these fields: firstname, secondname, e-mail
  - additional actions: right-clicking on a person (wherever in the GUI!) shows: “send e-mail to”, “delete person”, “edit person”
  - conditions and tests: when editing a person, firstname and secondname have to be filled out.
- This application description is then executed by a framework, the framework can be extended by plugins to achieve better interaction
- The framework formalizes the **Model – View – Controller** parts
- that’s how Haystack/DBIN work (=Semantic Web browsers)

Existing Open-Source and Standardization Efforts aim at Rich Client GUIs and can be reused, adapted or embraced for Nepomuk

social semantic desktop NEPOMUK

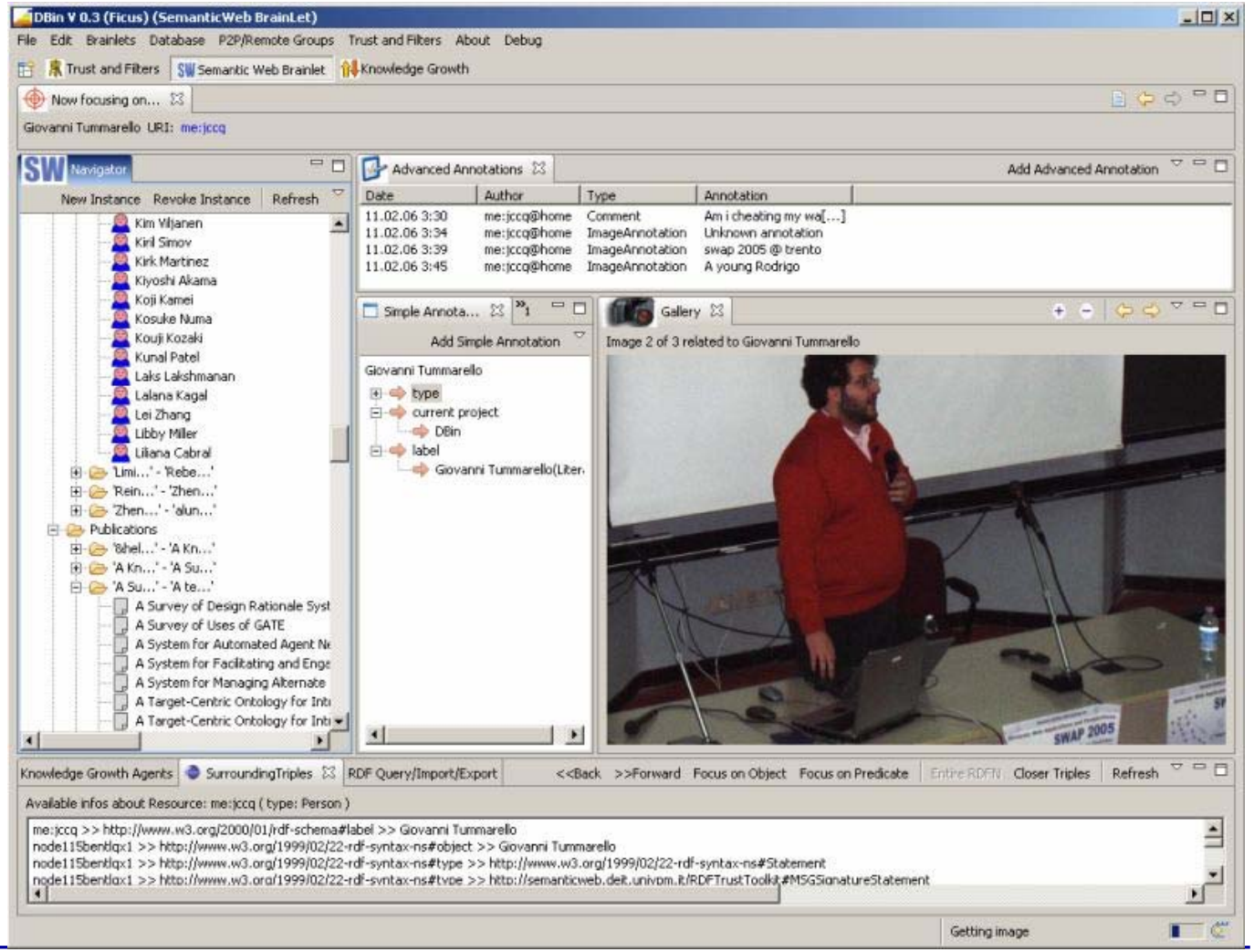
description follow

- DBIN.org Brainlets — Eclipse RCP
- Fresnel: Views and Lenses — MIT
- Haystack/Hayloft and semantic Actions — Eclipse RCP
- Data Binding Components and Forms framework from Eclipse RCP project
- “Chinese Framework” data binding — by DFKI
- Web-based frameworks (web vs richclient)

Example from DBIN: how to build a user interface in 91 lines of code

- the Semantic Web Research Brainlet
  http://dbin.org/brainlets/swbrainlet/
- Scripted Application:
  The application “Semantic Web Research” is created using
  - data: three ontologies and one instance data model.
  - a big tree on the left: showing persons, topics, etc
  - five components that are called when the user clicks into the tree:
    - GUEDEditor
    - Annotations
    - Content
    - Gallery
    - Chat

DBIN GUI: the result. This GUI is based on components and ~500 lines of script



```
<Brainlet
        name="SemanticWeb"
        author="Christian Morbidoni"
        version="1.0"
        uri="http://dbin.org/brainlets/swbrainlet#"
        banner="Aims to be a meeting point fo both people and resources involved in the Semantic Web research area"
        >
        <Ontology file="brainlet/ontologies/foaf.rdf" name="foaf" base="http://xmlns.com/foaf/0.1/"/>
        <Ontology file="brainlet/ontologies/ont.rdf" name="ont" base="http://dbin.org/swbrainlet/additionalOntology/"/>
        <Ontology file="brainlet/ontologies/flink-complete-May2005_schema.rdf" name="flink_ont" base="http://dbin.org/swbrainlet/flinkOntology/"/>
        <Data file="brainlet/rdfdata/flink-complete-May2005_refined.rdf" base=""/>
        <GUED name="SemanticWeb_all">
                <Topic name="Research Topics"
                        uri="http://www.cs.vu.nl/~pmika/socionet#ResearchTopic">
                        <Child
                                query="SELECT X FROM {X} serql:directSubClassOf {$parent} WHERE X != $parent"
                                recursive="true">
                                <Child subjectBy="serql:directType"/>
                        </Child>
                        <Child subjectBy="serql:directType"/>
                </Topic>
                <Topic name="People"
                        uri="http://xmlns.com/foaf/0.1/Person">
                        <Child
                                query="SELECT X FROM {X} serql:directSubClassOf {$parent} WHERE X != $parent"
                                recursive="true">
                                <Child subjectBy="serql:directType" icon="/icons/faccina.jpg"/>
                        </Child>
                        <Child subjectBy="serql:directType" icon="/icons/faccina.jpg"/>
                </Topic>
                <Topic name="Publications"
                        uri="http://www.semanticweb.org/ontologies/swrc-onto-2001-12-11.daml#Publication">
                        <Child
                                query="SELECT X FROM {X} serql:directSubClassOf {$parent} WHERE X != $parent"
                                recursive="true">
                                <Child subjectBy="serql:directType" icon="/icons/doc.jpg"/>
                        </Child>
                        <Child subjectBy="serql:directType" icon="/icons/doc.jpg"/>
                </Topic>
```

Data Definitions

Define the main tree to the left

define the user interface using RDF queries

```
        <View id="Focus" />
        <View
                id="GUEDNavigator"
                title="Navigator"
                icon="icons/SW.jpg"
                selecterFor="main" />
        <View
                id="Annotations"
                listenTo="main"
                selecterFor="annotation"
                />
        <View
                id="Content"
                title="Details"
                listenTo="annotation"
                selecterFor="annotationContent,main" />
        <View
                id="Gallery"
                listenTo="main,annotation" />
        <View
                id="Chat" />
</Brainlet>
```

Interaction definitions, if the user selects a Person in view “main”, it will be editable in the editor “annotations”.

Fresnel: Views and Lenses help to define display rules for any RDF

- Problem: the GUI has to be able to show something nice for any type of resource, be it E-Mail, Person, Task, or Document.
- Solution: A **style-language** defines how to render a type, there are GUIs that interpret the style and create a user interface that looks good
- Fresnel is such a style language
  http://www.w3.org/2005/04/fresnel-info/

Example of a Fresnel definition and its use in a GUI

- Defining a Lens for foaf: show only name, surname and picture

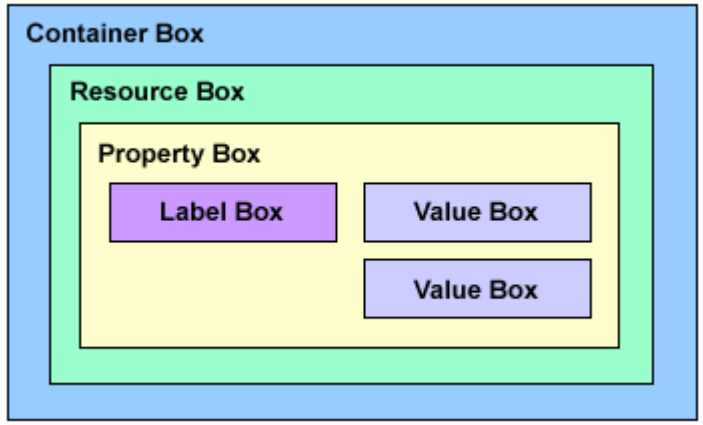
```
f:foafPersonDefaultLens rdf:type fresnel:Lens ;
                        fresnel:purpose fresnel:defaultLens ;
                        fresnel:classLensDomain foaf:Person ;
                        fresnel:group :foafGroup ;
                        fresnel:showProperties ( foaf:name
                                                 foaf:surname
                                                 foaf:depiction ) .
```

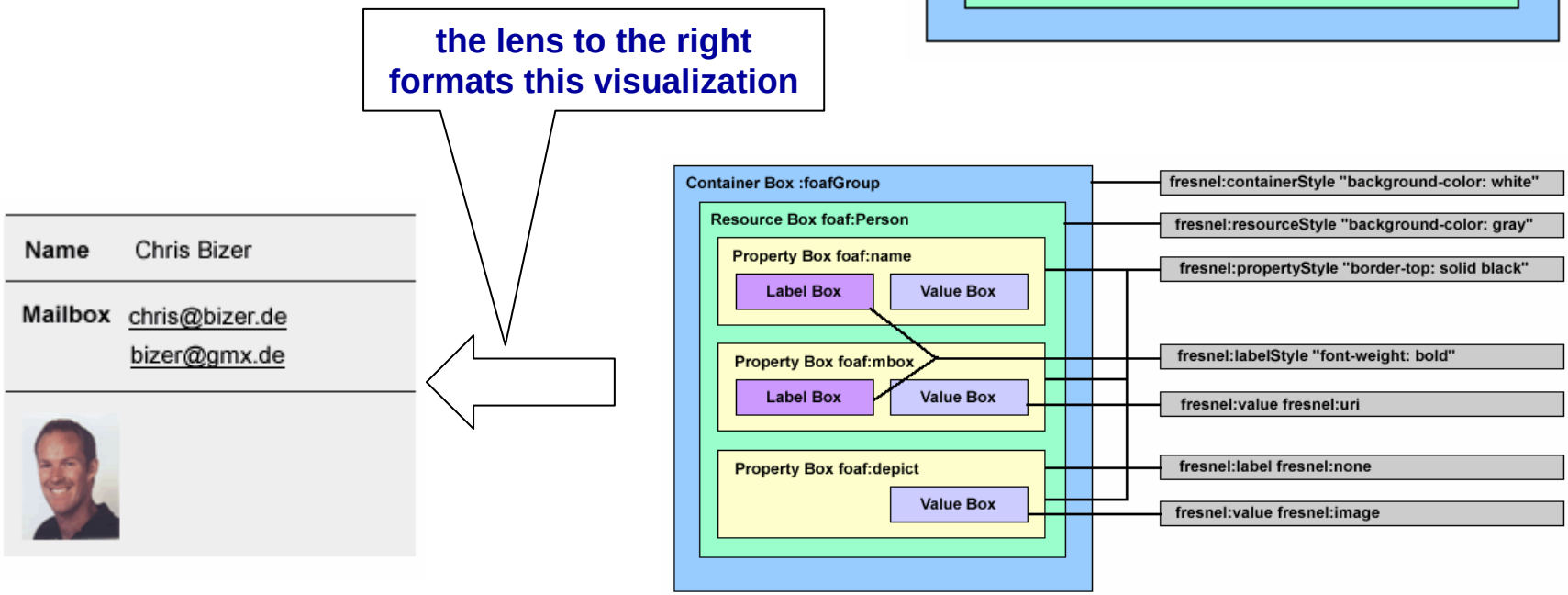
this is an ordered list, saying that name,surname and depiction should be shown in that order

- see http://www.w3.org/2005/04/fresnel-info/manual/

Fresnel

- Fresnel can be used to style the appearance using colors and other CSS features





Considering use of Fresnel in Nepomuk

- Fresnel has been around for a longer time, but the implementations are still in the beginning. look at this page to see that fresnel is not used: http://www.w3.org/2005/04/fresnel-info/#implementation
- Although there are implementations in Java and PhP, we may have problems using them straight away.
- **Conclusion**: Conform at least to the fresnel vocabulary and approach, but extend it when needed and make, if this is needed, an own implementation of the rendering.
- **Chance**: we may waste time here. We will meet many other people from all over the world who are happy to join.
- **Chance:** any definition in Fresnel can be reused in many applications, we hope that there will be an online repository to download many lenses and views to format any data.

Hayloft

- **Hayloft is a rewrite of the Haystack** project, but came to a slow stop when Dennis Quan left MIT and haystack wasn't continued, also Piggy-Bank is now popular and drew resources away
- Hayloft is kind-of-orphaned, but still some people develop it the names **Haystack and Hayloft** nearly mean the same.
- It is clearly the state of the art in Semantic Desktop Rich clients
  - views/lenses implemented
  - plugin system
  - personalization features
  - has its own database, own inference, etc.
  - actions are also defined using ontologies and therefore the "right-mouse-click" approach is very interesting

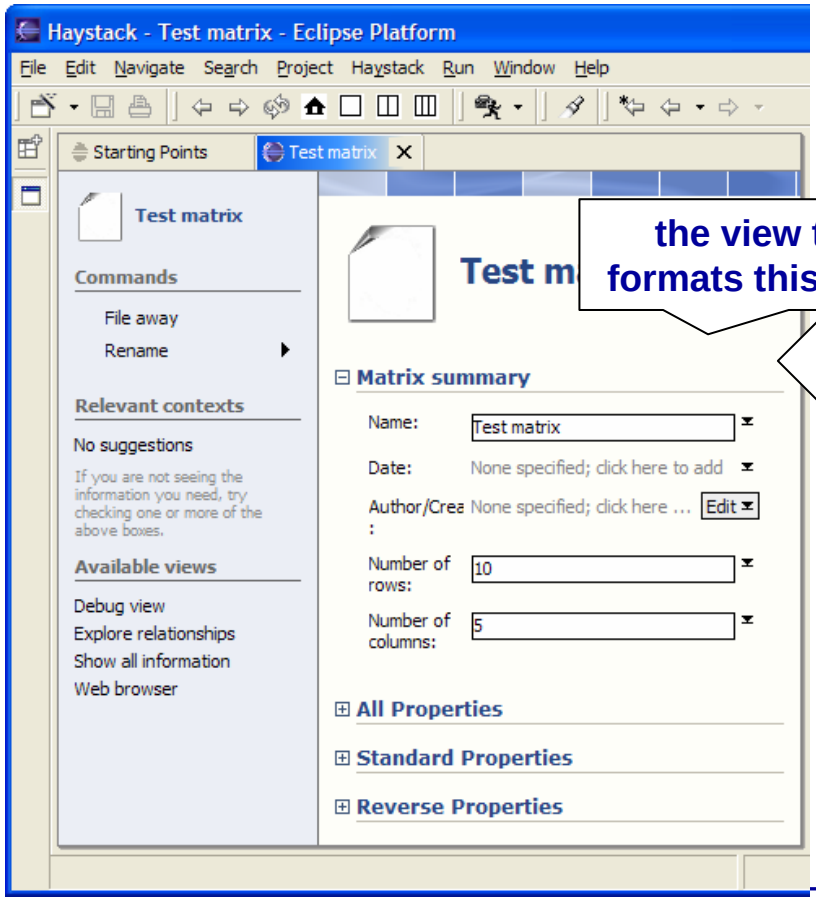
hm, I don't know why haystack stopped, but it seems to have stopped.

What we can learn from Haystack

- take their codebasis from SVN
  http://simile.mit.edu/haystack/branches/hayloft/
- we can also ask them how to reuse haystack for Nepomuk
  http://mailman.mit.edu/pipermail/haystack-discuss/2006-July/000110.html
- Somebody should read this document:
  "*Haystack's User Interface Framework and Guidelines*"
  http://haystack.csail.mit.edu/documentation/ui.pdf
- They really really thought about usability
  http://haystack.csail.mit.edu/overview.html
- The really really documented how to program this
  http://haystack.csail.mit.edu/developers/index.html
  http://haystack.csail.mit.edu/developers/development.html

One of the nicest things in Haystack are Views and Lenses and Actions

- Haystack allows to define views/lenses to render and edit RDF.



the view to the right formats this visualization

```
add { :Matrix
                rdf:type                rdfs:Class ;
                rdf:type                daml:Class ;
                rdfs:label              "Matrix"
}
add { :rowCount
                rdf:type                daml:DatatypeProperty ;
                rdfs:label              "Number of rows" ;
                rdfs:domain             :Matrix ;
                rdfs:range              xsd:int ;
                rdf:type                daml:UniqueProperty
}

add { :columnCount
                rdf:type                daml:DatatypeProperty ;
                rdfs:label              "Number of columns" ;
                rdfs:domain             :Matrix ;
                rdfs:range              xsd:int ;
                rdf:type                daml:UniqueProperty
}

VIEW
add { :matrixSummaryAspect
                rdf:type                metadata:MetadataAspect
;
                dc:title                "Matrix summary" ;
                metadata:propertiesToDisplay ${
                                rdf:type        data:DAMLListSource ;
                                data:damlList @(
                                                dc:title
                                                dc:date
                                                dc:creator
                                                :rowCount
```


Information Society Technologies

With Haystack we can define actions to be taken on the selected resources

- this method inverts a matrix. This is all needed to let the user call the method using right-mouse click or context menus

```
method :invert :matrixParameter = input :newMatrixName = name ;
dc:title                        "Invert matrix" ;
dc:description                  "Inverts a matrix." ;
rdf:type                        op:Operation ;
op:primaryParameter             :matrixParameter
        = size (extract input[0] :rowCount ?x)
        return ${
                                rdf:type        :Matrix ;
                                dc:title        name[0] ;
                                :rowCount       size ;
                                :columnCount    size
        }

add { :matrixParameter
        rdf:type                        op:Parameter ;
        rdf:type                        daml:ObjectProperty ;
        rdf:type                        daml:UniqueProperty ;
        rdfs:label                      "Matrix to invert" ;
        op:required                     "true" ;
        rdfs:range                      :Matrix
}

add { :newMatrixName
        rdf:type                        op:Parameter ;
        rdf:type                        daml:DatatypeProperty ;
        rdf:type                        daml:UniqueProperty ;
        editor:disallowBlanks           "true" ;
        rdfs:label                      "Name for result matrix" ;
        op:required                     "true" ;
        rdfs:range                      xsd:string
}
```

defines the operation

defines that the operations needs the matrix as one input

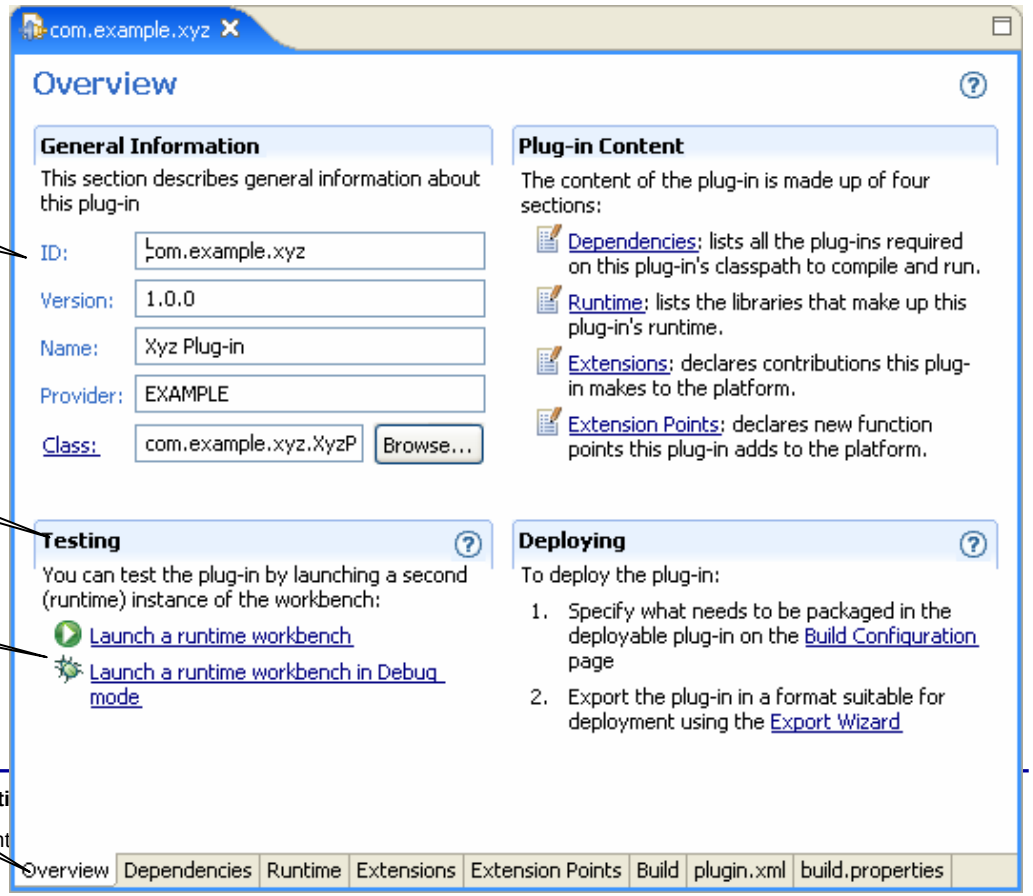
and the name for the inverted matrix as second input

Conclusion on Haystack / Hayloft Recommendation for Nepomuk

- This project is a huge source of useful information. Leo Sauermann did not reuse much of this in gnowsis, which was a mistake
- don’t make the same mistake twice: carefully read the publications of haystack and their documentation, reuse this project as much as possible

Things that are free in life: Eclipse Forms framework

- Eclipse RCP provides a Forms framework that also allows editing values. There should also be a data binding framework to edit beans, but Leo Sauermann doesn't know where it is
- http://www.eclipse.org/articles/Article-Forms/article.html
- This is done with forms
-



nice editing

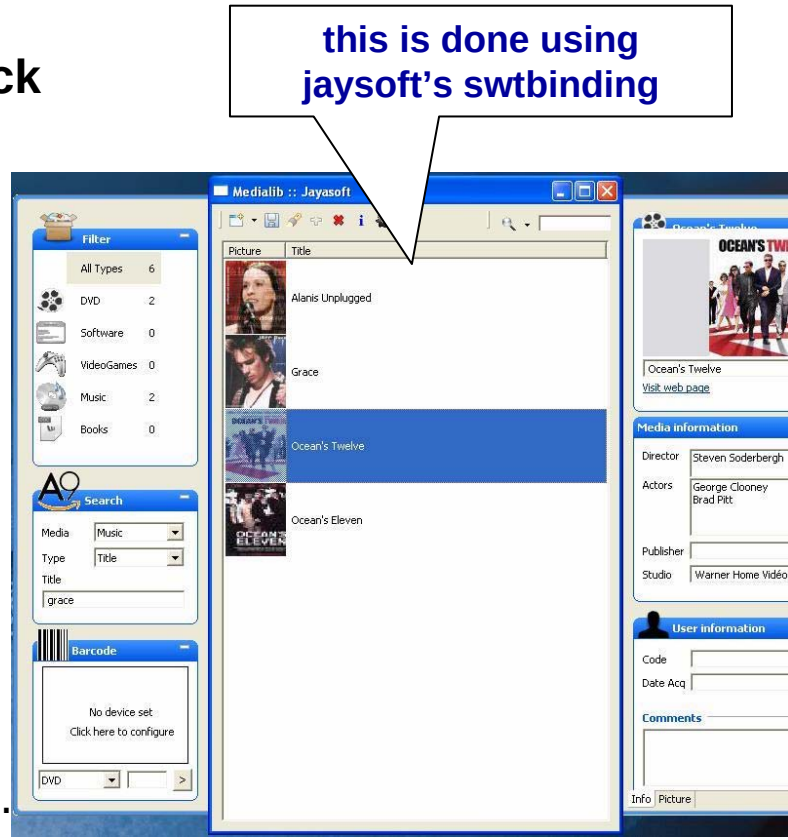
round borders, fancy looking

invoke some methods

create complicated wizards

Eclipse and Data Binding Components

- This library binds JGoodies Data Binding Framework to SWT:
  - http://www.jayasoft.org/swtbinding
  - **this is alphaware and can break our neck if we bet all our money on it**
- JGoodies Data Binding is a way to bind Swing components
  - These components usually listen to **Java Beans**, when a value changes they change the GUI
- Data binding components allow the connection of data to the GUI, similar but not the same as **lenses and views**.



Eclipse Data Binding

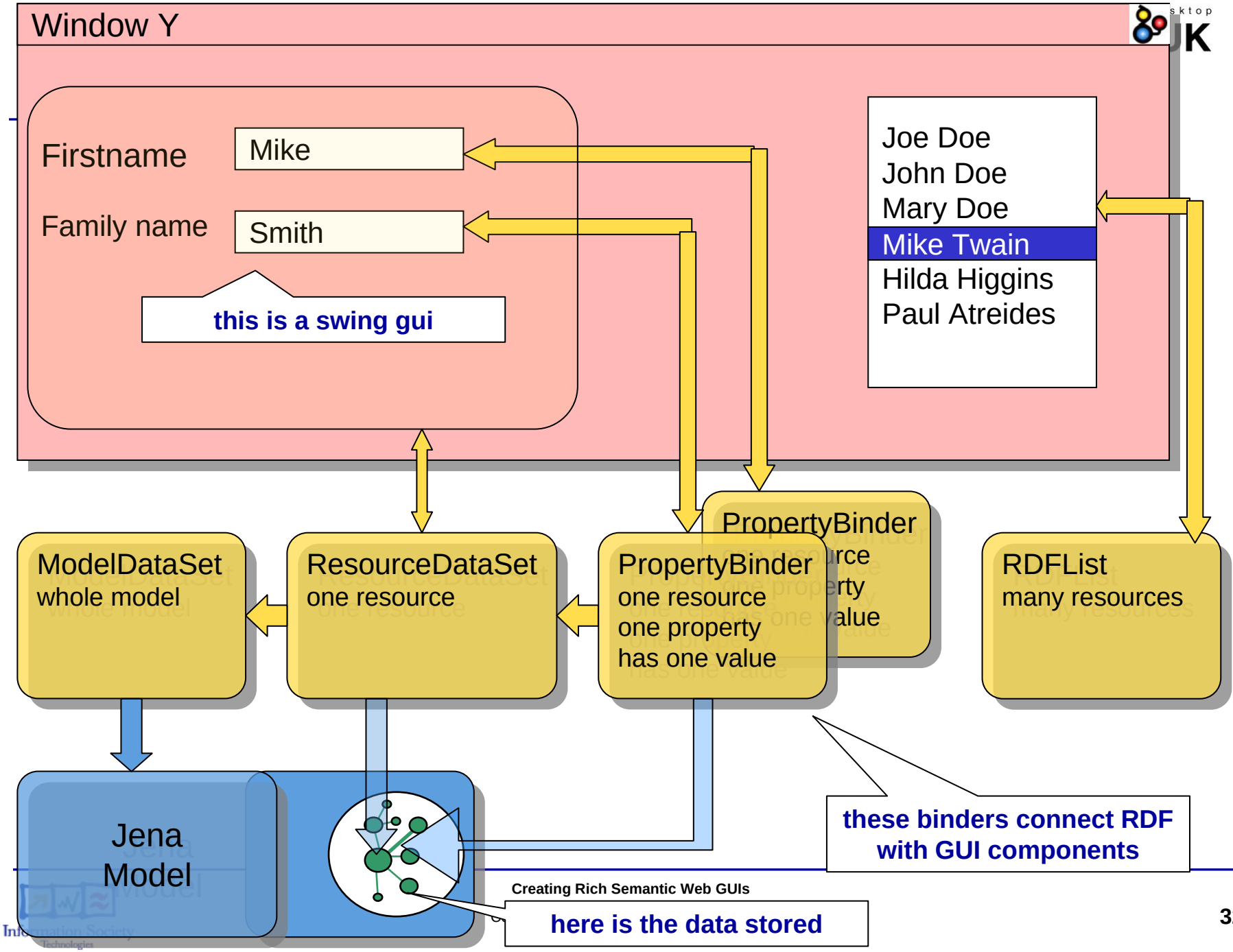
- Another way to bind data to the GUI is to use the IBM/Rational Rose Data Binding framework
  - As easy way to learn what data binding is this very long video (although the speaker is crap).
  - This framework also costs a lot
    Rational Rose + IBM = $ourmoney - opensource
  - http://www.alphaworks.ibm.com/tech/databinding4swt
  - video:http://www.alphaworks.ibm.com/demo/flash/display/databinding4swt0

Data binding Components: Conclusion for a decision

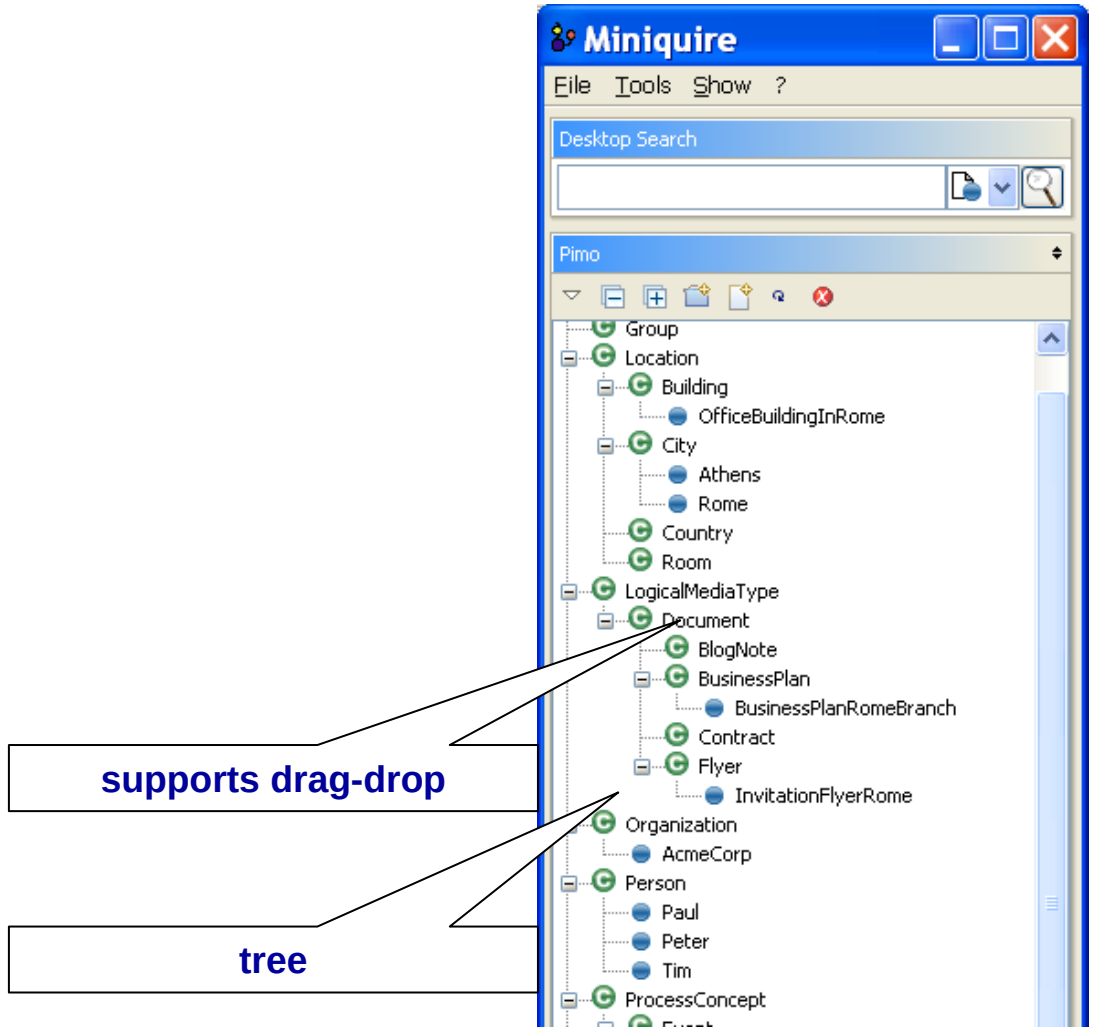
- Existing Data Binding components like IBM's databinding4swt or JGoodies Binding (and the billions of other frameworks that software engineers know) are mostly working on
  - Java Beans
  - SQL Databases
  - Web Services
- Our Data is mostly RDF, so we may think of:
  - extend Max Völkel's RDF Reactor & DFKI RDF2GO that they are like **Java Beans** and use an existing databinding framework on it
  - extend the **RDF-API** that it looks like SQL, then use an existing framework
  - write our **own binding framework** (or use the chinese framework, next slide) that binds RDF to the GUI.
  - extend what **Haystack or DBIN** have done.

The Chinese Framework by DFKI aka “gnogno-components”

- In gnowsis, a data binding framework was created to bind Jena RDF graphs to user interfaces
  http://gnowsis.opendfki.de/wiki/GnognoComp
  - The core of the framework is built of dataset objects. These dataset objects wrap around Jena objects and separate the gui from the rdf database behind.
  - ModelDataSet wraps a Jena Model.
  - ResourceDataSet represents the selected Jena Resource


Window Y
Firstname
Mike
Family name
Smith
this is a swing gui
Joe Doe
John Doe
Mary Doe
Mike Twain
Hilda Higgins
Paul Atreides
ModelDataSet
whole model
ResourceDataSet
one resource
PropertyBinder
one resource
one property
has one value
PropertyBinder
RDFList
many resources
Jena
Model
these binders connect RDF with GUI components
here is the data stored

This GUI was created using the Chinese Framework



Another reason to go for Eclipse RCP and Semantic Web: a possible community

- SwEDE:Semantic Web Development Environment
  http://owl-eclipse.projects.semwebcentral.org/

(all found via http://www.eclipseplugincentral.com/)

Web based User Interfaces – why don't we just program using HTML and python?

- for the goal-oriented, this approach seems to be the obvious one because
  - web interfaces look good (css, images, round corners!)
  - web interfaces use cool technology like AJAX, google maps mashups and firefox
  - web interfaces can be instantly reused for the WEB – hey!
  - web interfaces can be easily combined – program this in python, the other in java, copy the html together

- **BUT …**

Web interfaces have drawbacks that can be a show stopper for some features in Nepomuk

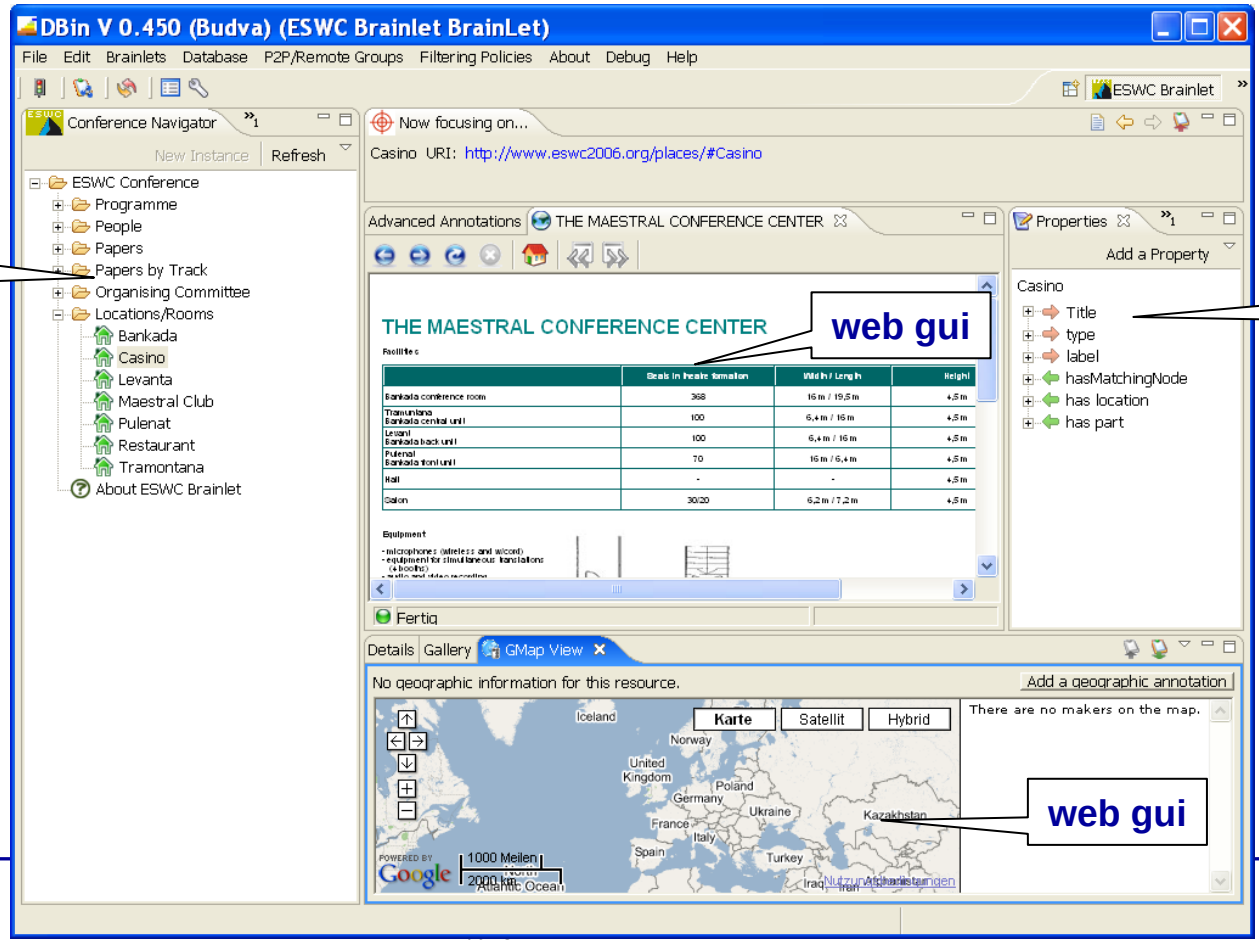
- You can **never drag/drop** from one application to another
- you can **hardly copy/paste** complex data (some flash-based rich editors can, but that's overkill)
- Although AJAX is cool and Google Maps looks even better it takes around 200.000 lines of JavaScript code to write it. Coding this gui was the nightmarish hell of a batallion of developers. Javascript is hard to write and hard to debug
- we would still need frameworks (struts, JSP, JSF, zope …)
- **BUT …**

Web interfaces will be used whereever possible to reuse code in the web

- Compromise
- It is possible to use the **rich-client interface** for features that are typically done in a rich client
  - Editing Complex Data. Domain Specific applications like iPad.
  - Annotating files from the filesystem using drag-drop
  - Combining many plugins for complex systems like task management
  - sidebar
- Use **web interface** for features that are typically web
  - semantic wiki
  - searching, information retrieval, faceted browsing
  - debugging interfaces - hacked much quicker in jsp ☺
  - web browsing, web annotations, photo annotations, social software etc
- The golden middle way: Embed web features in the rich client using a web-browser. Example: DBIN's google maps mashup.

Middle Way: combine web interfaces with Rich Client components

- DBIN example:
  web and Eclipse live together in semi-perfect harmony



Proposed Middleware Framework to formalize Model-View-Controller

social semantic desktop NEPOMUK

Model

The model is primarily stored in the **Nepomuk RDF Repository**.
To edit data, a temporary model may exist that represents data currently viewed/edited

contains
- Ontologies (classes, properties)
- Data (instances)
- View-Descriptions (view/lens)
- Formalized application logic (rules, checks, operations)

Haystack/Dbin ontologies

Manipulation via PIMO-Service
- insertInstance(…)
- deleteInstance(…)
- updateInstance(…changes)
- addOntology(…)
-

this is purely Nepomuk

Controller

Gets input from the view, when data changes. Stores model manipulations in the model.
- create a view
- store changes
- manage possible operations on data
- checks validity of input based on business rules/ontologies

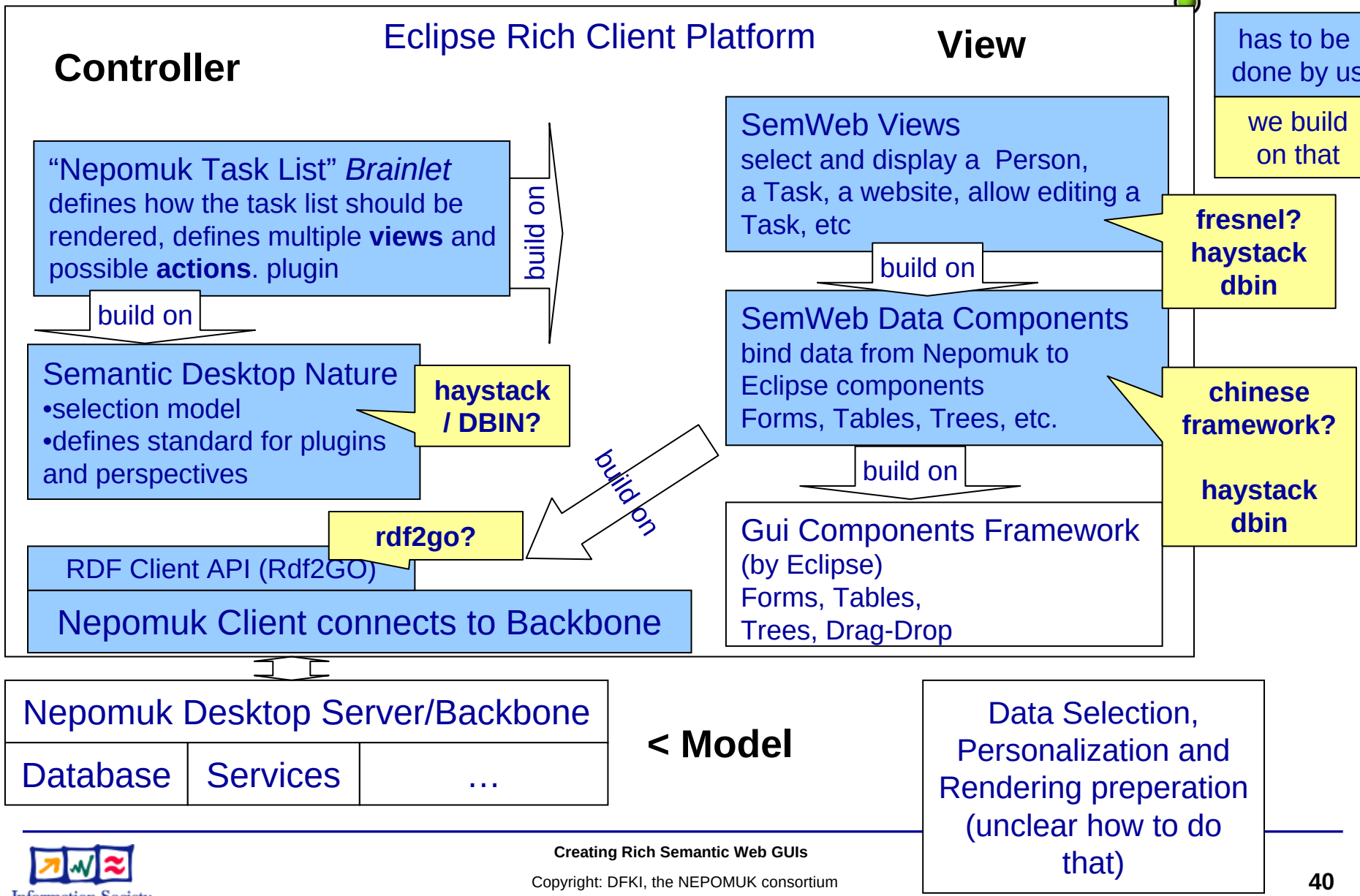
could base on
- **Haystack View/Lens and operation definitions**
- **DBIN interaction definitions**
- **Fresnel Lens/Views**
- **Eclipse Actions, RCP, Views, Plugins**

View

The visual components showing instance data, interpreting mouse clicks and keyboard events.
- generates User Interface components based on descriptions of user interface selected by controller
- listens to changes in the model
- invokes methods of the controller
- assists user based on ontologies/rules

could base on
- **Haystack GUI implementation**
- **DBIN plugins**
- **Fresnel implementations**
- **Eclipse Forms Framework**
- **Bean Data Binding Components**
- **WEB guis!**

Possible software architecture to build a Nepomuk rich client



How to implement iMapping based on this proposed framework

- iMapping is programmed as plugin to the Nepomuk Eclipse Rich Client
- **iMapping is a new view** on the data **model**, supporting zooming, etc using the piccollo framework. It uses the same **controller** objects and logic as other components
- it can integrate easily with conventional (Haystack-like) Eclipse plugins.
- For actions, navigation operations, styling and look-and-feel, etc. we build common components for the RCP, these are reused by iMapping.

Suggestions for Nepomuk

- This was much information. This was important information.
- We have state-of-the-art work and predecessor projects **we cannot ignore: Haystack, DBIN,** gnowsis,Eclipse
- We can save a lot of time by reusing the right framework
- For the rich client, we have to evaluate the related work, to build web interfaces we also have to evaluate related work (powl, rhizome, etc)
- Plan?
  - Until October we need a decision, in January we prepare the components and the framework and document our decisions, from August 2007 on we implement the GUI.