ENGINEERING
INGEGNERIA
INFORMATICA

Design and Development of Intentional Systems with PRACTIONIST *Studio*

Angelo Marguglio, Giuseppe Cammarata,
Susanna Bonura, Giuseppe Francaviglia,
Michele Puccio and Vito Morreale.

R&D Lab
ENGINEERING Ingegneria Informatica S.p.A.

WOA 2008
dagli Oggetti agli Agenti
Evoluzione dell'agent development:
metodologie, tool, piattaforme e linguaggi

Palermo 17-18 Novembre 2008

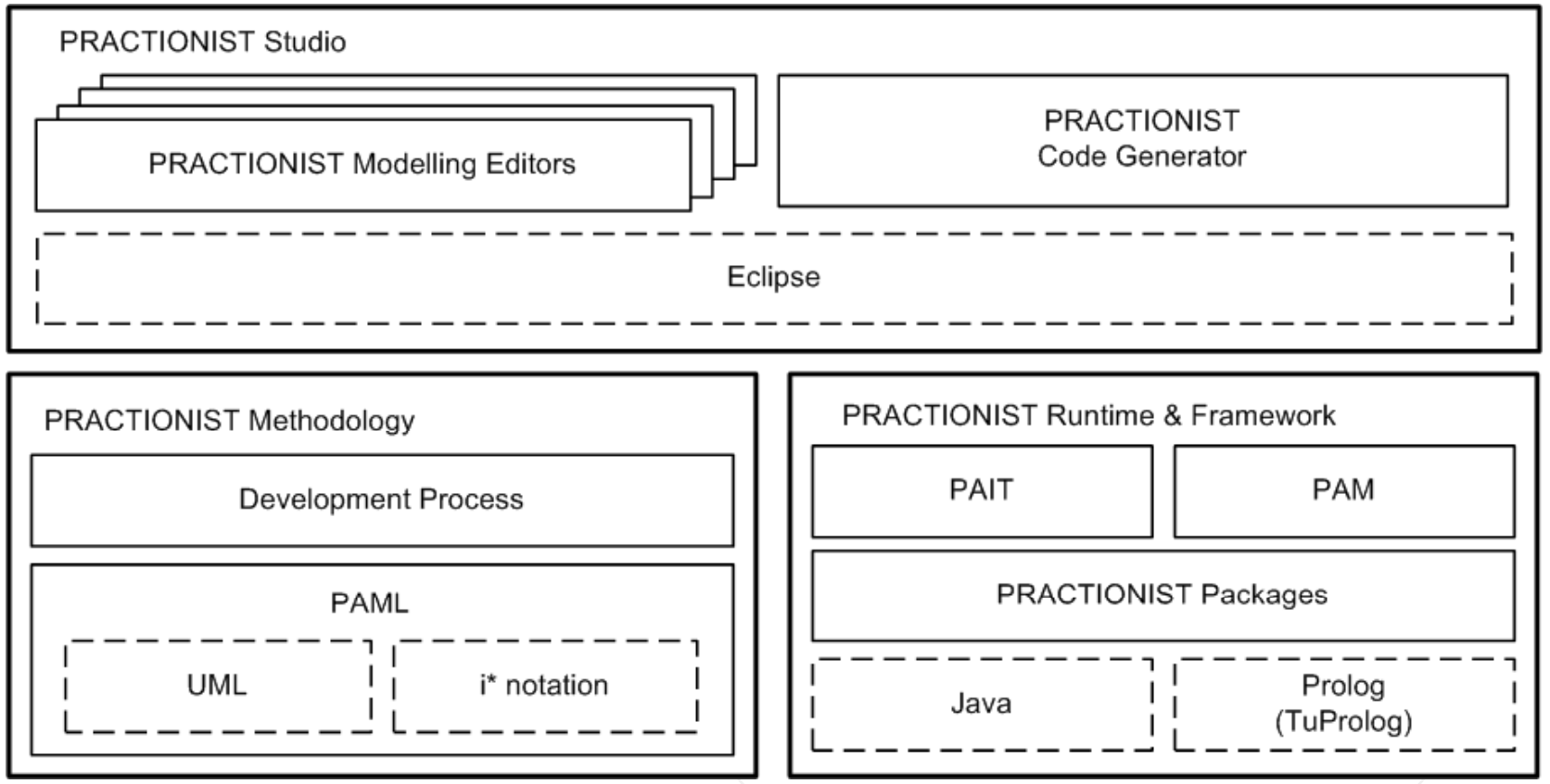
Outline

- Motivations
- PRACTIONSIT Suite
- PRACTIONIST Agent Modelling Language (PAML)
- PRACTIONSIT *Studio* (PS)

Motivations

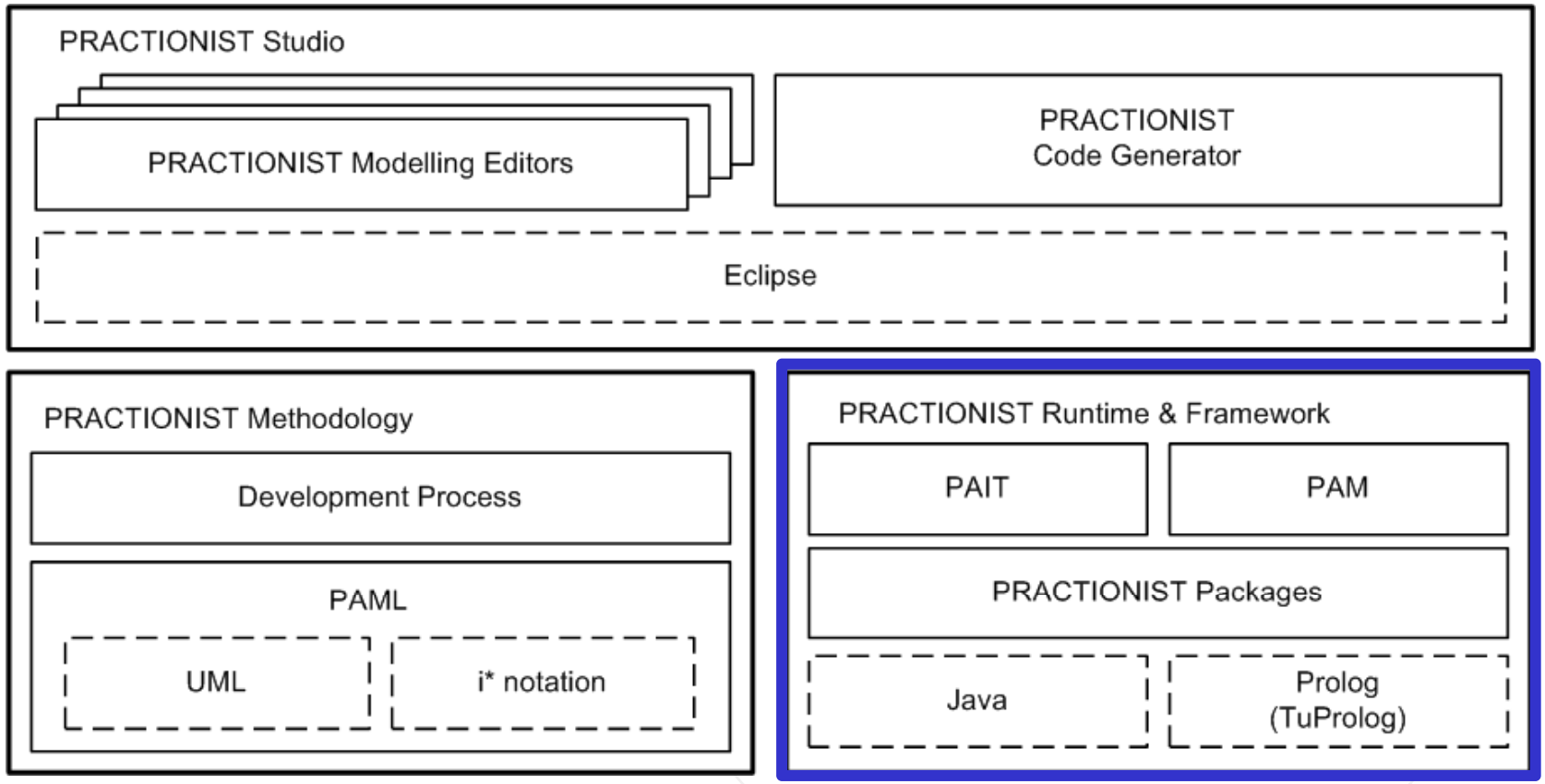
- Based upon the **BDI** model
- Stress aspects such as **mental states** like beliefs, desires and intentions
- **Lack of industry ready tools** for design and development of MAS
  - Strong tie-up with specific methodologies
  - Cover only a subset of development phases
  - Simple prototypes
  - Very limited assistance

PRACTIONIST Suite



PRACTIONIST

PRACTIONIST Suite



PRACTIONIST

PRACTIcal reasONIng sySTem

Built-in services

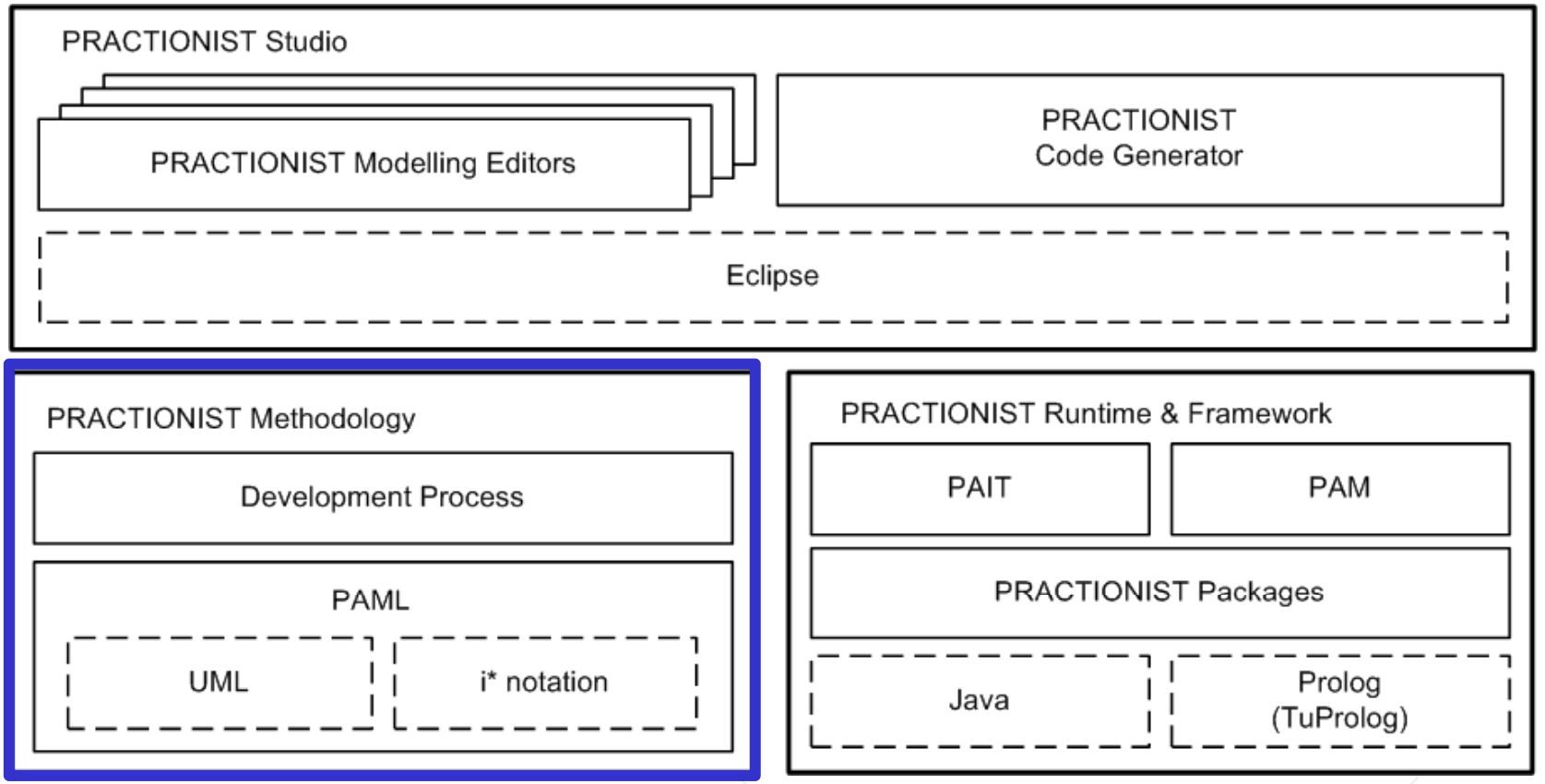
- Belief logic
- Deliberation mechanisms
- Means-ends reasoning
- Planning
- Plan executions

Main features

- Goal model
- Plan library
- Perceptors
- Effectors
- Belief base

PRACTIONIST

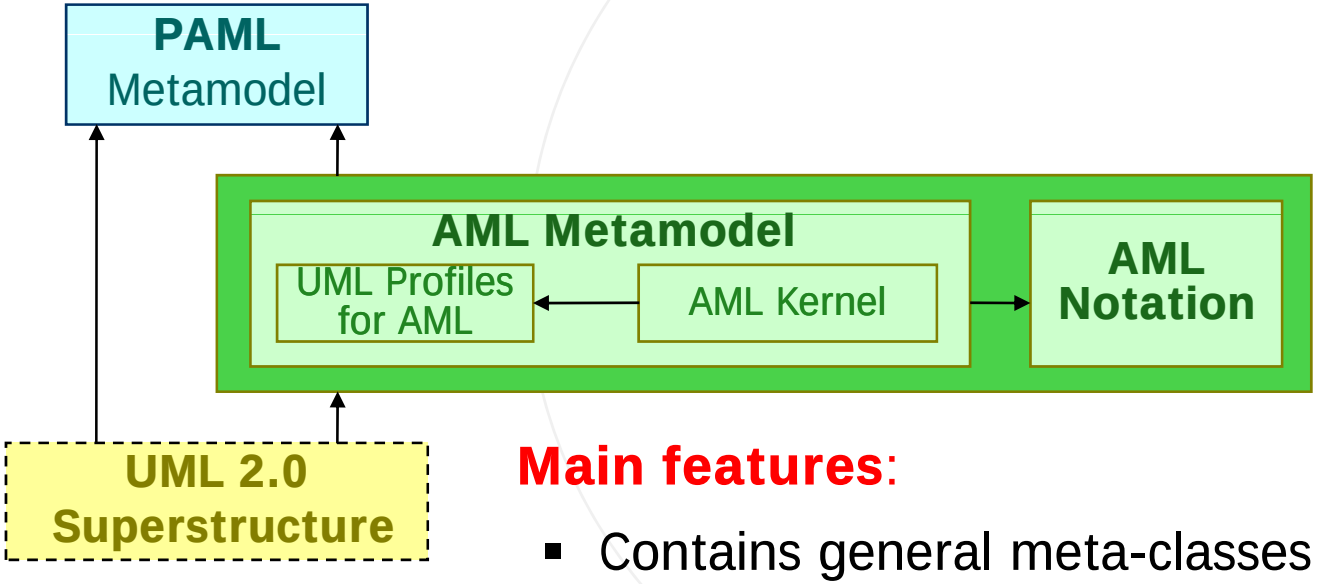
PRACTIONIST Suite



PRACTIONIST

PAML

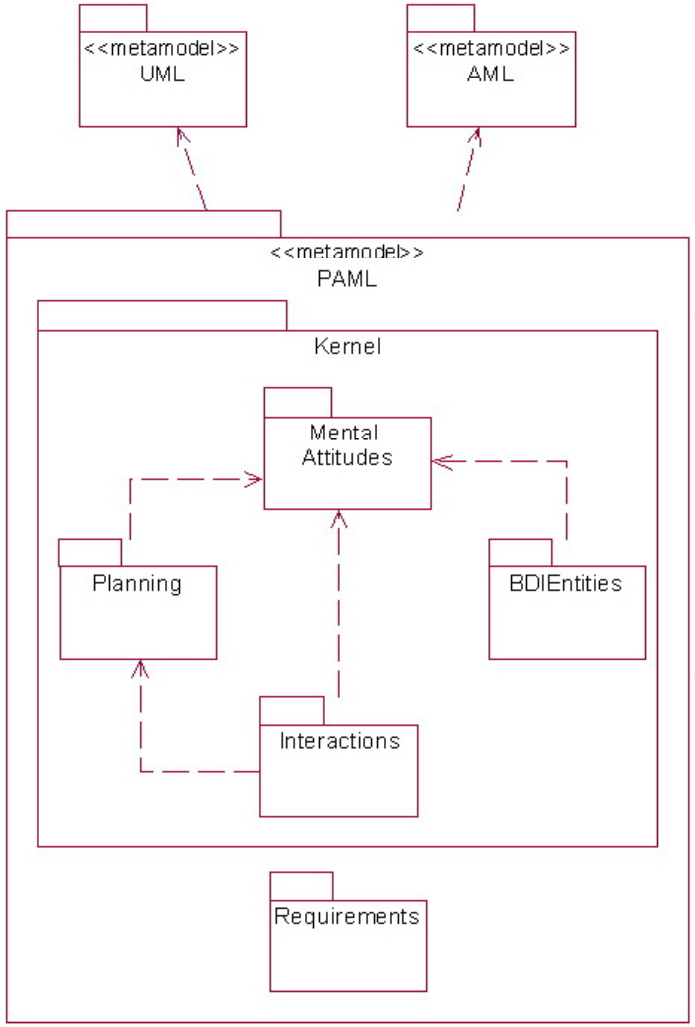
Semi-formal visual modelling **language** for specifying, modelling and documenting BDI multi-agent systems, designed using the development methodology defined within the PRACTIcal reasONIng sySTem (PRACTIONIST)



Main features:

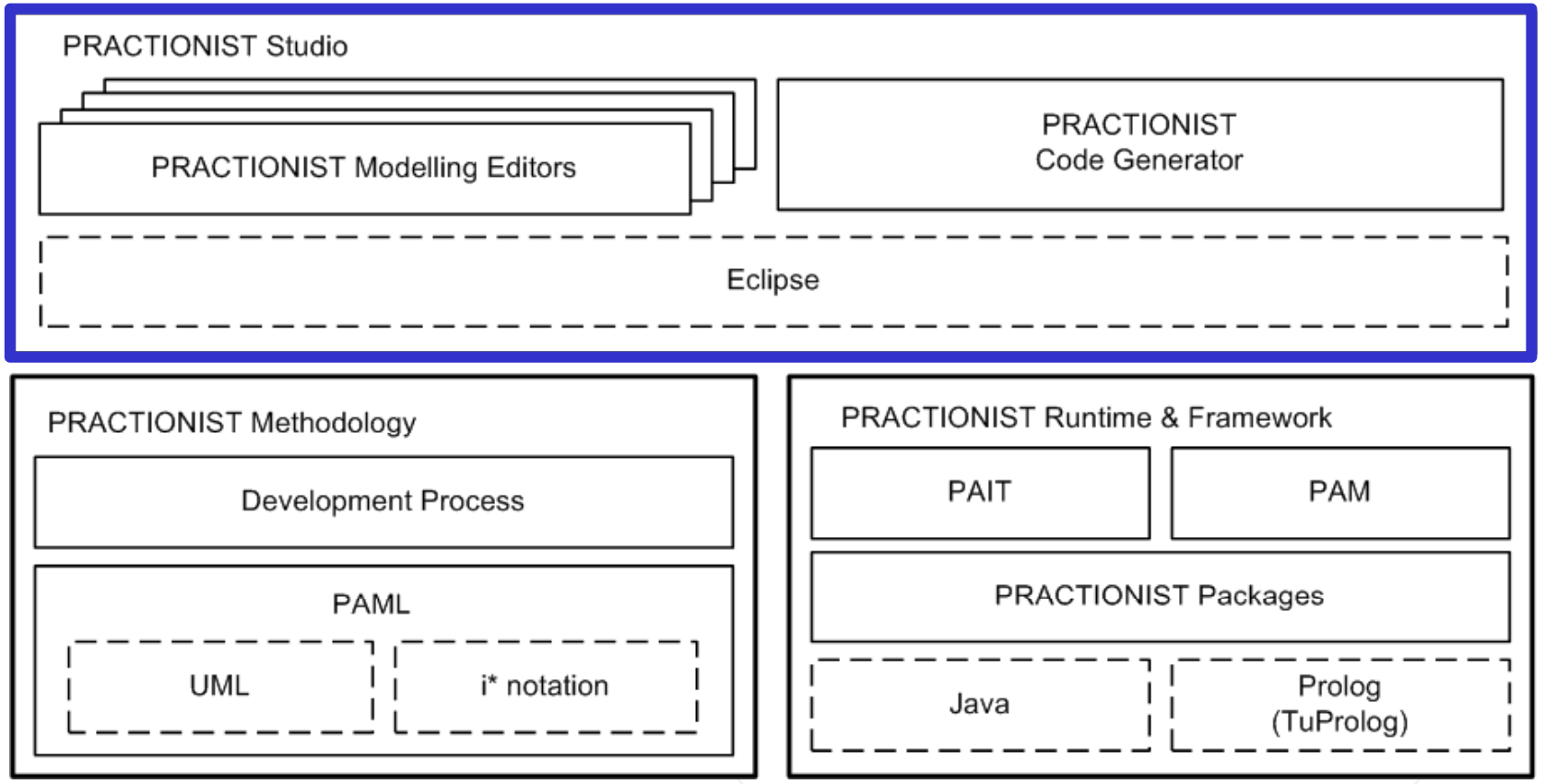
- Contains general meta-classes to model intentional components of **BDI agents**
- Includes meta-classes specific to **PRACTIONIST-based** systems

PAML - Packages



- `Kernel` package (artifacts, agents and their components)
- **`Mental Attitudes`** package (intentional attitudes such as beliefs, goals and plans)
- `BDIEntities` package (artifacts and agents)
- `Interactions` package (perceptors, perceptions, effectors and actions)
- `Planning` package (plan body activities)
- `Requirements` package (i* notation)

PRACTIONIST Suite



PRACTIONIST

PRACTIONIST *Studio* (PS)

- Main goals
  - Support (*i*) the development **methodology** of the framework, (*ii*) and the graphic modelling of the main **abstractions** characterizing the PRACTIONIST agent
  - Support the realization of agent-based applications according to PRACTIONIST model, **from design to the code generation**
- Implementation
  - Developed in **Java** (cross-platform)
  - Built using the **Eclipse** support
  - Implementation of the **PAML**
- Release-build policies
  - **Open source**
  - Eclipse plugins or standalone RPC

PRACTIONIST

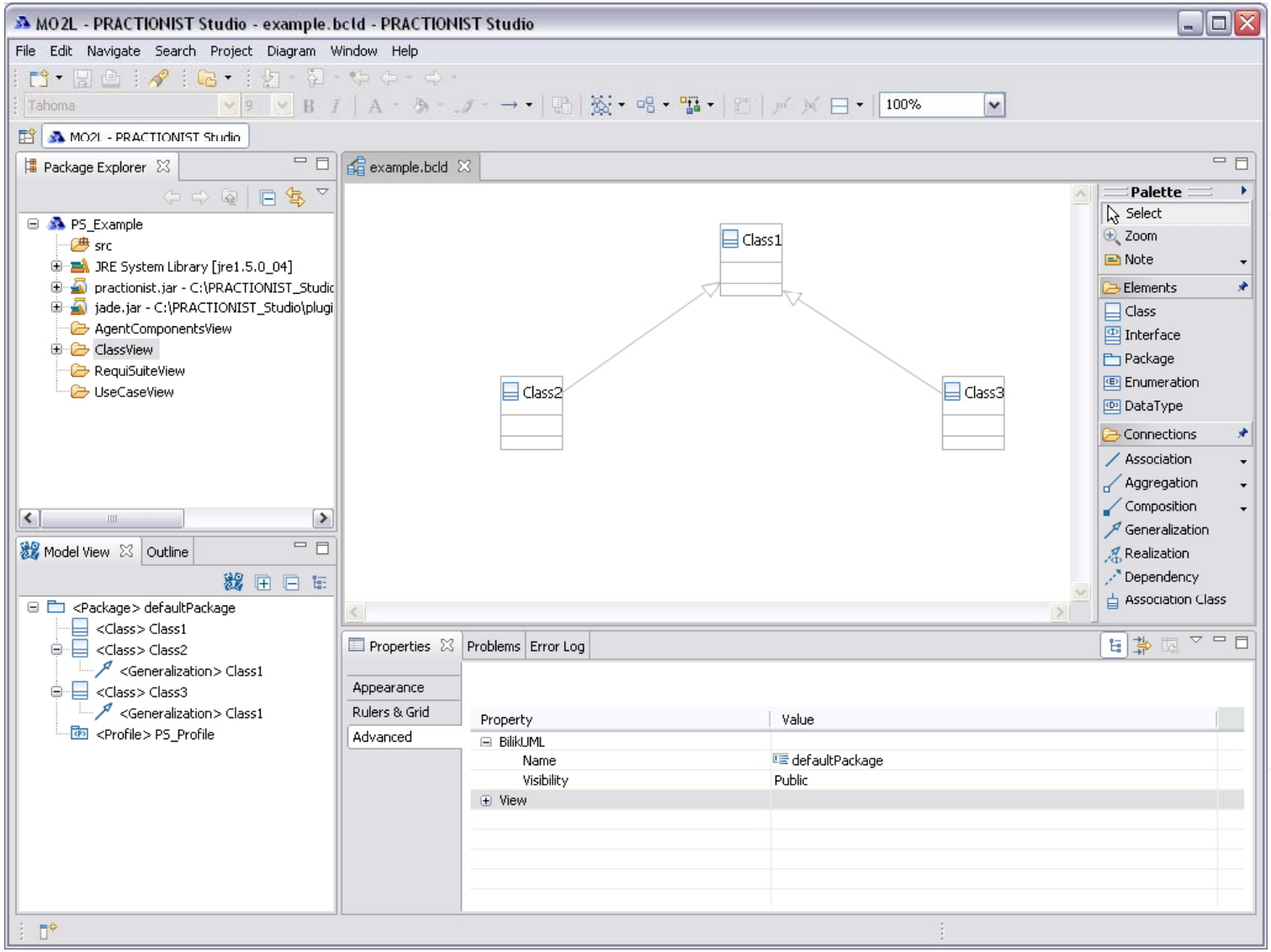
PS – Developed editors

- **UML2** based
  - *Class editor*
  - *Use Case editor*
- **i*** based
  - *Strategic Dependency (SD) editor*
  - *Strategic Rationale (SR) editor*
- **PRACTIONIST** specific
  - *Goal editor*
  - *Domain editor*
  - *Plan editor*
  - *Plan Body editor*
  - *Effector/Action – Perceptor/Perception editor*
  - *Agent editor*

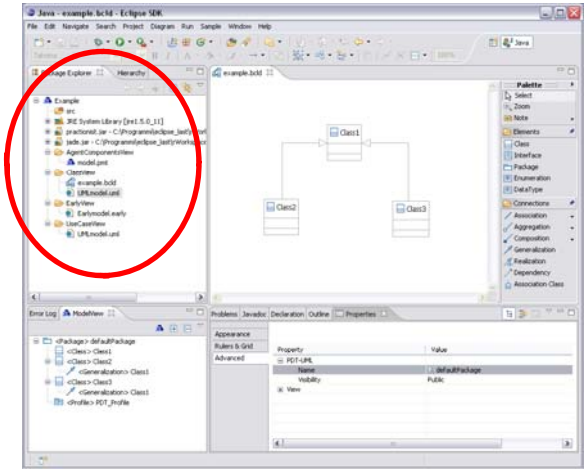
PS – Editors Common Infrastructure

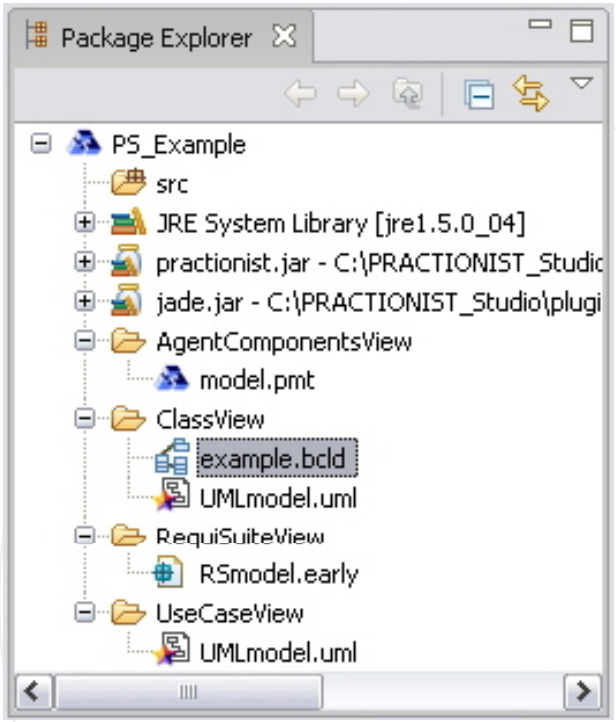
- **Integrated** development using the PS-Project
- Common editing facilities
- **Unified** underlying model
- Model View
- Entity reuse in several views of the system
- **Validation** check
- Automatic **code generation** facility

PS – Main GUI


MO2L - PRACTIONIST Studio - example.bcld - PRACTIONIST Studio
File Edit Navigate Search Project Diagram Window Help
Package Explorer
PS_Example
src
JRE System Library [jre1.5.0_04]
practionist.jar - C:\PRACTIONIST_Studio
jade.jar - C:\PRACTIONIST_Studio\plugi
AgentComponentsView
ClassView
RequiSuiteView
UseCaseView
example.bcld
Class1
Class2
Class3
Palette
Select
Zoom
Note
Elements
Class
Interface
Package
Enumeration
DataType
Connections
Association
Aggregation
Composition
Generalization
Realization
Dependency
Association Class
Model View
Outline
<Package> defaultPackage
<Class> Class1
<Class> Class2
<Generalization> Class1
<Class> Class3
<Generalization> Class1
<Profile> PS_Profile
Properties
Problems
Error Log
Appearance
Rulers & Grid
Advanced
Property
Value
BilikUML
Name
defaultPackage
Visibility
Public
View

PS – Project

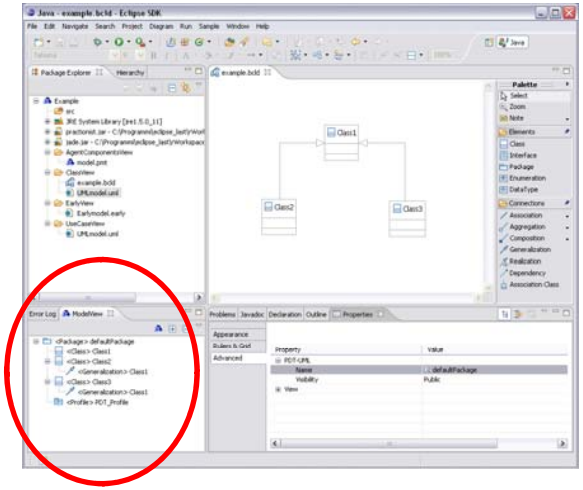


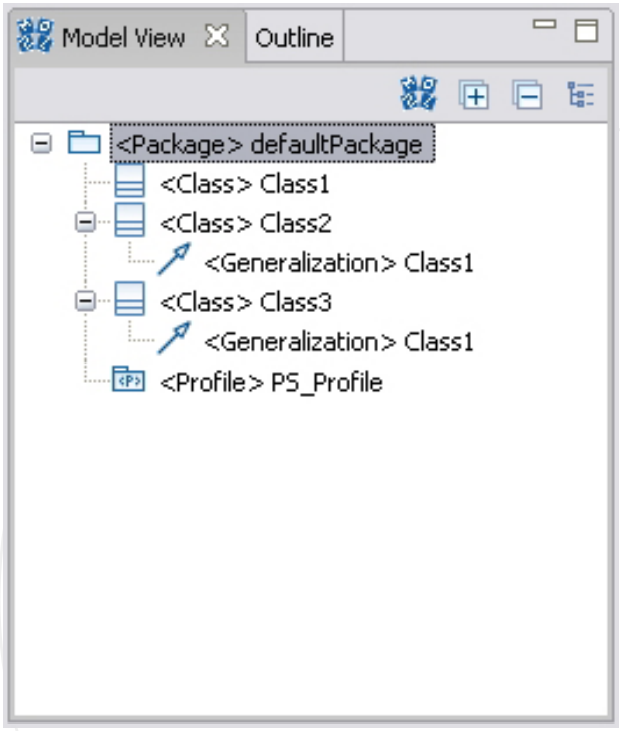


- **Custom Eclipse Project** supporting the right organization of diagrams and source code
- **Automatic creation** of the sections regarding the different phases of the methodology
- **src** folder containing the generated source code of the agents involved in the designed system

PRACTIONIST

PS – Model View



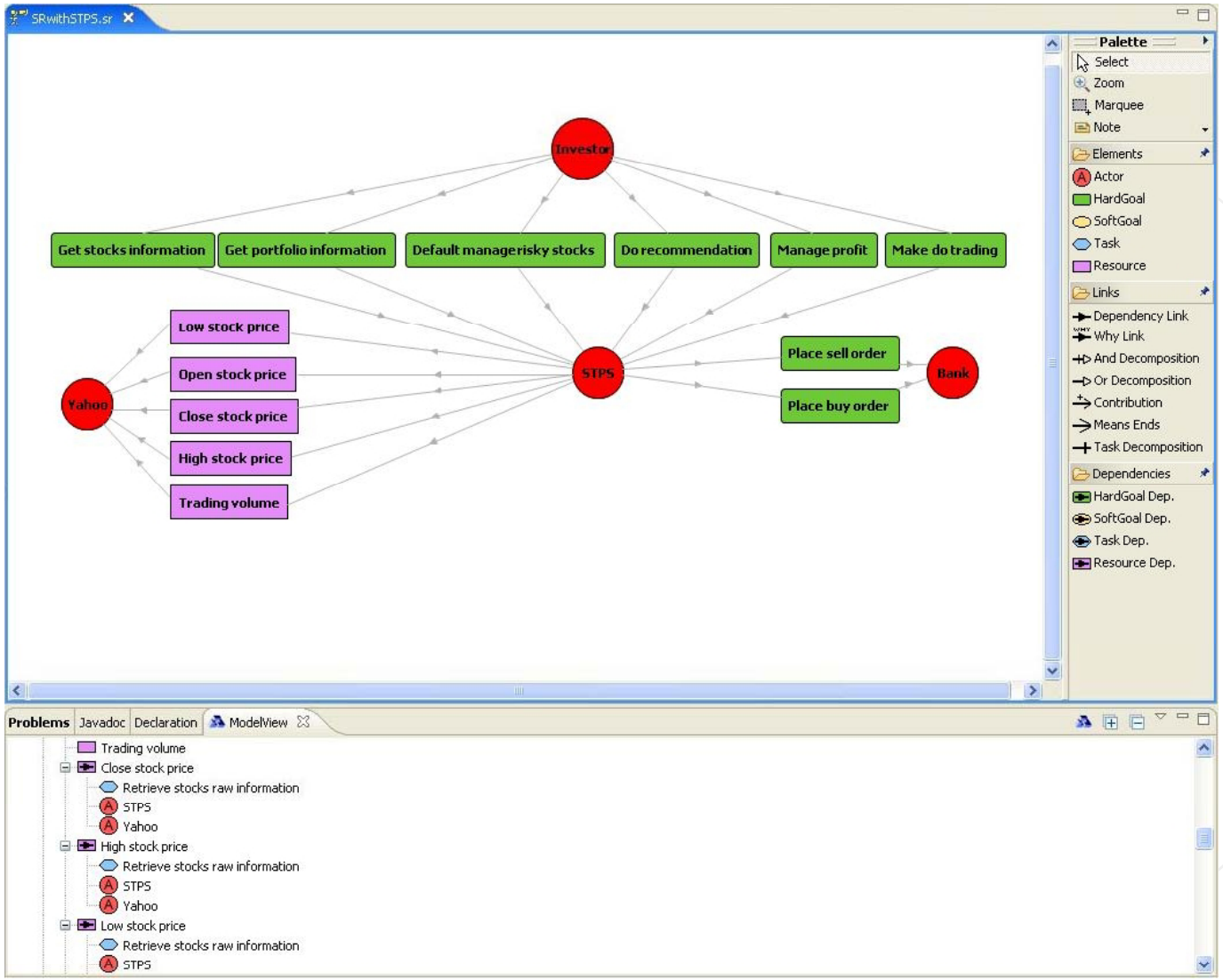


- **Centralized view for the model** underlying a structured phase of the methodology
- Used to start the **drag and drop** feature
- Used to start the "**Delete from View**" e "**Delete from Model**" actions

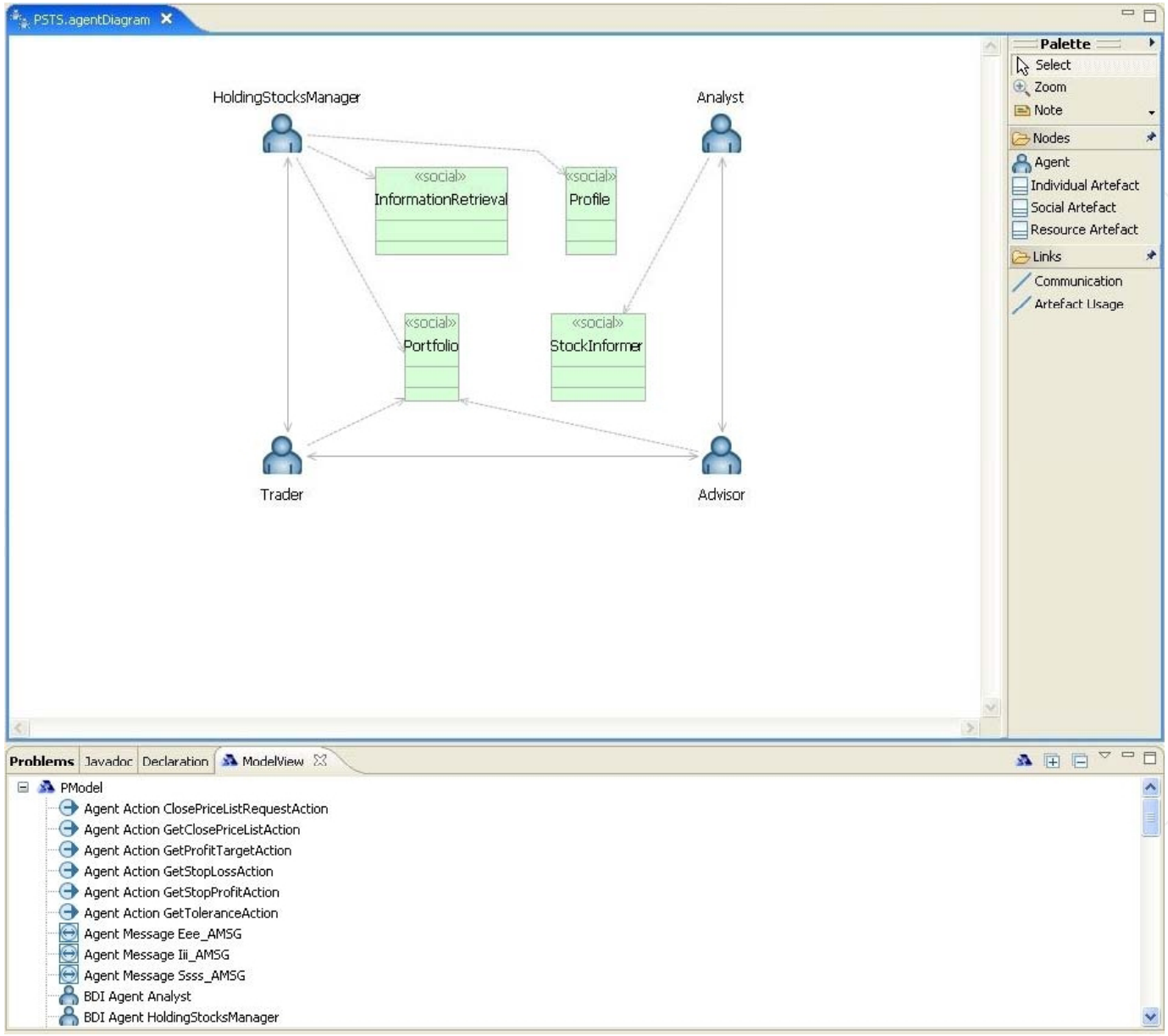
Case study: PSTS

- PRACTIONIST Stock Trading System
- System (with a high complexity) supporting stock markets' operations and decisions
- Other existing agent-based systems in critical fields such as financial and stock trading

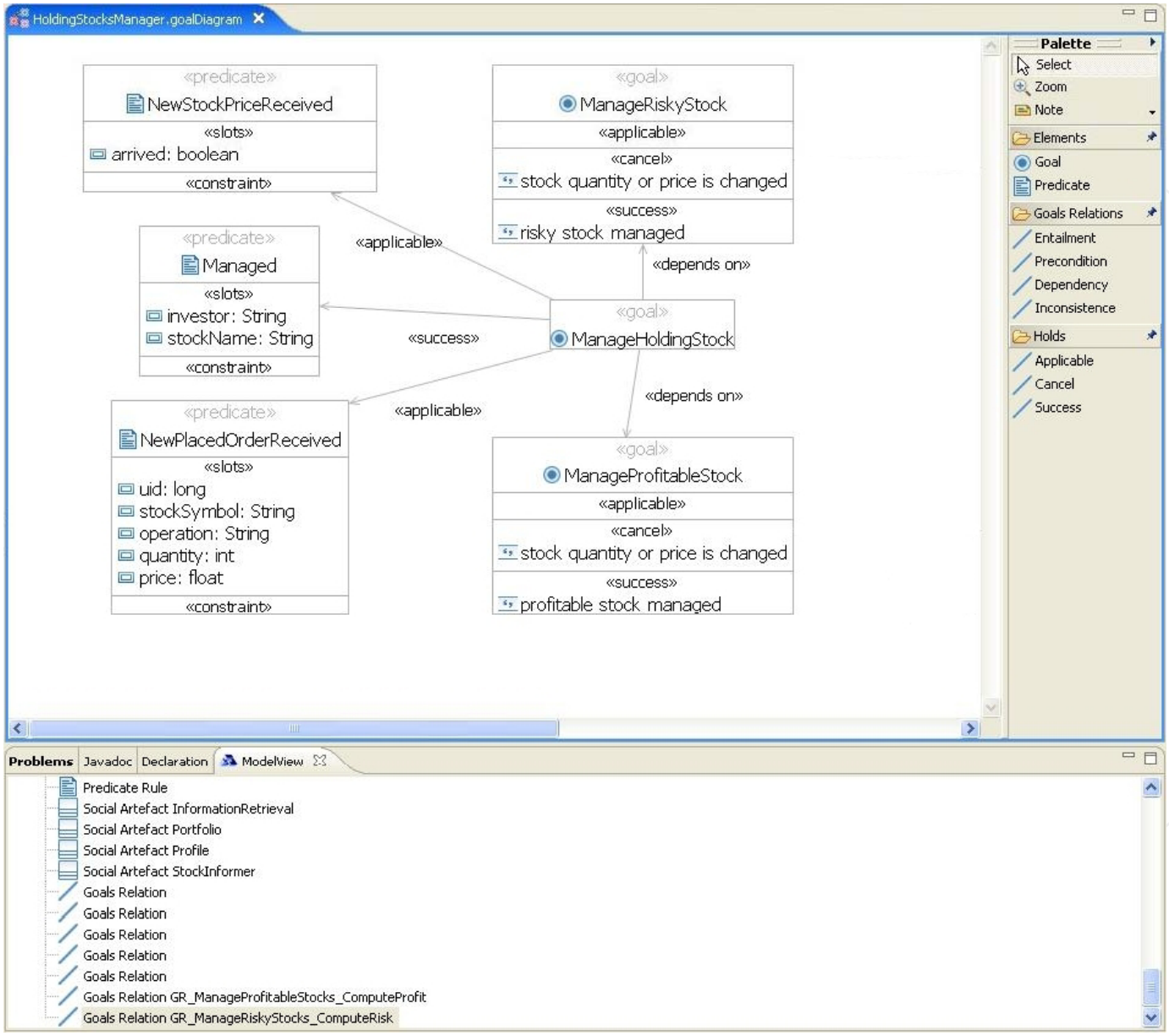
PS - Strategic Dependency Editor



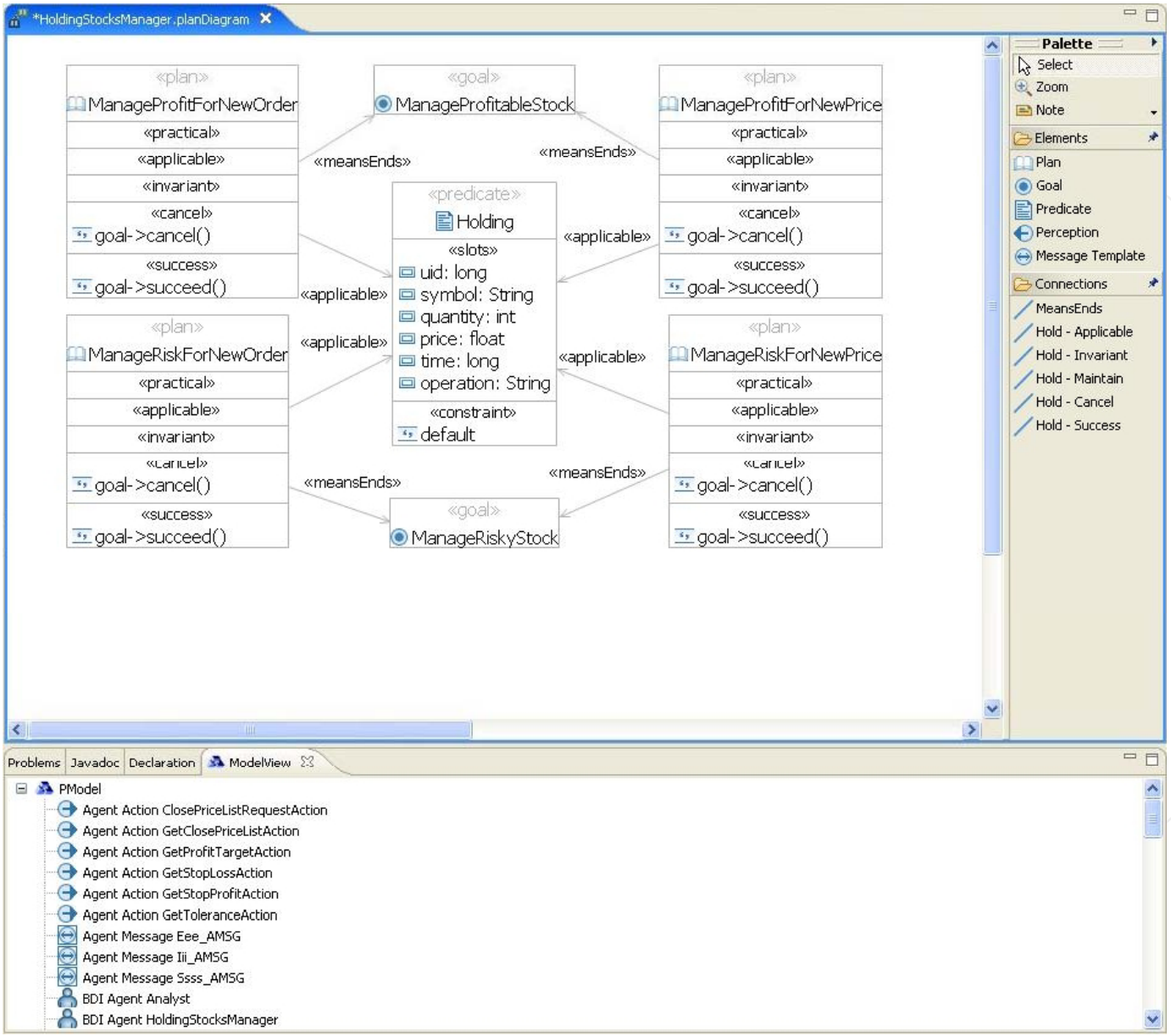
PS - Agent Editor



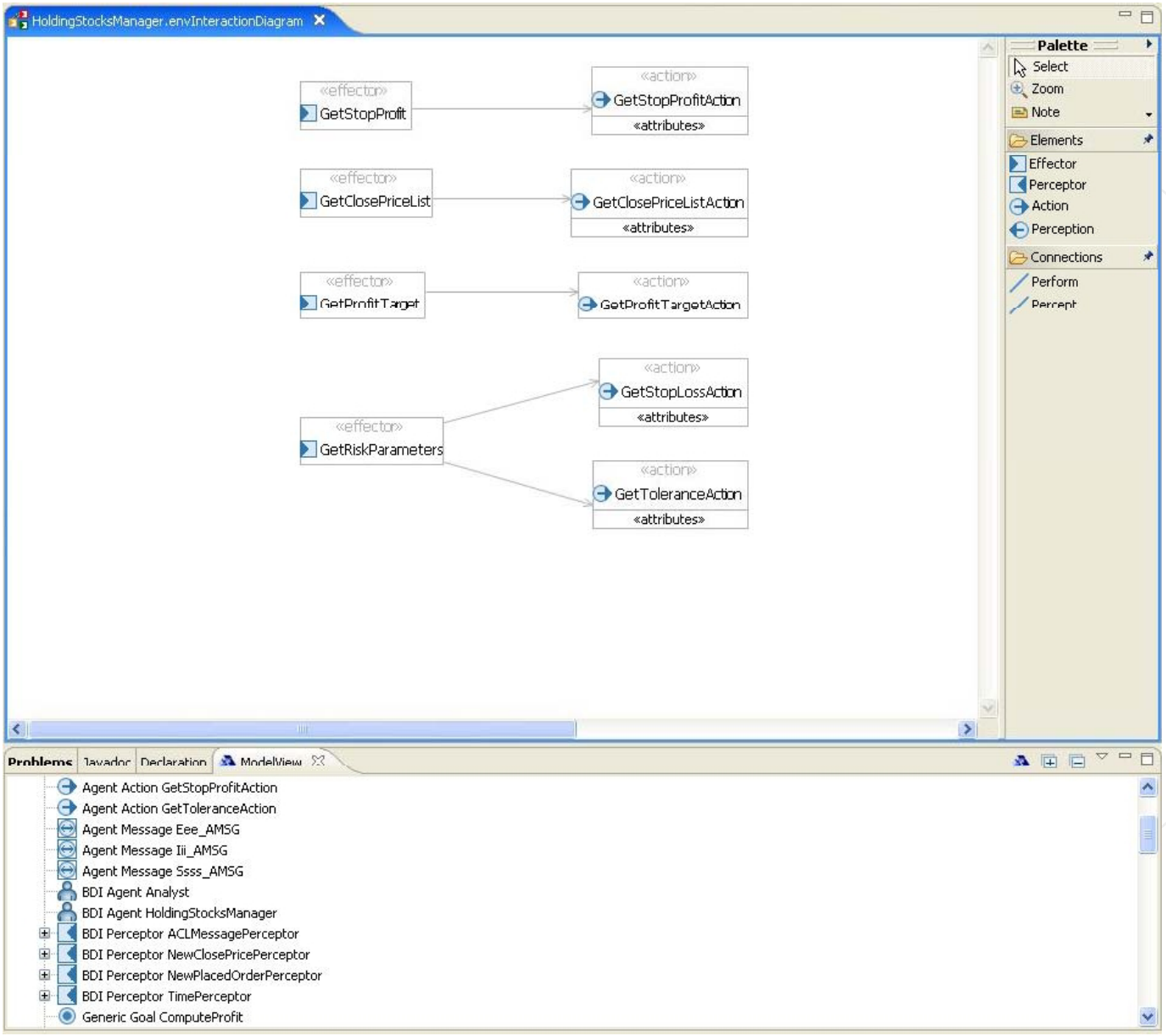
PS - Goal Editor



PS - Plan editor

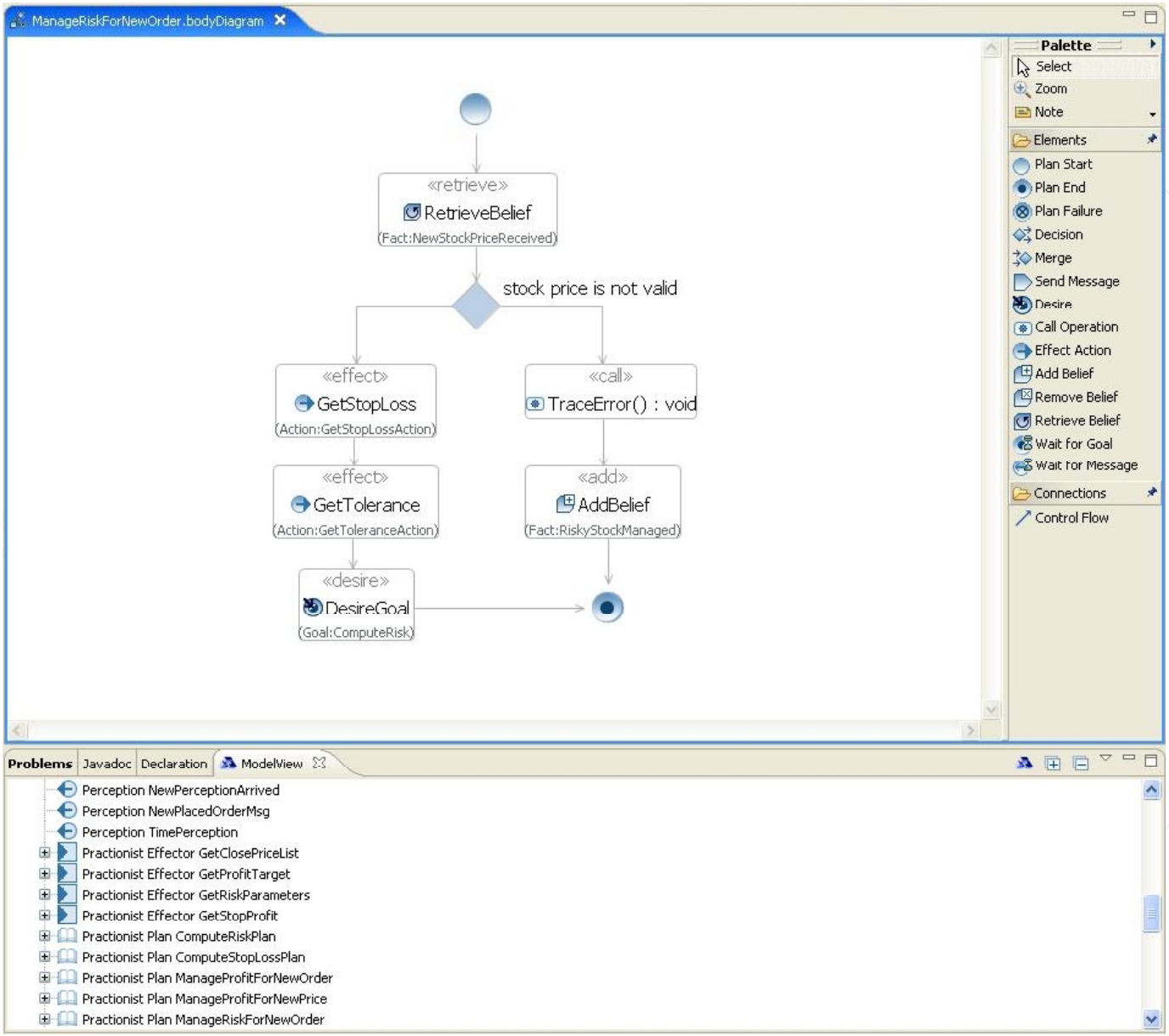


PS - Effector / Perceptor Editor

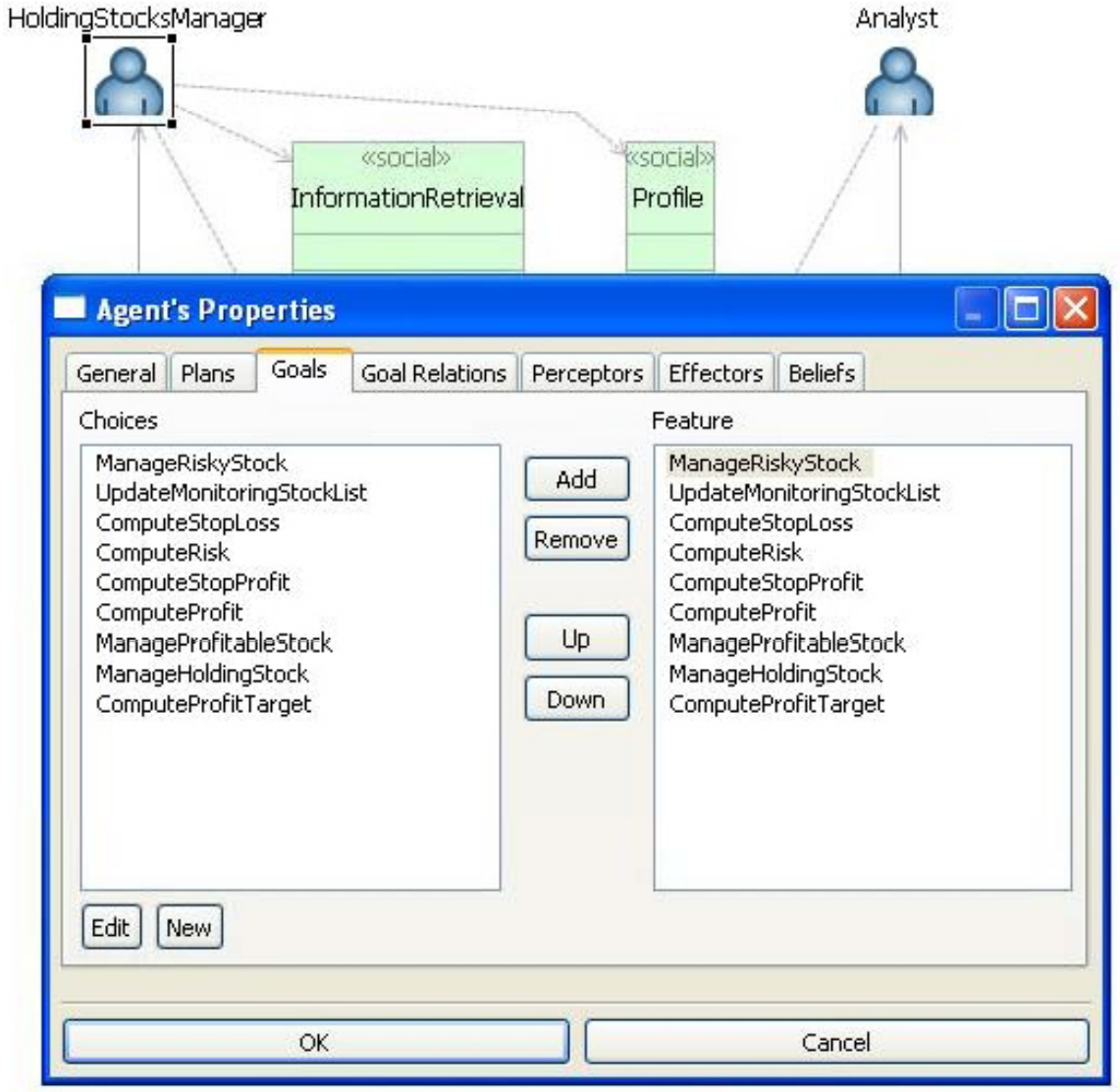


PRACTIONIST

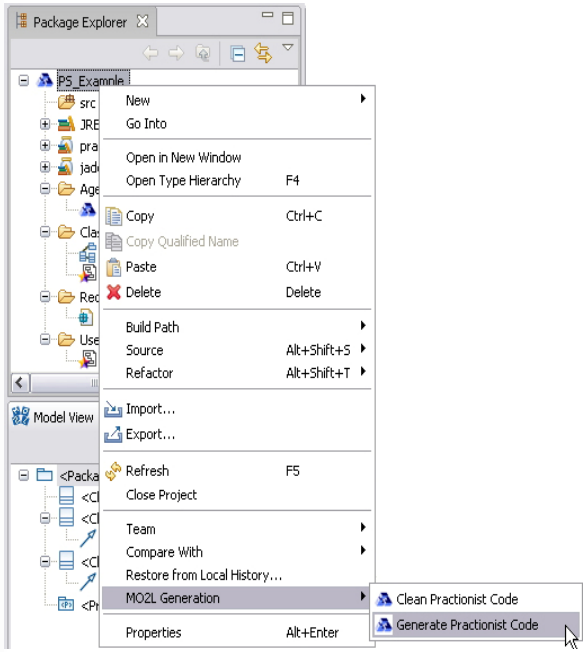
PS - Plan Body Editor



PS – Agent Structure



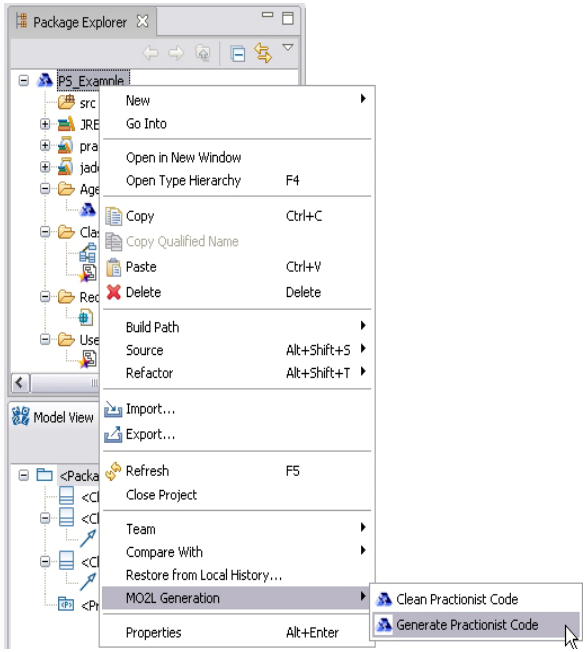
PS – Code generator




```
public class ACLMessagePerceptor extends AbstractPerceptor
{
 ……
  public Perception perceive()
  {
   // TODO:Insert your conditions here in order to return
   // the perception
   if(false)
   {
    // TODO:Insert the Perception's parameters here
    return new ACLMessagePerception();
   }
   return null;
  }
}
```

- Source code generation of the designed system from the developed diagrams
- Implementation of the interfaces and extension of the classes of the framework

PS – Code generator

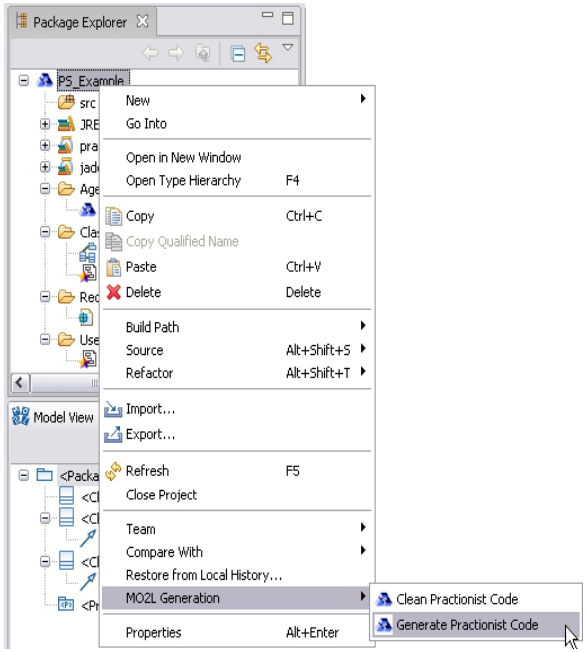


- Source code generation of the designed system from the developed diagrams
- Implementation of the interfaces and extension of the classes of the framework

```
public class ACLMessagePerceptor extends AbstractPerceptor
```

```
public class GetRiskParameters implements Effector
{
……
  public Action perform(Action action)
  {
   // TODO: Auto-generated method stub
   if(action instanceof GetStopLossAction)
    {
      GetStopLossAction getStopLossAction =
            (GetStopLossAction) action;
   // TODO: Manage the action execution and its success condition
     return getStopLossAction;
    }
   if(action instanceof GetToleranceAction)
    {
      GetToleranceAction getToleranceAction =
            (GetToleranceAction) action;
   // TODO: Manage the action execution and its success condition
     return getToleranceAction;
    }
   return null;
  }
……
}
```

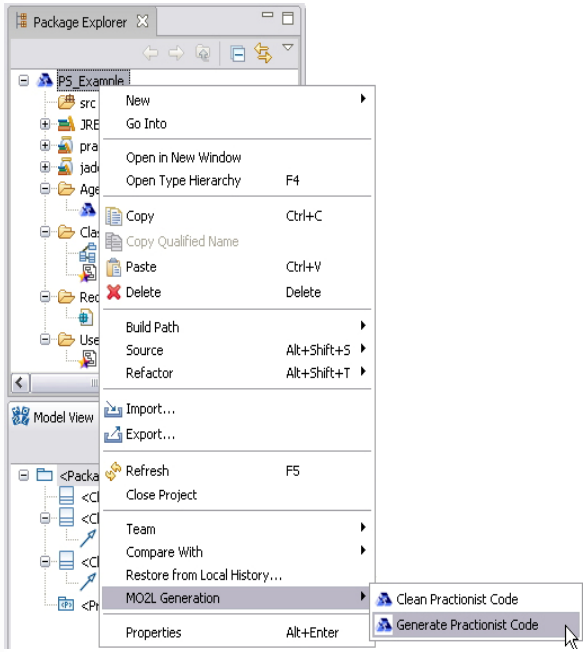
PS – Code generator



- Source code generation of the designed system from the developed diagrams
- Implementation of the interfaces and extension of the classes of the framework

```
public class ACLMessagePerceptor extends AbstractPerceptor
public class GetRiskParameters implements Effector
{
.....
  public Action perform(Action action)
  {
   public class GR_ManageRiskyStocks_ComputeRisk
                                  implements EntailmentRel
   {
    .....
    // TODO: Auto-generated stub
     public Goal verifiesRel(SerializableGoal goal1,
                             SerializableGoal goal2)
     {
      if(goal1 instanceof ManageRiskyStock
         && goal2 instanceof ComputeRisk)
         return new ComputeRisk();
      return null;
     }
    .....
   }
...
}
```

PS – Code generator



- Source code generation of the designed system from the developed diagrams
- Implementation of the interfaces and extension of the classes of the framework

```
public class HoldingStocksManager extends AbstractAgent
{
 ……
 protected void initialize(){
  addBeliefSet("C:/PS_Example/pl/holdingStocksManager.pl");
  /*****************Perceptors*****************/
  // TODO:Remember to put the perceptor's parameters here
  addPerceptor(new ACLMessagePerceptor());
  ……
  /*****************Effectors*****************/
  // TODO:Remember to put the effector's parameters here
  addEffector(new GetRiskParameters());
  ……
  /*****************Goals*****************/
  // TODO:Remember to put the goal's parameters here
  registerGoal(new ManageRiskyStock(), "");
  ……
  /***********Goals**Relations***********/
  ……
  /*****************Plans*****************/
  // TODO:Remember to put the plan's parameters here
  addPlan(TopLevelPlan.class, "TopLevelPlan");
  ……
  // TODO:Remember to put your commit goal here
  commit(new Start(null));
 }
}
```

PRACTIONIST

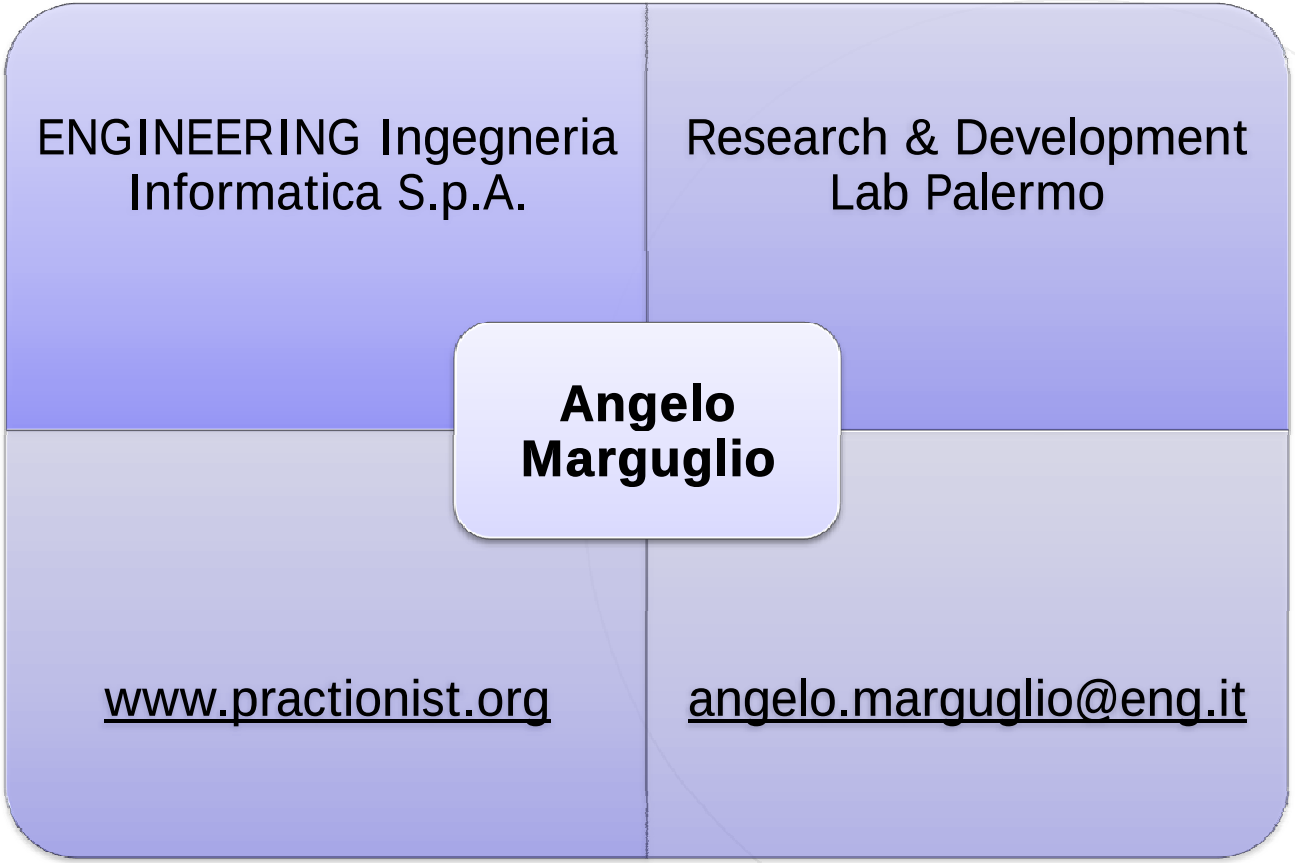
MO2L – MOdelling TOOLs

- **MO2L Architecture**
  - Exploit Eclipse support
  - Common infrastructure
- **BilikUML**
  - UML2 compliant
  - To be improved with other diagrams (especially sequence and activity)
- **RequiSuite**
  - Requirements analysis tools
  - Lack of use case description facilities

Future works

- Other diagrams (with focus on interaction)
- Improve automatic code generation
- Reverse engineering
- Documentation management
- More applications

Thanks!!!



PRACTIONIST