

# Organization Modelling Environment



Eric Yu & Lin Liu  
Knowledge Management Lab  
Computer Science Department  
University of Toronto

<http://www.cs.toronto.edu/km/ome/>



# OME2 & OME3

- OME2 supports the modelling and analyzing of  $i^*$  models.
  - SD: insurance, assurance, and enforcement relationships
  - SR: propagating workability, viability, and believability.....
- OME3 has been designed with an inherent dynamic nature and extensibility, it can support any frameworks defined on top of OME meta framework.
  - NFR (Non-Functional Requirements Framework)
  - $i^*$  (Strategic Actor-based Modelling Framework)
  - GRL (Goal-Oriented Requirements Language)
  - Annotation, reduced ER, ...

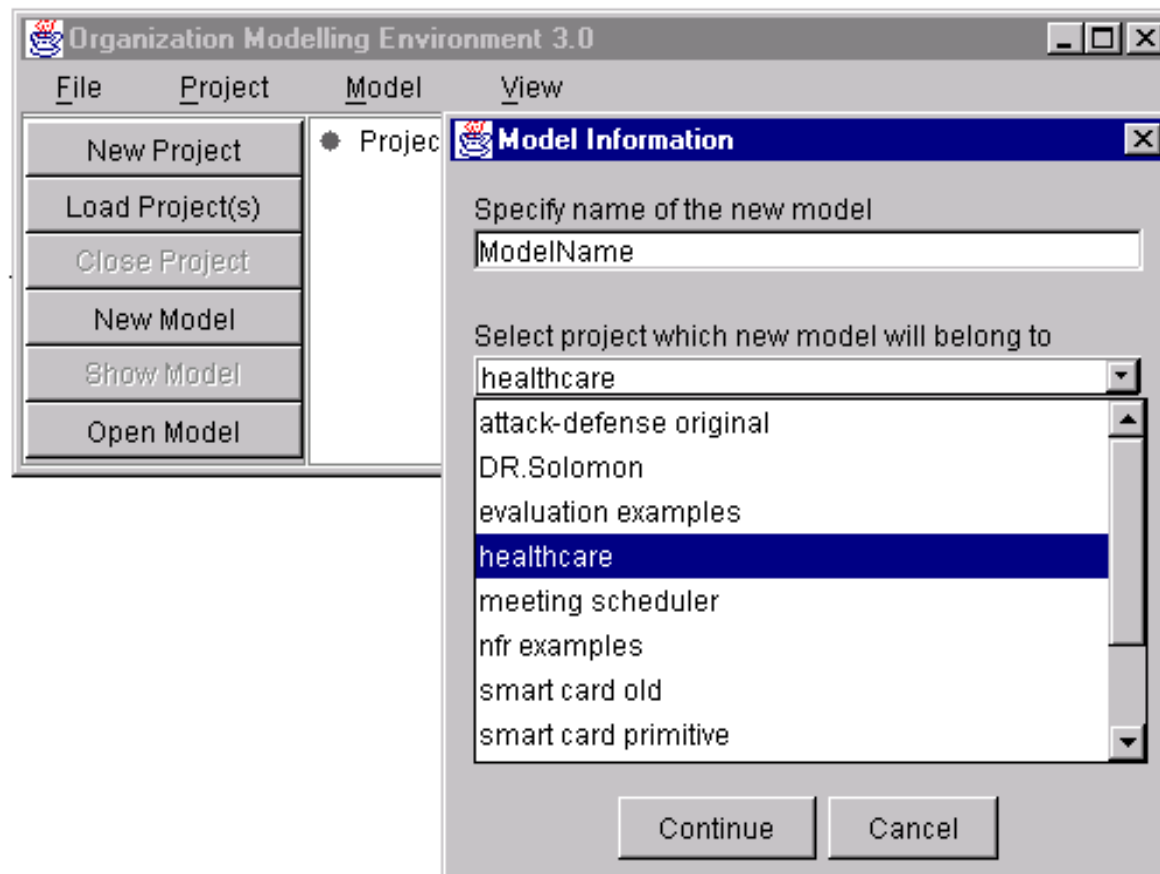
# Goal of OME3

- As a conceptual modelling tool, OME helps
  - Building graphical models with a GUI
  - Managing domain knowledge and software Knowledge
- As a model analysis tool, OME supports
  - Modelling and reasoning on system properties of interest
  - Guiding the design of organization as well as system
- As a framework developing tool, OME supports
  - Describing new kinds of frameworks
  - Development of framework specific modelling and reasoning mechanisms

# Tool Interfaces (1) : To Developers

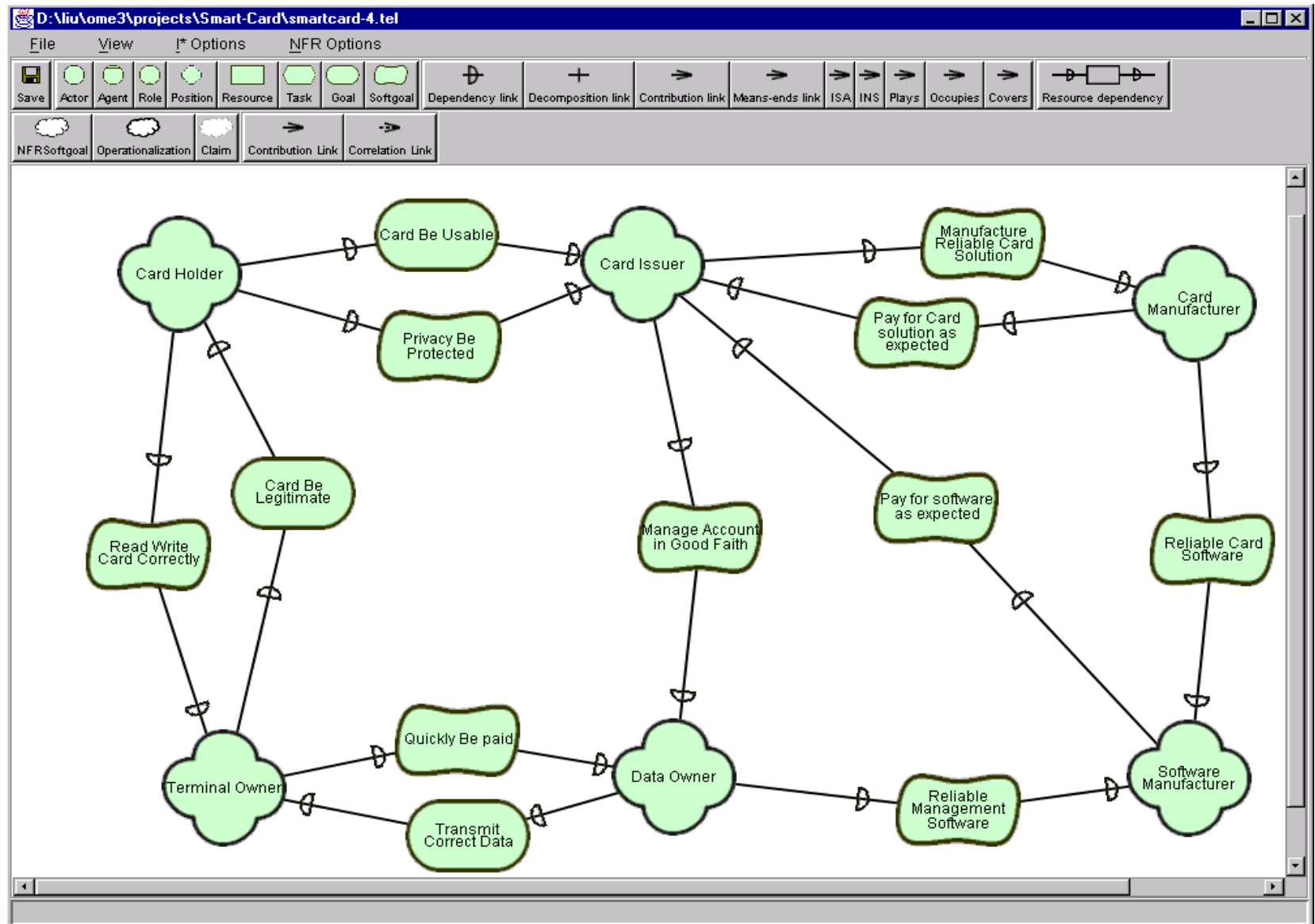
- project/model management interface

*create and load projects and models, put models into project folders*



# Tool Interfaces (2) : To developers

## ■ Graphical model Editor



# Tool Interfaces (3) : To developers

## ■ Model evaluation

The screenshot displays the NFR software interface. The main window shows a diagram with nodes and relationships. The nodes are represented by cloud shapes containing text and labels. The relationships are represented by arrows. The nodes include:

- Accuracy[Received(summary)]  $w^+$
- CorrectIncomingInformation[summary]  $w^+$
- Information[Summary]
- Certification[Summary]
- Certification[LowSummary]
- reimbursement-related activities

The relationships are labeled with "Or" and "Help". The "Certification[LowSummary]" node is highlighted with a red box. The "Impact Resolution" dialog box is open, showing the following steps:

**Step 1:** Use this area to resolve the weak impact(s) for Certification[Summary]

Impact	Source
None	N/A

Resolve Weak Impact(s) to:

X  $w^-$  U  $\approx$   $w^+$  ✓

**Step 2:** Use this area to set label for Certification[Summary]

Impact	Source
✓	Contribution from Or nodes

Set Label for Certification[Summary]:

X  $w^-$  U  $\approx$   $w^+$  ✓

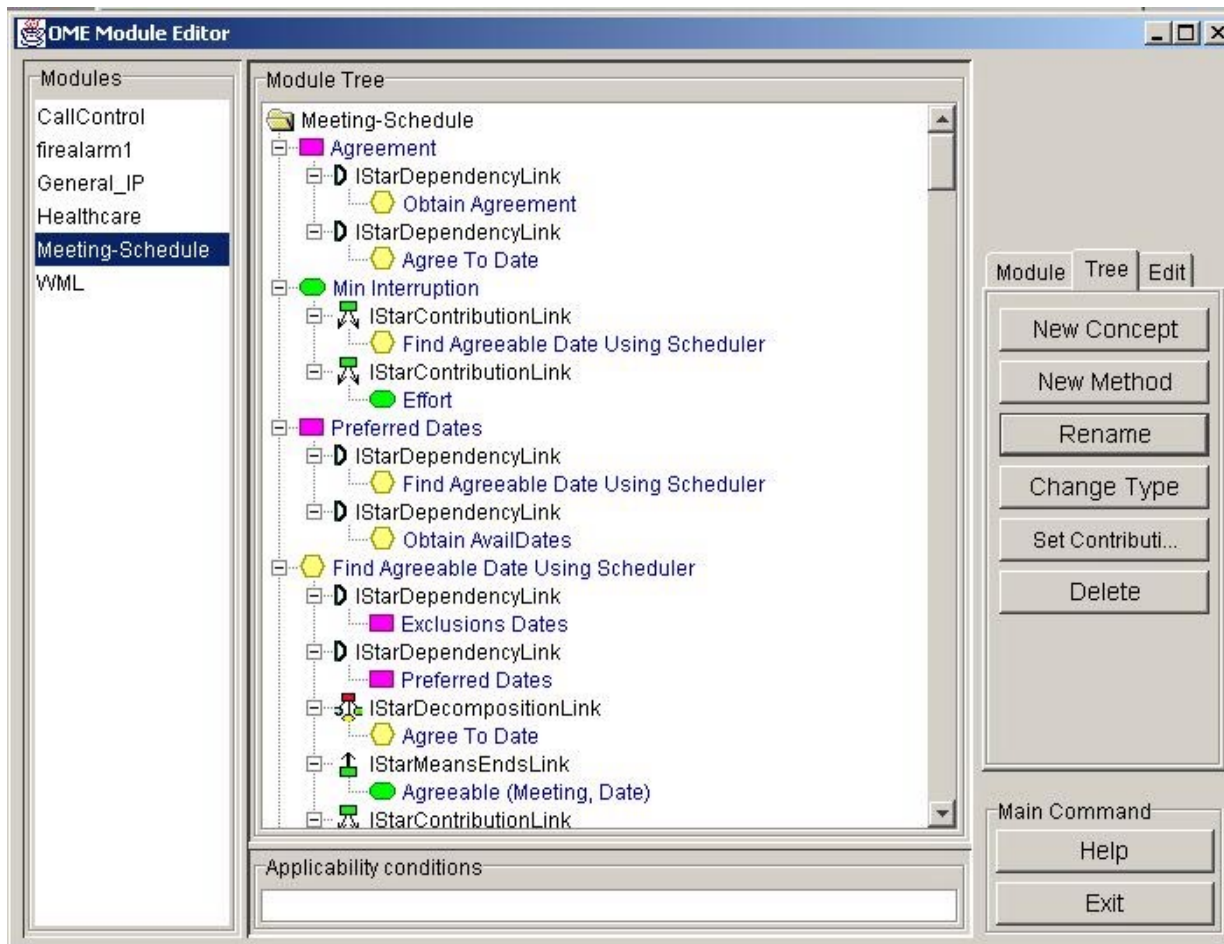
Recommended label: ✓

Keep propagating upwards(if necessary)

Accept Cancel

# Tool Interfaces (4) : To domain experts

- Domain catalogues management interface



11/4/01

CASCON 2001

# Tool Interfaces (5) : To theoreticians

## ■ Telos Language & ConceptBase

```
{
*-----
* Instances of GRLSoftgoal
* Thu Aug 9 16:54:59 EDT 2001
*-----
}
```

Individual GRLNonIntentionalElement in Class with  
attribute  
alias1: String  
end

Individual GRLSoftgoal in Class with  
attribute  
obj: GRLNonIntentionalElement  
end

Individual GRLAndContribution in Class with  
attribute  
to : GRLSoftgoal;  
from : GRLSoftgoal  
end

Individual Security in Class isA GRLSoftgoal  
with  
rule  
parameterizerule : \$ forall is2/Confidentiality n/GRLNonIntentionalElement (exists is1/Security ac/GRLAndContribution (ac to is1) and (ac from is2) and (is1  
obj n)) ==> (is2 obj n) \$ ;  
inheritrrule : \$ forall s/Security ac/GRLAndContribution (exists c/Confidentiality (acl from c)) ==> (acl to s) \$  
end

Individual Confidentiality in Class isA GRLSoftgoal  
end

Individual Individual isA Proposition  
end

11/4/01

CASCON 2001

# Tool Interfaces (6) : To *XML Funs*

## ■ XML /GXL format exports

```
<!ELEMENT goal-model (model-constructors)>
```

```
<!ATTLIST goal-model
```

```
    goal-model-id
```

```
    ID
```

```
    #REQUIRED
```

```
    goal-model-name
```

```
    CDATA
```

```
    #IMPLIED
```

```
    description
```

```
    CDATA
```

```
    #IMPLIED >
```

```
<!ELEMENT model-constructors (model-constructor)+ >
```

```
<!ELEMENT model-creator (actors?, intentional-elements, intentional-relationships) >
```

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE grl-spec (View Source for full doctype...)>
- <grl-spec>
+ <element-definitions>
- <goal-model>
  <goal-model goal-model-id="C:\ome3\projects\grl-samples\Z151-Fig7.xml" />
  - <model-constructors>
    <actors />
  - <intentional-elements>
    - <intentional-element>
      - <belief>
        <belief belief-id="Element_4" belief-name="Convergence of media reduces cost of ownership" />
      </belief>
    - <softgoal>
      <softgoal softgoal-id="Element_3" softgoal-name="Low Cost" />
    </softgoal>
    - <task>
      <task task-id="Element_2" task-name="Use VoiceLAN" />
    </task>
  + <task>
    ☐ <goal>
      <goal goal-id="Element_0" goal-name="Voice Be Transmitted" />
    </goal>
    </intentional-element>
  </intentional-elements>
+ <intentional-relationships>
</model-constructors>
</goal-model>
</grl-spec>
```



## Tool Interfaces (7) : To power users

### ■ ***Who is a “Power User”?***

- One who uses OME to build or experiment with new modelling frameworks
- One who develops plug-ins to extend OME’s functionality

### ■ ***How to Create a Framework File?***

### ■ ***How to Build a Plug-in?***



## *Create a Framework File*

- Framework file must use valid Telos **syntax**.
- Framework must be described in proper Telos **semantic**, for example, pay attention to the concept of metaclass, simple class, omega-class, token, etc.
- Specification of **types** occurs in **simple class** level, **instances** of these types in models are at **token** level.
- The specification of types must meet the expectations (**rules**) of OME.



# An element type declaration

```
SimpleClass MyFirstElement # (1)
  IN OMEElementClass, OMEInstantiableClass # (2,3)
  ISA OMEElement # (4)
  WITH name
    : "Element" # (5)
  defaultname # (6)
    : "An Element"
  imagename # (7)
    : "Graphic1.gif"
  imagesize # (8)
    width : 80;
    height : 50
  autogui : 1 # (9)
END
```

# A link type declaration

```
SimpleClass MyLink
  IN OMELinkClass, OMEInstantiableClass          # (10)
  ISA OMELink
  WITH
  attribute                                       # (11)
    to : OMEObject;
    from : MyFirstElement
  name
    : "Link"
  imagename                                       # (12)
    : "Arrow.gif"
  imagesize
    width : 20;
    height : 20
  stroke
    : "dashed"                                   # (13)
  autogui
    : 1
```

END

CASCON 2001



# Attributes

- All objects have a special attribute “**name**”.
- Objects can have arbitrary attributes, (e.g. domain, value, etc.). OME supports two kinds of attributes, “**value**” and “**reference**”.
- Value attributes can be of type **string** or **integer**, and they may have values.
- Value attributes must be specified under a special **attribute category** in the framework description.
- The tool has a “Value Attribute” interface that is provided to plug-ins, and implemented by **TelosValueAttribute**.

# Constraints

```
SimpleClass ValidChild
                                     # (14)
  ISA OMEElement
END
```

```
SimpleClass ChildA
  IN OMEElementClass, OMEInstantiableClass
  ISA OMEElement, ValidChild
                                     # (15)
  WITH
    name
      : "Child A"
    imagename
      : "Resource.gif"
    imagesize
      width : 80;
      height : 50
    autogui
      : 1
END
```

```
SimpleClass ChildB
  IN OMEElementClass, OMEInstantiableClass
  ISA OMEElement, ValidChild
                                     # (15)
  WITH
    name
      : "Child B"
    imagename
      : "Task.gif"
    imagesize
      width : 80;
      height : 50
    autogui
      : 1
END
```

```
SimpleClass MyExpandableElement
  IN OMEElementClass, OMEInstantiableClass
  ISA OMEElement, OMEGrowableElement
                                     # (16)
  WITH
    attribute
      children : ValidChild
                                     # (17)
    name
      : "Expandable"
    imagename
      : "Actor.gif"
    imagesize
      width : 80;
      height : 80
    autogui
      : 1
END
```



# OME-plug-in interface

- ❏ Specify *when* (under *what condition*) the plug-in should be loaded (via the definition of *isCompatibleWith()*);
  - ❏ *Build the methods* current plug-in should include(classes inherited *PluginMethods*);
  - ❏ Specify *where* the methods will appear in the GUI, possible alternatives are: *Menubar*, *toolbar*, and *popup menu*.
- 
- *In OME, model manager is responsible to establish and manage mappings between models and plug-ins. The operations include: *initializePlugins()*, *getAllPlugins()*, and *getPlugins()*.*
  - *The View asks the ModelManager for the plug-ins corresponding to the model of current view.*

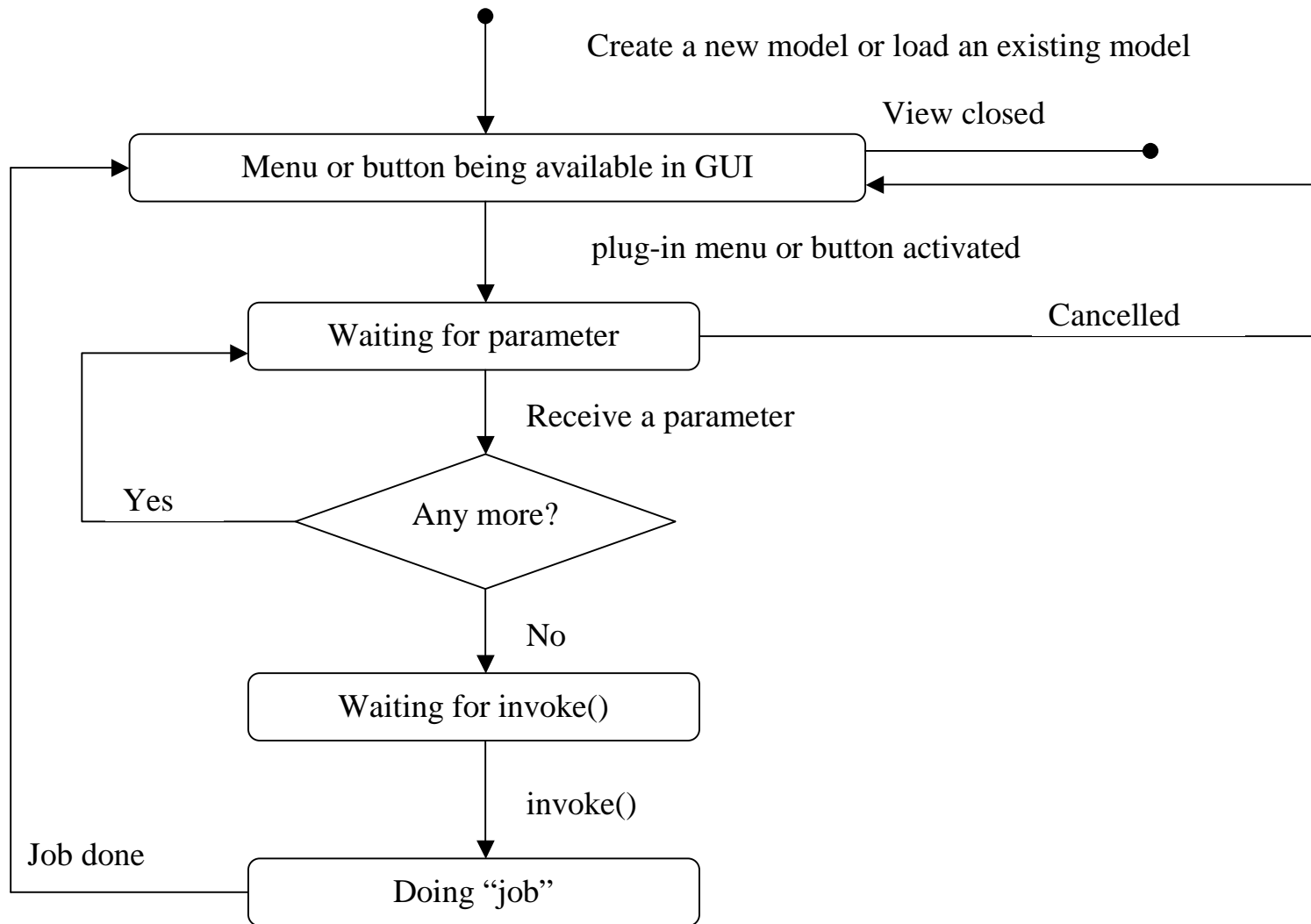


## *Build a plug-in method*

### ■ ***What is a plug-in Method?***

- A plug-in method can be seen as an agent providing certain kinds of services to the View.
- A plug-in method is reactive agent, who simply waiting for requests (or signals) from the View.
- A plug-in method is made available in GUI at model loading/creating time.

# The Lifecycle of a Plug-in Method



11/4/01

Figure3: The life cycle of a plug-in method agent

```

/** This is a toy example of plug-in for learning purposes. */
import java.util.Collection;
import java.util.LinkedList;
import java.awt.Image;
import OME.*;

public class ToyPlugin implements OMEPlugin {

    OMEModel model;
    LinkedList popupmethods;

    public ToyPlugin(OMEModel model) {
        this.model = model;
        popupmethods = new LinkedList();
        popupmethods.add(new ToyMethod());
    }

    public static boolean isCompatibleWith(OMEModel model) {
        return true;
    }

    public Collection getMethods(View v) {
        return null;
    }

    public Collection getToolbarMethods(View v) {
        return null;
    }

    public Collection getMenuBarMethods(View v) {
        return null;
    }

    public Collection getPopupMethods(View v) {
        11/4/01 return popupmethods;
    }

```

```

// Methods for this plug-in

private class ToyMethod implements PluginMethod {
    public void invoke() {
        D.o("Hello World, I'm the ToyPlugin's first method!");
    }

    public String getName() {
        return "Toy Method";
    }

    public Image getImage() {
        return null;
    }

    public Collection getSubmenu(ViewContext ovc) {
        return null;
    }

    public Collection getSubmenu() {
        return null;
    }

    public PluginParameter nextParameter() {
        return null;
    }

    public void passParameter(Collection param) {
    }

    public void cancelled() {
    }

    public boolean isEnabled(ViewContext con) {
        ViewObject vo = con.associatedObject();

        if (vo instanceof ViewLink) {
            return true;
        }

        return false;
    }
}

```

CASCON 2001

# OME3 API

The screenshot shows a Netscape browser window titled "OME 3 Documentation - Netscape". The address bar shows the file path: `file:///C:/lin/liu/Doc/ome3_documents/Dme_API/ome_public_doc/index.html`. The main content area displays the documentation for the **Class AbstractPluginMethod**. The left sidebar lists "All Classes" including [AbstractPluginMethod](#), [CreateElementMethod](#), [CreateLinkMethod](#), [MenuMethod](#), [ModelAttribute](#), [ModelElement](#), [ModelLink](#), [ModelObject](#), [ModelValueAttribute](#), [ObjectMethod](#), [OMEElement](#), [OMEFramework](#), [OMELink](#), [OMEModel](#), [OMEObject](#), [OMEPlugin](#), [OMEType](#), [PluginMethod](#), [PluginParameter](#), [PopupMenuSeparatorMethod](#), [View](#), [View.GraphicLocation](#), [View.Location](#), [ViewContext](#), [ViewElement](#), [ViewLink](#), and [ViewObject](#).

The main content area has a navigation bar with links: [Class](#), [Tree](#), [Deprecated](#), [Index](#), and [Help](#). Below this, there are links for [PREV CLASS](#), [NEXT CLASS](#), [FRAMES](#), and [NO FRAMES](#). The summary section includes [INNER](#), [FIELD](#), [CONSTR](#), and [METHOD](#). The detail section includes [FIELD](#), [CONSTR](#), and [METHOD](#).

The class name **OME Class AbstractPluginMethod** is displayed. Below it, the inheritance hierarchy is shown: `java.lang.Object` | `+--OME.AbstractPluginMethod`.

**Direct Known Subclasses:**  
[CreateElementMethod](#), [CreateLinkMethod](#), [MenuMethod](#), [ObjectMethod](#), [PopupMenuSeparatorMethod](#)

The class definition is shown as: `public abstract class AbstractPluginMethod extends java.lang.Object implements PluginMethod`.

A description follows: "This class is an adapter class that implements all the methods in the `PluginMethod` interface as no-op's and that also includes a list of the supported parameter types as static fields."

Another description: "This class exists as a convenience so that classes wishing to implement the `PluginMethod` interface can instead extend this class and override the methods of interest. (If one implements the `PluginMethod` interface, one has to define all of the methods in it. This adapter class defines no-op methods for them all, so that one need only define the methods one cares about)."

# OME3 Architecture

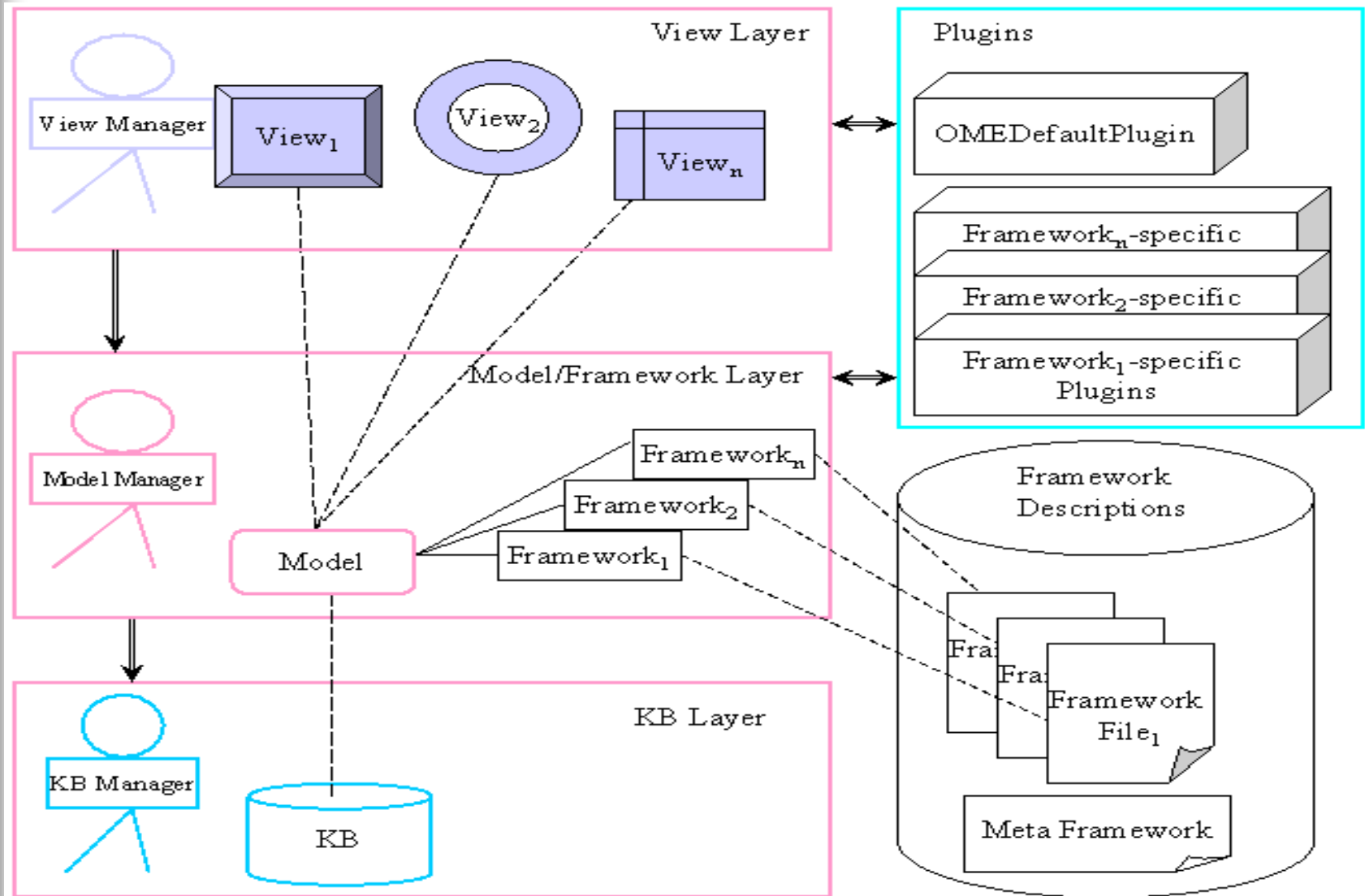
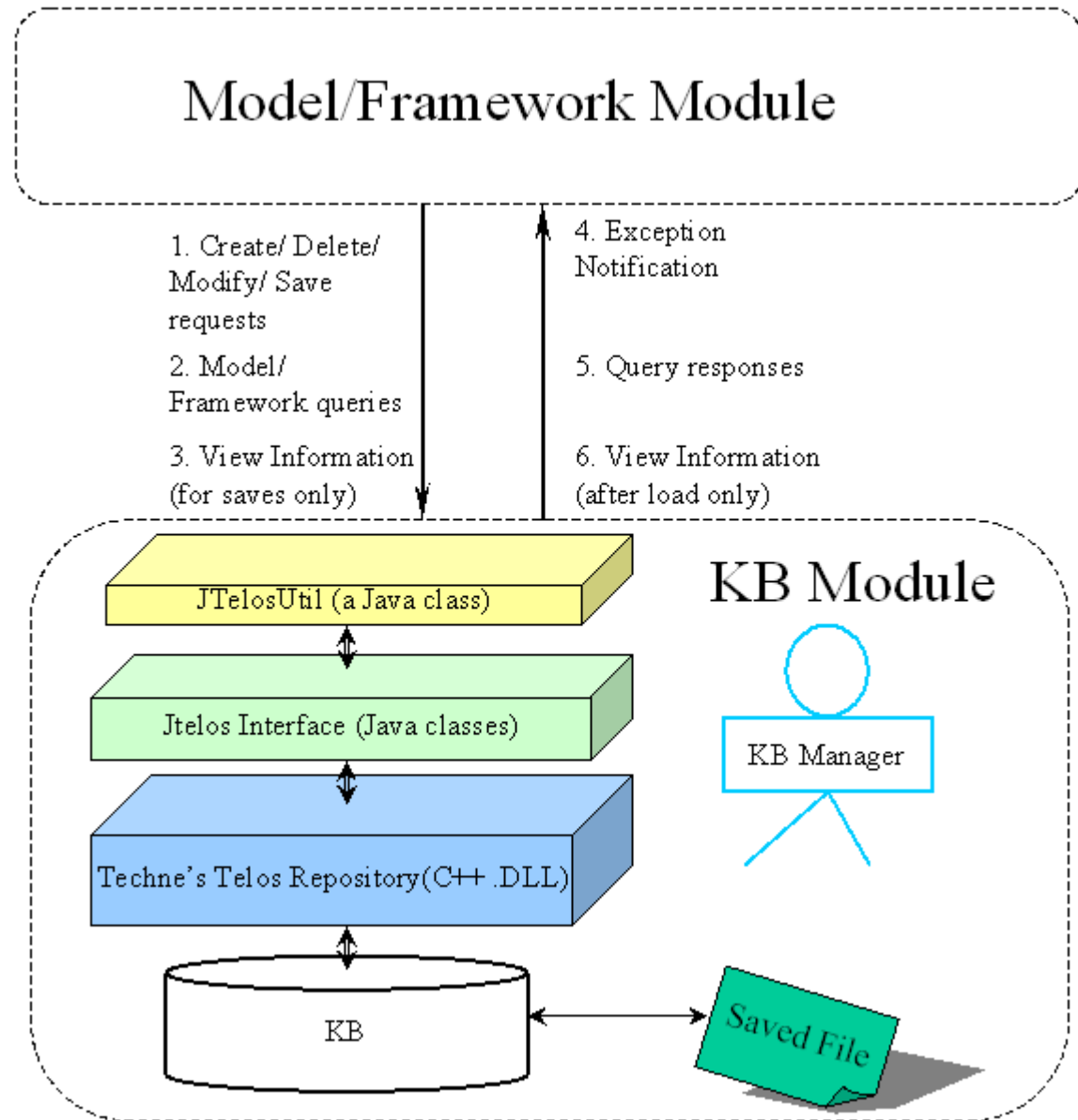


Figure 1: The Architecture Overview

# The KB



11/4/01

CASCON 2001  
Figure 2. The KB module



# The model/framework

## ■ Functions of model/framework module

- provides a simpler, more abstract interface to the information stored in the KB.
- insulates the rest of the tool from a potential re-implementation of the KB.
- conceptually separates the model from its framework.

## ■ Functions of model manager

- communicates with: KB manager, user, view manager, to create new models, load and save existing models.
- Decides which plug-ins to include, and provides an an interface to plug-ins for view module.



# The View

## ■ Functions of the view module:

- presents representations of the model to the user.
- gives the user access to modify the model and/or its representation.
- provides the user with access to the functionality provided by plug-ins.
- provides the plug-ins a means to receive input from the user.
- makes changes to models by changing the copy of the model kept by the View.
- passes on the changes to deeper layers.

*Each model can have several different kinds of views. These views give various presentations of the same model.*



# The plug-ins

- plug-ins are a set of modules that extend the functionality of OME3.
  - Each plug-in must implement the `OMEPlugin.java` interface.
- A plug-in is a collection of methods.
  - Each method must implement the `PluginMethod.java` interface.
  - These methods can analyze and/or alter the view of the model, or the underlying model itself.
- plug-ins are loaded dynamically, depending on the model and framework(s) in use.
  - Only the plug-ins associated with the frameworks used by the current model will be loaded.
  - This architecture allows framework specific functionality to be implemented in a plug-in.
  - There is a special plug-in, `OMEDefaultPlugin`, which is always activated no matter which model is loaded.

# Plug-ins

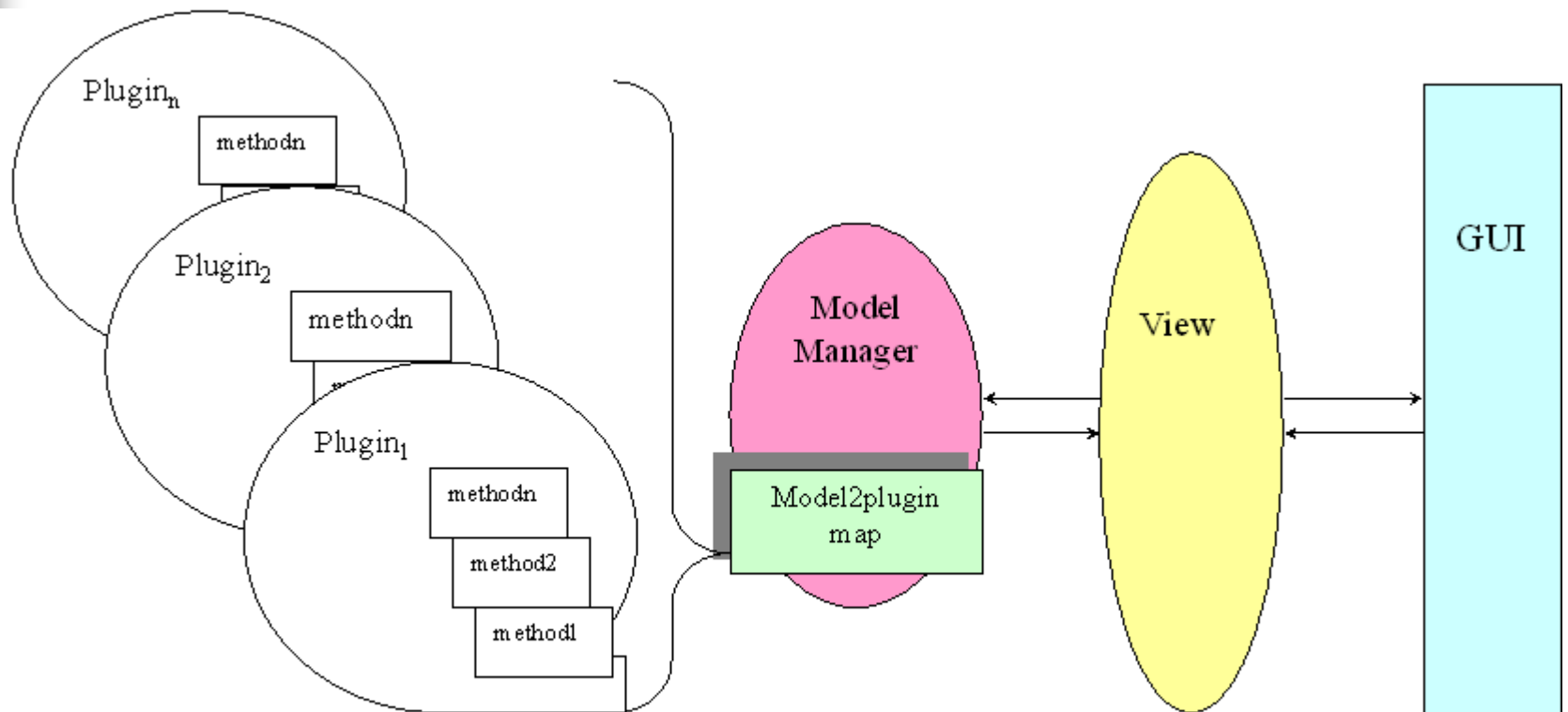


Figure 1: Model Manager controls the map between models and plugins



# Framework Description Files

## ■ Meta framework(ome.tel)

- includes following base types that all types must inherit from:

### □ *(Non-expandable) Elements*

- Nodes in a model graphic representation

### □ *Expandable Elements*

- Objects being able to include other (non-expandable) objects inside their boundary
- Objects with internal states

### □ *Links*

- A connection between two other objects (can either be an elements or a link) in the model
- Arcs (possibly directed) in a model graphic representation

## ■ Frameworks ( istar.tel & nfr.tel)

- A framework specifies the types of objects that can exist in a model employing the framework.
- A framework also specifies the relationships that can exists between these objects in the model.
- In OME3, these object types and relationships are specified in a textual Telos file (.tel).

11/4/01 In order to be usable, the types must branch off the *OME Meta Framework* (ome.tel).  
OASIS CONF 2004



# OME new features

- Reasoning Mechanisms of GRL, i\* & NFR
- Mechanisms for integration & validation:
  - export models into XML format (for integration with UCM, UML, doors, ...)
  - export models into O-telos format (for validation with ConceptBase tool)
- User Interface Improvement



# Reasoning Mechanisms of GRL, i\* and NFR

- Implement **know-how KB** and a knowledge accumulation/update mechanism, to
  - Provide general guidance for modelling procedure as well as reasoning procedure;
  - Abstract and accumulate *domain knowledge* (shown as domain catalogues) and *application specific knowledge*;
  - Actively acquire knowledge by querying developers.

*Typical know-how knowledge includes:*

- *how to combine the contributions of low level nodes;*
- *what kind of NFR structures may lead to the always denied/satisfied softgoals;*
- *which nodes are the inflection nodes, and which are the nodes makes difference when changes occurs*
- *how to detect “conflict-of-interests”, “common-interests”, “loops-of-dependencies”, “attack-defense contending”, “best-configurations”,.....*
- *how to evaluate current circumstance and alternatives on achieving/satisficing internal goals and softgoals.*



# Future Works

- More Reasoning Mechanisms
- Automated Plug-in and Framework generation Tool
- Web-based interface
- Tighter Integration with other tools: UCMNav, Rational Rose, etc.



# Automated Plug-in and Framework generation Tool

- Provide more friendly tool support for power users:
  - build **interactive** GUIs
    - to create new frameworks based on **template** (OME Meta framework ome.tel);
    - to program new plug-ins based on template (PluginMethods.java, OMEPlugin.java), similar to **Visual studio**.



**Thank you  
&  
Discussions**

11/4/01

CASCON 2001



# Related Links

- OME tool:

<http://www.cs.toronto.edu/km/ome/>

- Publications:

<http://www.cs.toronto.edu/~eric/>

- GRL Modelling Language:

<http://www.cs.toronto.edu/km/GRL/>