Practical Session

Programming Multiagent Systems
WESAAC 2013

Olivier Boissier	(EMSE, France)
Rafael H. Bordini	(PUC-RS,Brazil)
Jomi F. Hübner	(UFSC, Brazil)
Alessandro Ricci	(Bologna University, Italy)

Scenario (1)

- Giacomo wants to build a house
- We consider two main phases:

1. Contracting specialised companies
   Giacomo hires various companies specialised in different aspects of construction
2. Building the house
   Contractors execute the main workflow for building the house under Giacomo's supervision

Scenario (2)

- Phase 1: Contracting specialised companies
  - The objective here is to hire one company for each of these tasks:

A. Site preparation
B. Lay floors
C. Build walls
D. Build roof
E. Fit windows
F. Fit doors
G. Install the plumbing
H. Install the electrical system
I. Paint the exterior of the house
J. Paint the interior of the house

NB: The same company can be hired for more than 1 task

Scenario (3)

- Phase 2: Building the house
  - After the companies have been hired, they have to execute their tasks on time and in coordination with each other
  - Some tasks depend on others and some tasks can be done in parallel, as represented by the workflow (";" for sequence and "|" for parallel)

$$a ; b ; c ; (d \mid e \mid f) ; (g \mid h \mid i) ; j$$

Scenario (3)

- Phase 2: Building the house
  - After the companies have been hi
    execute their tasks on time and in
    each other
  - Some tasks depend on others and
    be done in parallel, as represented
    (";" for sequence and "|" for parall

(a) Site preparation

(b) Lay floors

(c) Build walls

(d) Build roof

(e) Fit windows

(f) Fit doors

(g) Install the plumbing

(h) Install the electrical system

(i) Paint the exterior of the house

(j) Paint the interior of the house

$$a \; ; \; b \; ; \; c \; ; \; (d \mid e \mid f) \; ; \; (g \mid h \mid i) \; ; \; j$$

An MAS for House Building

- To make the scenario suitable for a course like this, we introduced some simplifications in the system design
- We summarise the solution using our approach, commenting on:
  - Agents
  - Contracting Phase (using Environment)
  - Building Phase (using Organisation)

Agents

- Two types of agents:
    - **House owner** (Giacomo): provides the requirements for the house, with budget limitations
    - **Companies**: they will offer their service and, if hired, will execute the house building tasks; they are characterised by their competences in house building

Agents Implementation

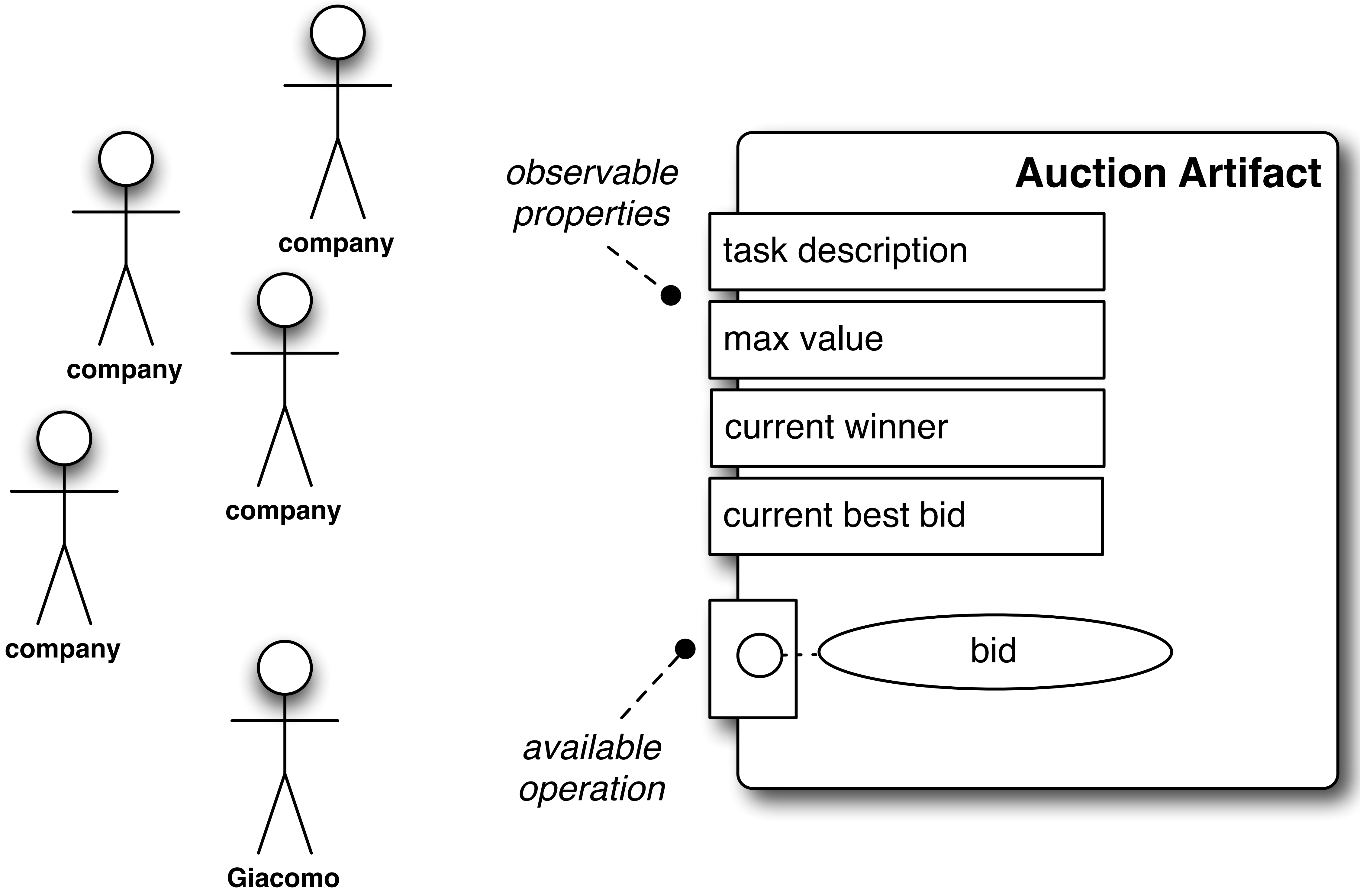
- The house owner is programmed in Jason
- Companies are programmed in Jason
  (or 2APL, Jadex, ...)
- A heterogeneous agent system

Contracting Phase

- Electronic **auctions** will be used to hire the required companies
- One auction for each task
- Each auction is started with:
    - the task description
    - the maximum value the owner can pay for it
- By the end of an auction, the company to be hired for that task is determined

Environment Implementation

- An **auction artifact** encapsulates the auction mechanism
- Giacomo creates instances of such artifacts for creating/managing the various auctions; one such auction is used for hiring companies for each of the house building tasks
- Companies can perceive those artifacts and bid according to their competence and following their own strategies
- After some time Giacomo decides to finish the auction, observing the current best bid shown on the artifact


company
company
company
company
Giacomo
observable properties
Auction Artifact
task description
max value
current winner
current best bid
bid
available operation

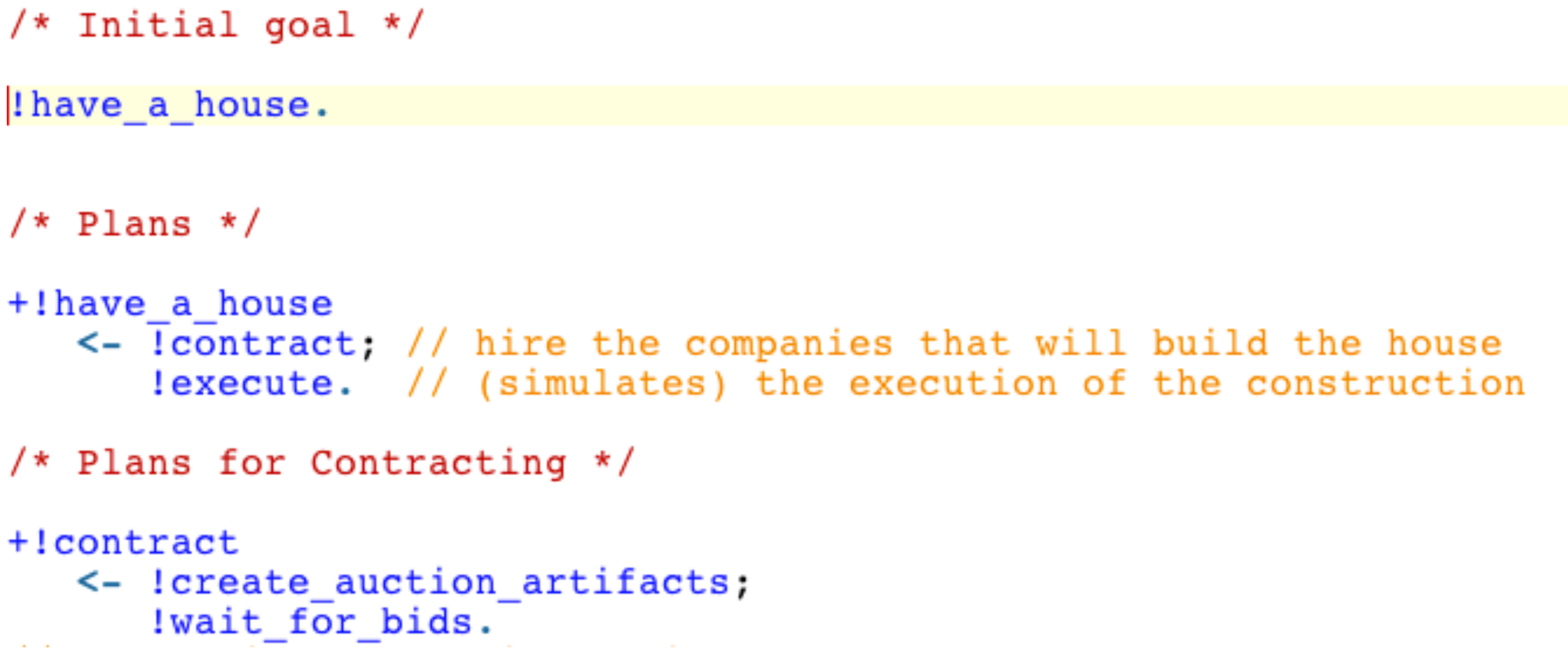
src/env/tools/AuctionArt.java

```java
/**
 *      Artifact that implements the auction.
 */
public class AuctionArt extends Artifact {


    @OPERATION public void init(String taskDs, int maxValue)  {
        // observable properties
        defineObsProperty("task",          taskDs);
        defineObsProperty("maxValue",      maxValue);
        defineObsProperty("currentBid",    maxValue);
        defineObsProperty("currentWinner", "no_winner");
    }

    @OPERATION public void bid(double bidValue) {
        ObsProperty opCurrentValue  = getObsProperty("currentBid");
        ObsProperty opCurrentWinner = getObsProperty("currentWinner");
        if (bidValue < opCurrentValue.intValue()) {  // the bid is better
            opCurrentValue.updateValue(bidValue);
            opCurrentWinner.updateValue(getOpUserName());
        }
    }

}
```

Contracting: Agent Side

- Giacomo has plans to launch all auctions by creating the corresponding artifacts
- Company agents have plans to look for the auction artifacts of their interest and plans defining their own bidding strategy
- After some time Giacomo looks at the best bid in each auction artifact and awards a contract for the winning company

src/agt/giacomo.asl

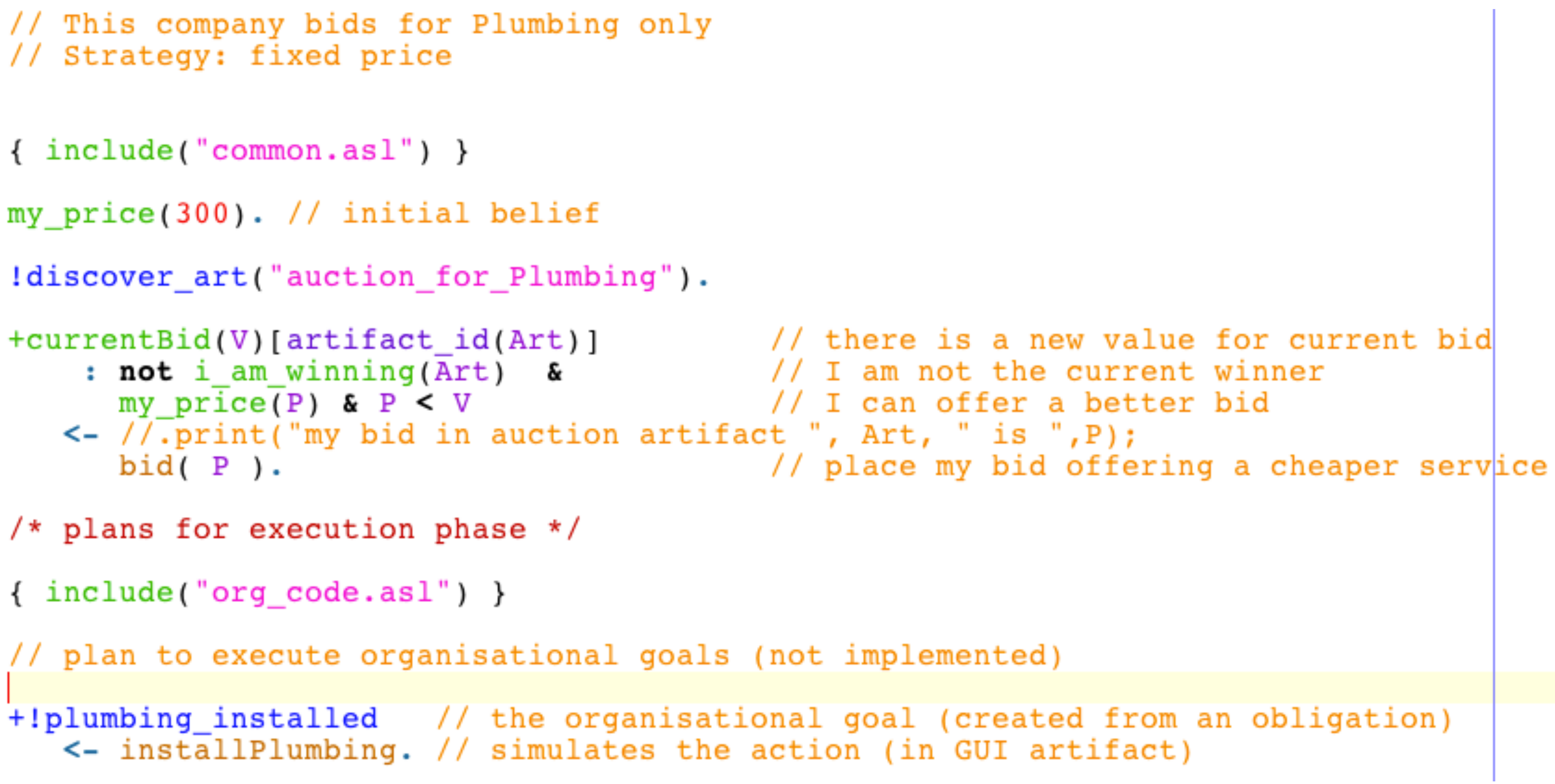


```
/* Initial goal */

!have_a_house.


/* Plans */

+!have_a_house
   <- !contract; // hire the companies that will build the house
      !execute.  // (simulates) the execution of the construction

/* Plans for Contracting */

+!contract
   <- !create_auction_artifacts;
      !wait_for_bids.
```

src/agt/giacomo.asl

```
+!create_auction_artifacts
   <-  !create_auction_artifact("SitePreparation", 2000); // 2000 is the max
       !create_auction_artifact("Floors",          1000);
       !create_auction_artifact("Walls",           1000);
       !create_auction_artifact("Roof",            2000);
       !create_auction_artifact("WindowsDoors",    2500);
       !create_auction_artifact("Plumbing",         500);
       !create_auction_artifact("ElectricalSystem", 500);
       !create_auction_artifact("Painting",        1200).

+!create_auction_artifact(Task,MaxPrice)
   <- .concat("auction_for_",Task,ArtName);
      makeArtifact(ArtName, "tools.AuctionArt", [Task, MaxPrice], ArtId);
      focus(ArtId).
-!create_auction_artifact(Task,MaxPrice)[error_code(Code)]
   <- .print("Error creating artifact ", Code).

+!wait_for_bids
   <- println("Waiting the bids for 5 seconds...");
      .wait(5000); // use an internal deadline of 5 seconds to close the auc
      !show_winners.
```

src/agt/companyA.asl

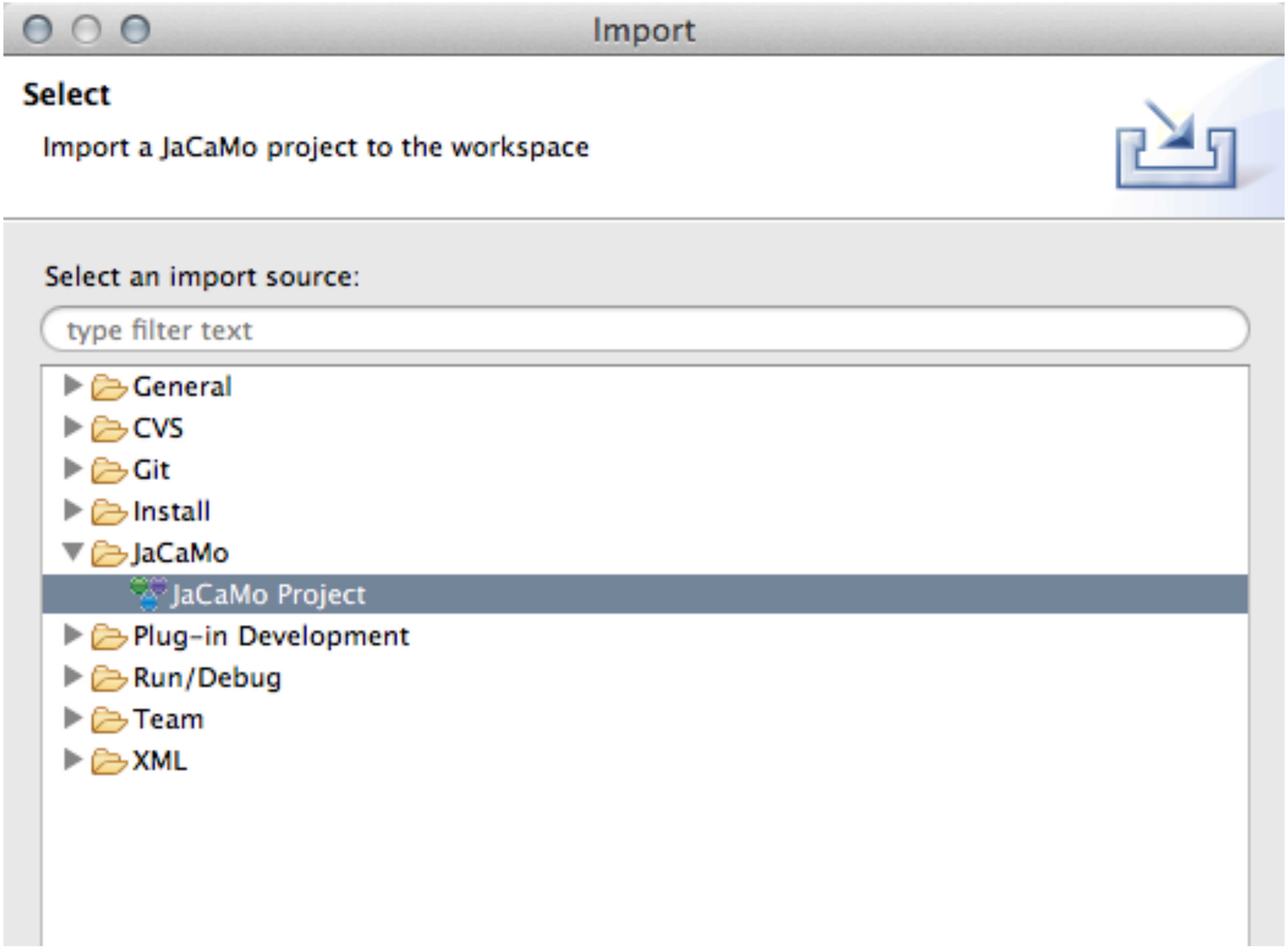


```
// This company bids for Plumbing only
// Strategy: fixed price


{ include("common.asl") }

my_price(300). // initial belief

!discover_art("auction_for_Plumbing").

+currentBid(V)[artifact_id(Art)]          // there is a new value for current bid
     : not i_am_winning(Art)  &           // I am not the current winner
       my_price(P) & P < V                // I can offer a better bid
    <- //.print("my bid in auction artifact ", Art, " is ",P);
       bid( P ).                          // place my bid offering a cheaper service

/* plans for execution phase */

{ include("org_code.asl") }

// plan to execute organisational goals (not implemented)

+!plumbing_installed   // the organisational goal (created from an obligation)
   <- installPlumbing. // simulates the action (in GUI artifact)
```

Eclipse Project

Requirements

- JaCaMo >= 0.4 (link), that includes
  - The code of the application used in this tutorial
  - Documentation
  - More examples
- Eclipse >= Luna
- JaCaMo Eclipse plugin (link)


Import

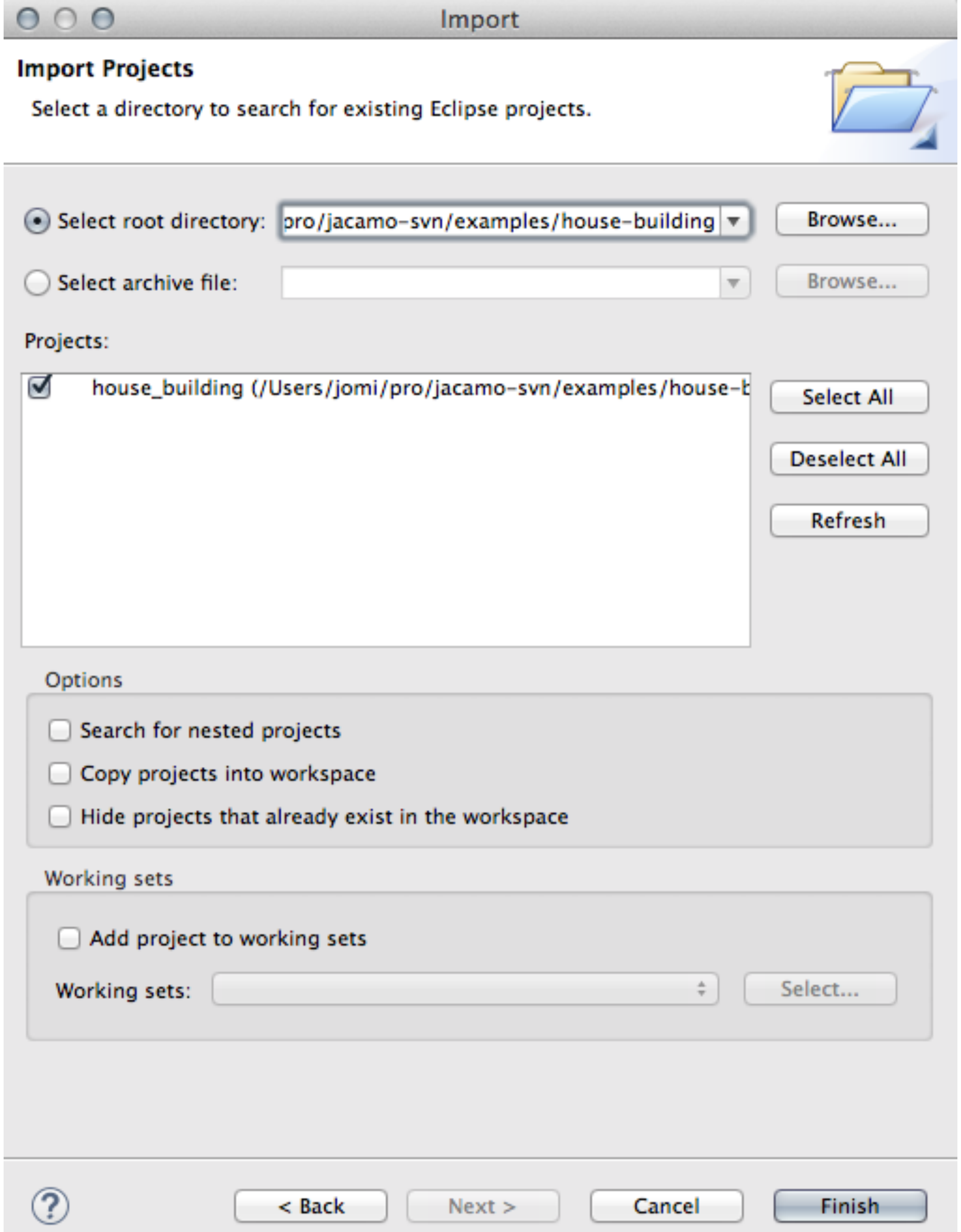
Select

Import a JaCaMo project to the workspace

Select an import source:

type filter text

- General
- CVS
- Git
- Install
- JaCaMo
  - JaCaMo Project
- Plug-in Development
- Run/Debug
- Team
- XML



Import Projects

Select a directory to search for existing Eclipse projects.

(● Select root directory:) pro/jacamo-svn/examples/house-building

Select archive file:

Projects:

[✓] house_building (/Users/jomi/pro/jacamo-svn/examples/house-b

Options

[ ] Search for nested projects

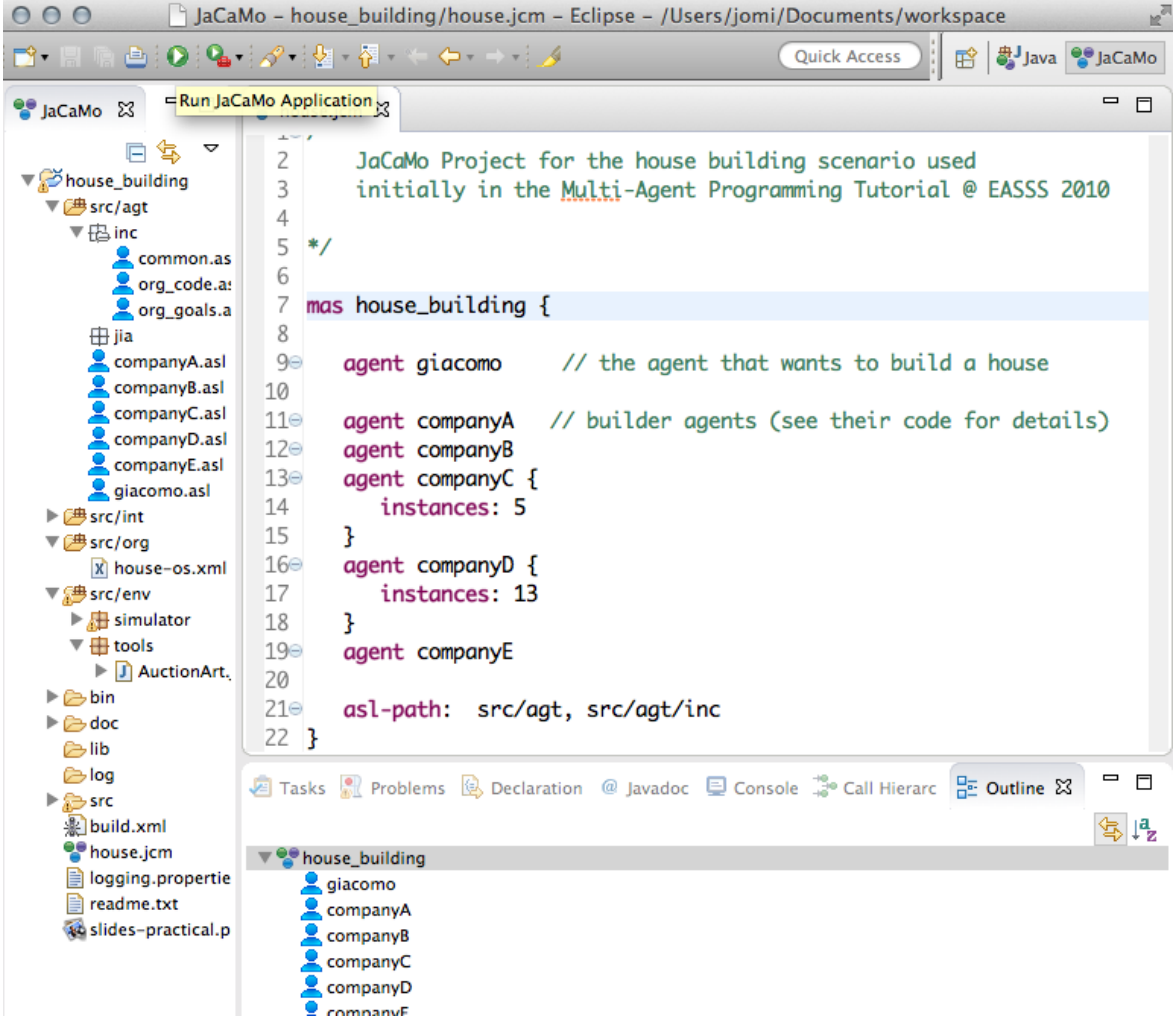
[ ] Copy projects into workspace

[ ] Hide projects that already exist in the workspace

Working sets

[ ] Add project to working sets

Working sets:



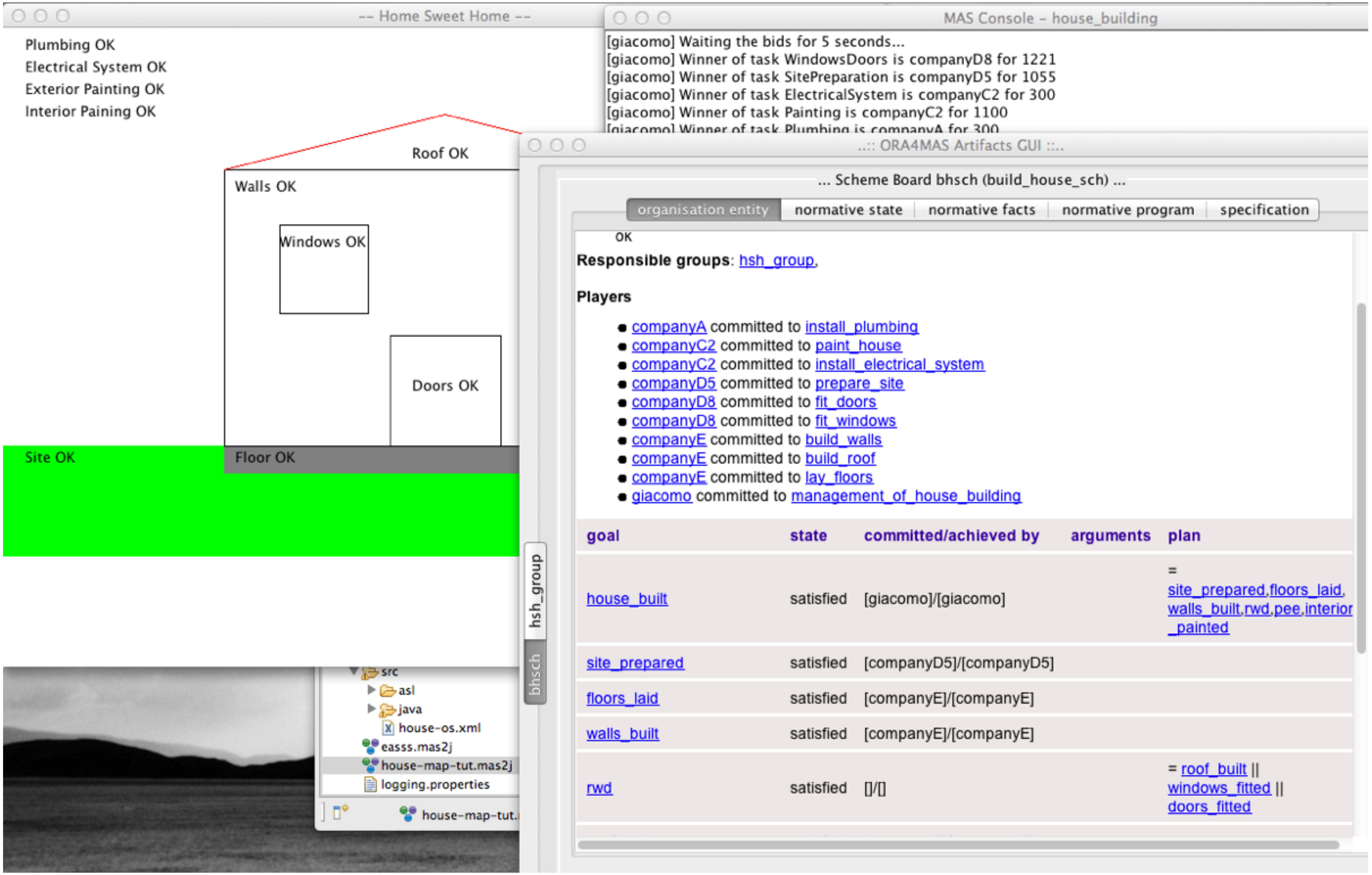
JaCaMo – house_building/house.jcm – Eclipse – /Users/jomi/Documents/workspace

Run JaCaMo Application

```
    JaCaMo Project for the house building scenario used
    initially in the Multi-Agent Programming Tutorial @ EASSS 2010

*/

mas house_building {

   agent giacomo     // the agent that wants to build a house

   agent companyA   // builder agents (see their code for details)
   agent companyB
   agent companyC {
      instances: 5
   }
   agent companyD {
      instances: 13
   }
   agent companyE

   asl-path:  src/agt, src/agt/inc
}
```

```
house_building
  src/agt
    inc
      common.as
      org_code.a:
      org_goals.a
    jia
    companyA.asl
    companyB.asl
    companyC.asl
    companyD.asl
    companyE.asl
    giacomo.asl
  src/int
  src/org
    house-os.xml
  src/env
    simulator
    tools
      AuctionArt.
  bin
  doc
  lib
  log
  src
  build.xml
  house.jcm
  logging.propertie
  readme.txt
  slides-practical.p
```

```
house_building
  giacomo
  companyA
  companyB
  companyC
  companyD
  companyE
```



```
[giacomo] Waiting the bids for 5 seconds...
[giacomo] Winner of task WindowsDoors is companyD8 for 1221
[giacomo] Winner of task SitePreparation is companyD5 for 1055
[giacomo] Winner of task ElectricalSystem is companyC2 for 300
[giacomo] Winner of task Painting is companyC2 for 1100
[giacomo] Winner of task Plumbing is companyA for 300
```

... Scheme Board bhsch (build_house_sch) ...

organisation entity | normative state | normative facts | normative program | specification

OK

Responsible groups: hsh_group,

Players

- companyA committed to install_plumbing
- companyC2 committed to paint_house
- companyC2 committed to install_electrical_system
- companyD5 committed to prepare_site
- companyD8 committed to fit_doors
- companyD8 committed to fit_windows
- companyE committed to build_walls
- companyE committed to build_roof
- companyE committed to lay_floors
- giacomo committed to management_of_house_building

goal	state	committed/achieved by	arguments	plan
house_built	satisfied	[giacomo]/[giacomo]		= site_prepared,floors_laid, walls_built,rwd,pee,interior_painted
site_prepared	satisfied	[companyD5]/[companyD5]		
floors_laid	satisfied	[companyE]/[companyE]		
walls_built	satisfied	[companyE]/[companyE]		
rwd	satisfied	[]/[]		= roof_built || windows_fitted || doors_fitted

Exercises

1. Change the code of the auction artifact to:
   A. create a new observable property that shows the state of the auction (open or closed)
   B. add a new operation `clearAuction` (after `clearAuction`, the state of the auction becomes closed and attempts to use the `bid` operation will fail)
2. Change the house owner program so that the agent uses the new `clearAuction` operation

Homework

1. Change the auction artifact so that it shows and manages the bidding deadline and has a new operation for starting the auction; the `clearAuction` operation is no longer needed
2. Create a new company for one of the tasks and give it any bidding strategy you like

Homework

3. Choose a new auction mechanism and implement a new auction artifact that implements that mechanism; you should not change the agents for this exercise

4. Now choose another mechanism that will require different strategies in the agents and implement them

Moise Specification

Building Phase

- After all auctions are over, Giacomo sends messages for the hired companies to enter into the execution phase
- A virtual organisation is created to assist with coordination and cooperation in the execution of the global workflow
- Implementation choice:
  - The organisation is specified using the Moise organisation modelling language

Building: Organisation Side

- Moise functional specification is used to define the workflow
- Moise structural specification is used to define the role and group structures
- Moise normative specification is used to distribute the tasks of the workflow to the roles

Functional Specification

- The functional specification simply defines a social scheme for the global workflow

$$a\ ;\ b\ ;\ c\ ;\ (d \mid e \mid f)\ ;\ (g \mid h \mid i)\ ;\ j$$

- One **mission** for each task except for the painting of the exterior and of the interior of the house that are grouped into the same mission
- A task for the management of the execution of the workflow is also added

Functional Specification

- The functional specification simply
social scheme for the global work

a ; b ; c ; (d | e | f) ; (g | h |

- One **mission** for each task except
of the exterior and of the interior o
are grouped into the same missio

- A task for the management of the e
workflow is also added

(a) Site preparation

(b) Lay floors

(c) Build walls

(d) Build roof

(e) Fit windows

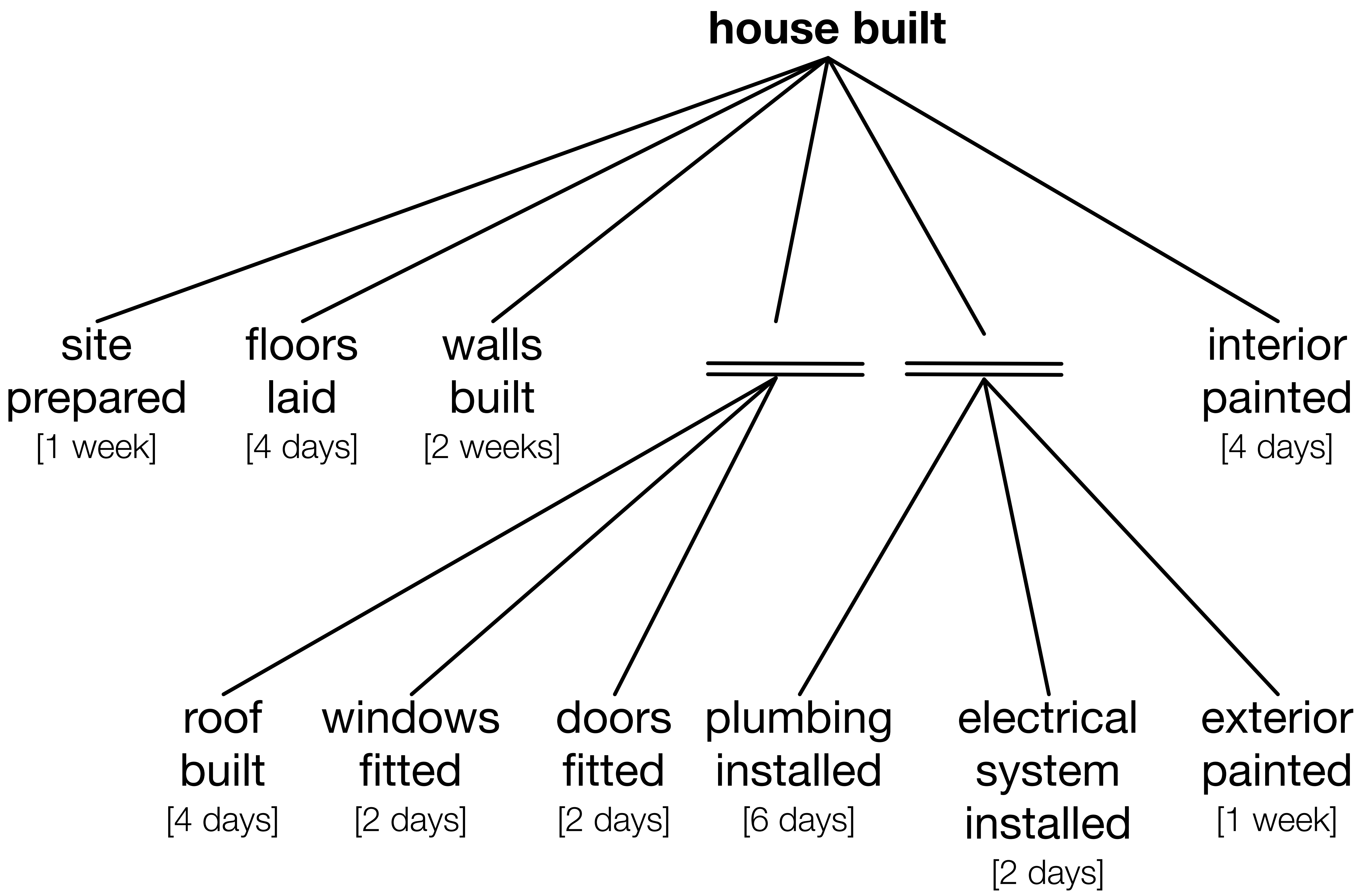
(f) Fit doors

(g) Install the plumbing

(h) Install the electrical system

(i) Paint the exterior of the house

(j) Paint the interior of the house


house built
site prepared [1 week]
floors laid [4 days]
walls built [2 weeks]
interior painted [4 days]
roof built [4 days]
windows fitted [2 days]
doors fitted [2 days]
plumbing installed [6 days]
electrical system installed [2 days]
exterior painted [1 week]

Structural Specification (1)

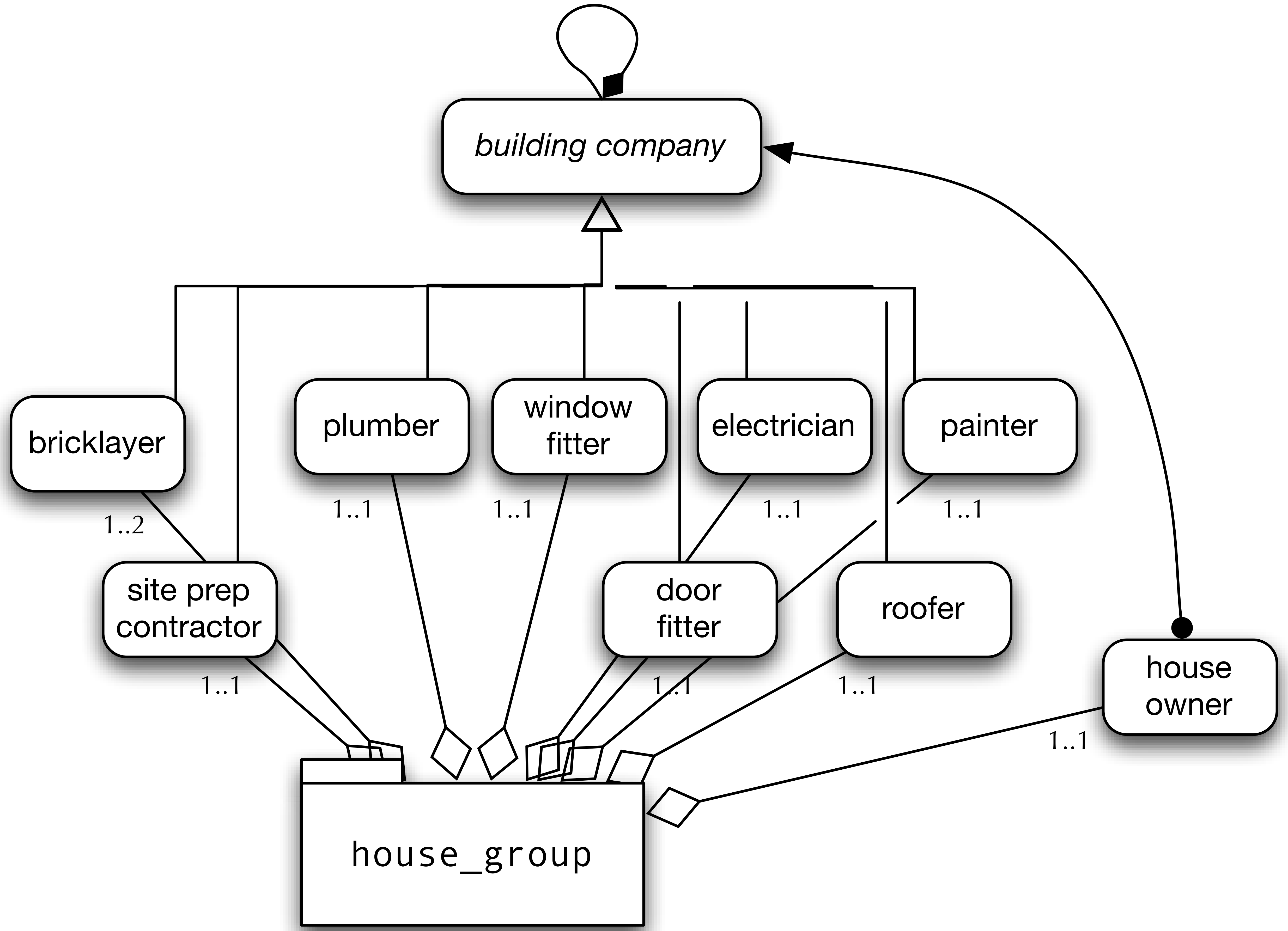
- Role hierarchy
  - `house_owner`
  - `building_company`
    - abstract role, specialised into: `site_prep_contractor`, `bricklayer`, `roofer`, `window_fitter`, `door_fitter`, `plumber`, `electrician`, `painter`

Structural Specification (2)

- The roles are used in a **group** called ‘`house_group`’ where:
  - `house_owner` has cardinality (1,1)
  - `site_prep_contractor` has cardinality (1,1)
  - `bricklayer` has cardinality (1,2)
  - `roofer` has cardinality (1,1)
  - `window_fitter` has cardinality (1,1)
  - `door_fitter` has cardinality (1,1)
  - `plumber` has cardinality (1,1)
  - `electrician` has cardinality (1,1)

Structural Specification (3)

- Notes:
  - the role `building_company` is compatible with `building_company` so that the same agent can play more subroles
  - role `house_owner` has authority over the `building_company` role
  - a communication link connects the role `build_company` to `house_owner`


building company
bricklayer
plumber
window fitter
electrician
painter
site prep contractor
door fitter
roofer
house owner
house_group
1..2
1..1
1..1
1..1
1..1
1..1
1..1
1..1
1..1

Normative Specification (1)

- Given the definition of the **missions**, the following **norms** are defined:
  - any agent playing the **role** `houseOwner` is **obliged** to **commit** to **mission** ‘`mManagement`’
  - role `site_prep_contractor` to the mission concerning the site preparation goal
  - role `bricklayer` to the mission of laying the floors
  - `bricklayer` is also obliged to commit to the mission of building the walls

Normative Specification (2)

- role `window_fitter` to the mission related to fitting the windows
- `door_fitter` is obliged to commit to the mission of fitting the doors
- `plumber` to installing the plumbing
- `electrician` to installing the electrical system
- `painter` to the mission concerning the painting of the house

Building: Agent Side

- Owner agent is equipped with the plans to construct the virtual organisation based on the result of contracting phase
- Company agents have plans to enter the organisation, adopt the role corresponding to their contract and to catch the different events generate by the OMI
- Companies have plans to execute autonomously the various actions related to the goals related to the missions they are committed to in the organisation scheme
- NB: agents are benevolent with respect to the organisation, i.e. they don't violate the norms

Building: Environment Side

- Artifacts that model the state of the environment (e.g., model the state of the construction of a wall)

Exercises

1. Do the following changes in the organisation specification:
   A. tasks `site_preparation` and `lay_floors` can be done in parallel
   B. all tasks have to be done in sequence

Homework

1. Develop an agent that tries to adopt roles related to tasks he is not supposed to (malevolent agent!)

   (e.g. Giacomo trying to play some company role)

2. Develop an agent that does not fulfill the tasks

Homework

3. Change the Giacomo agent so that it reacts to the norm violation
    - Giacomo should create a new auction for that task and forbid the violating company from taking part in the new auction

4. Change the system to build two houses in parallel

Homework

5. Change the Giacomo agent so that it reads the Moise specification and creates the necessary auction artifacts based on the specified tasks

6. Change Giacomo so that it is able to monitor the building of the house and check whether the tasks are being done appropriately

Conclusions

- Separation of concerns
  - agent, environment, organisation, interaction
  - MAP = AOP + EOP + OOP + IOP
- Design == Implementation
- In our code, (almost) no (direct) communication in the MAS!