Програмування інтелектуальних інформаційних систем

3 курс, осінь 2021

- Доц. Баклан І.В.
- Email: iaa@ukr.net
- Web: baklaniv.at.ua

Лекція 17

Введення до платформи JaCaMo

У цій лекції ми представляємо JaCaMo, особливу платформу, прийняту в нашій дисципліні для практичного багатоагентного орієнтованого програмування. Ця платформа підтримує практичне програмування на основі абстракцій, представлених у попередньому розділі: програмування організованих агентів, розташованих у спільному середовищі. JaCaMo побудований поверх трьох існуючих платформ, які розроблялися роками (Boissier et al. 2013, 2019), а саме Jason (Bordini et al. 2007) для агентів програмування, CArtAgO (Ricci et al. 2009) для середовищ програмування та Moise (Hübner et al. 2007) для організацій з програмування.

У цій лекції класичний приклад Hello-World має версію з кількома агентами. Ми починаємо з найпростішої програми, яку ми можемо написати на JaCaMo, і вдосконалюємо її, щоб поступово показувати деякі найважливіші аспекти мови програмування та самої платформи.

Вказівки щодо створення, редагування та запуску додатків JaCaMo можна знайти у другій частині цієї лекції.

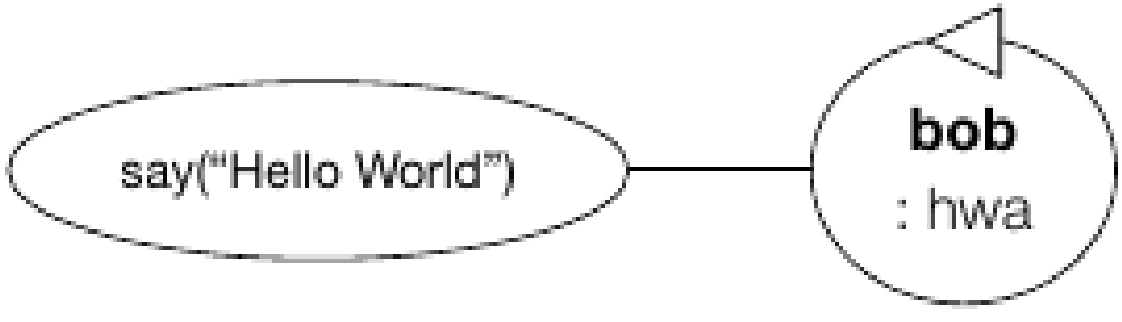
Одноагентний Hello-World

Ми починаємо з системи, яка має єдиного і дуже простого агента, який просто роздруковує повідомлення, використовуючи наступний план, написаний на Jason (і зберігається у файлі під назвою **`hwa.asl`**):

```
+!say(M) <- .print(M).
```

Цей план можна читати як "коли я маю мету! Скажіть (**`M`**), досягніть її, надрукувавши значення змінної **`M`**" (**`M`** - це змінна, оскільки вона починається з великої літери).

Для запуску агента JaCaMo використовує файли програм (імена яких закінчуються на **`.jcm`**). У нашому прикладі файлом програми є **`sag_hw.jcm`**, в якому ми даємо ім'я агенту (**`bob`**) та початкову мету **`(скажімо ("Hello World"))`**. Вміст цього файлу, зображеного графічно на малюнку 17.1, виглядає наступним чином:



Мал. 17.1 Конфігурація Hello-World для одного агента.

```
mas sag_hw { // the MAS is identified by sag_hw

agent bob: hwa.asl{// initial plans for bob are in
hwa.asl

goals: say("Hello World") // initial goal for bob

}

}
```

Результат виконання маємо такий

```
JaCaMo Http Server running on http://192.168.0.15:3272

[bob] Hello World
```

Для кращого розуміння результатів виконуються такі кроки у виконанні файлу програми (`.jcm`):

1. Агент на ім’я **`bob`** створюється з початковими переконаннями, цілями та планами, як це визначено зі змісту файлу з назвою **`hwa.asl`**.

2. Мета **`say ("Hello World")`** делегується **`bob`**, створюючи подію

```
+! say ("Hello World").
```

3. План у файлі **`hwa.asl`**, як показано раніше, запускається і використовується для обробки цієї події.

4. Виконання плану дає результат `[bob] Hello World` як результат виконання внутрішньої дії `.print`.

5. Агент продовжує працювати, але не має нічого робити, оскільки він є єдиним агентом у системі і сам не генерував жодних подальших цілей або змін у середовищі, які могли б привести його до подальших дій.

6. Як показано у результатах виконання, існує URL-адреса для перевірки поточного стану агентів (що включає їх переконання, наміри та плани), і згодом ми побачимо, що те саме стосується середовища та організацій.

Мультиагентний Hello-World

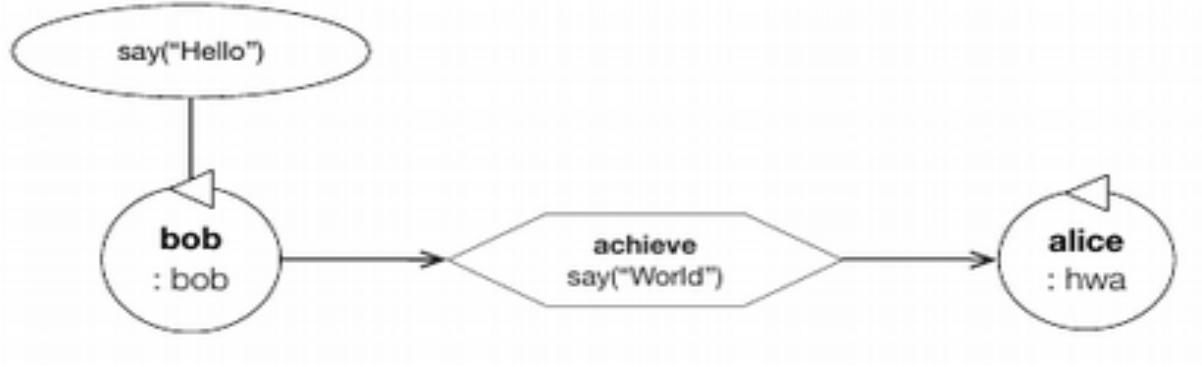
Тепер у нас є два агенти - Боб і Аліса. Агент Боб друкує "Hello", а Аліса - "World". Для того, щоб створити обидва агента з одного коду (як і в попередньому прикладі, що має лише один план), ми можемо використовувати такий файл програми:

```
mas mag_hw {
      agent bob: hwa.asl {
            goals: say("Hello")
      }
      agent alice: hwa.asl {
            goals: say("World")
      }
}
```

Однак результат виконання може бути таким:

```
[alice] World
[bob] Hello
```

Агенти працюють одночасно і асинхронно переслідують свої цілі, і тому така початкова реалізація не може гарантувати порядок надрукованих повідомлень. Потрібна деяка координація, щоб боб друкував першим, а потім Аліса. Ми можемо вирішити проблему, коли Боб надішле повідомлення Алісі, як тільки його повідомлення буде надруковане (див. Малюнок 17.2).



17.2 Координація комунікацій

Нова програма виглядає наступним чином (буде включена у файл з назвою bob.asl):

```
+!say(M) <- .print(M);
.send(alice,achieve,say("World")).
```

Надсилаючи повідомлення про досягнення Alicb, Боб делегує Алісі ціль `say("World")`. Вона використовує план `+! say (M) <- .print (M)`. для досягнення мети, як і раніше.

Оскільки мета Аліси тепер походить від Боба, а не від ініціалізації системи, файл програми потрібно змінити таким чином:

```
mas mag_hw {
agent bob { // file bob.asl is used
goals: say("Hello")
}
agent alice: hwa.asl
}
```

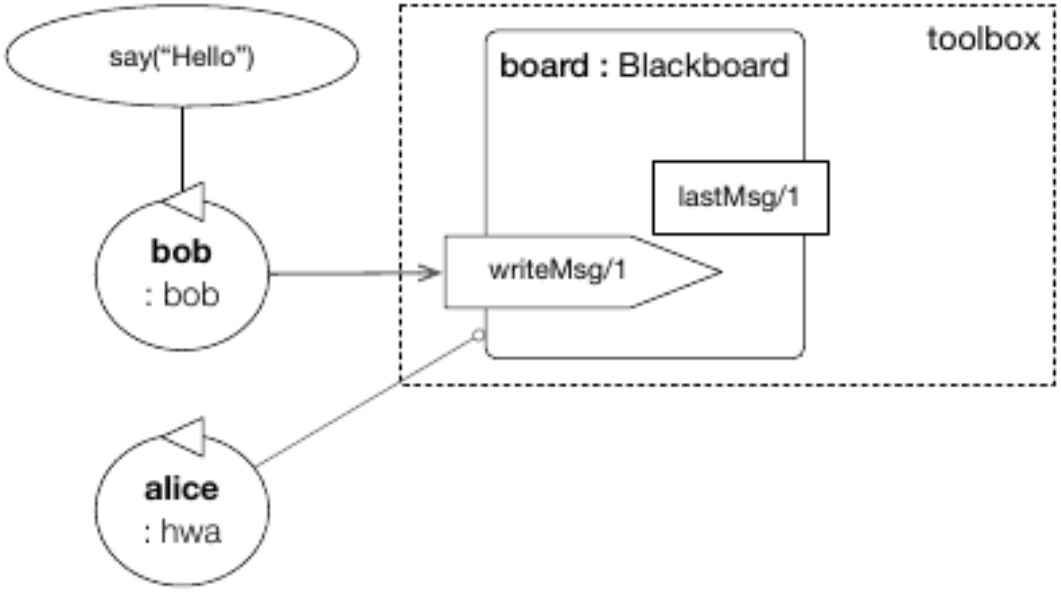
Середовище Hello-World

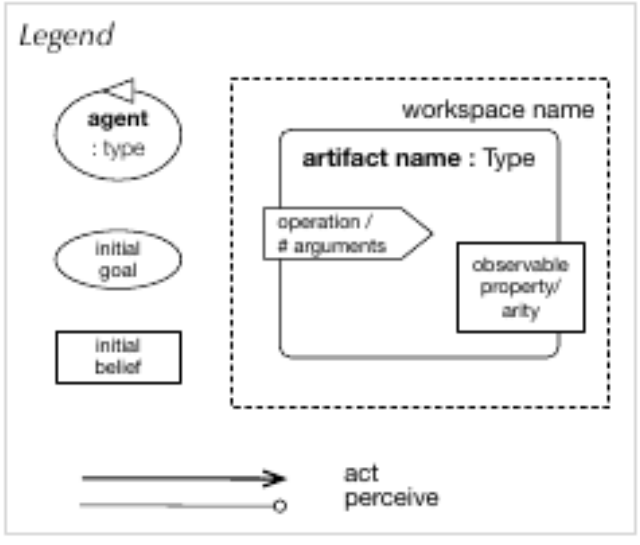
Приклад тепер розглядає середовище з артефактом дошки як дошку, яку агенти можуть використовувати для написання повідомлень та сприйняття повідомлень, написаних на ньому. У цій версії прикладу Hello-World Боб пише повідомлення “Hello” на дошці; і Аліса, яка спостерігає за дошкою, пише повідомлення «World», як тільки вона переконається, що повідомлення «Hello» написано.

Середовища структуровані у робочі області; всі агенти в робочій області мають спільний доступ до всіх екземплярів артефактів у цій робочій області. У файлі програми ми можемо вказати початковий набір артефактів та робочих областей, які слід створити під час створення MAS. У цьому випадку файл `sit_hw.jcm` має такий вигляд:

```
mas sit_hw {
  agent bob {
    join: room  // bob joins workspace toolbox
    goals: say("Hello")
  }
  agent alice {
    join: room // alice also joins workspace toolbox
    focus: room.board  // and focus on artifact board
  }
  workspace room {  // creates the workspace toolbox
    artifact board: tools.Blackboard // with artifact
board
  }
}
```

Початкова конфігурація включає робочу область під назвою room, де розміщено артефакт дошки з інструментами типу. Чорна дошка (див. малюнок 17.3). Обидва агенти приєднуються до кімнати робочої області під час ініціалізації, щоб отримати доступ до артефакту плати. Агент Аліса зосереджується на (тобто спостерігає) артефакті. Необхідно зосередитися на тому, щоб агент Аліса була уважною до змін у спостережуваних властивостях цього артефакту: коли щось написано наступного разу, коли Аліса відчує навколишнє середовище, переконання, що відповідає властивості спостережуваного артефакту, буде автоматично створено, і вона зможе реагувати на це.





17.3 Координація з використанням середовища.

Артефакти реалізовані на Java. Вихідний код (у файлі **Blackboard.java**) простого артефакту дошки виглядає наступним чином:

```
package tools;
import cartago.*;
public class Blackboard extends Artifact {
  void init() {
    defineObsProperty("lastMsg","");
  }
  @OPERATION void writeMsg(String msg) {
    System.out.println("[BLACKBOARD] " + msg);
    getObsProperty("lastMsg").updateValue(msg);
  }
}
```

Класи Java використовуються як шаблони для визначення артефактів, з використанням коментованих методів для визначення операцій з артефактами та заздалегідь визначених методів, успадкованих **API** **Artifact** для роботи з властивостями, що спостерігаються, та іншими механізмами артефактів.

Вихідним кодом для bob в цьому випадку стає

```
+!say(M) <- writeMsg(M).
{ include("$jacamoJar/templates/common-cartago.asl") }
```

Тобто агент використовує дію **writeMsg**, надану артефактом, для запису повідомлення на дошці. Інструкція включення завантажує деякі корисні плани в бібліотеку планів bob.

Вихідний код Alice – це

```
+lastMsg("Hello") <- writeMsg("World!").
{ include("$jacamoJar/templates/common-cartago.asl") }
```

Агент (який спостерігає за дошкою) пише повідомлення "`World`", як тільки він переконається, що останнє повідомлення, написане на дошці (зроблене для спостереження за допомогою властивості `lastMsg`) – "`Hello`".

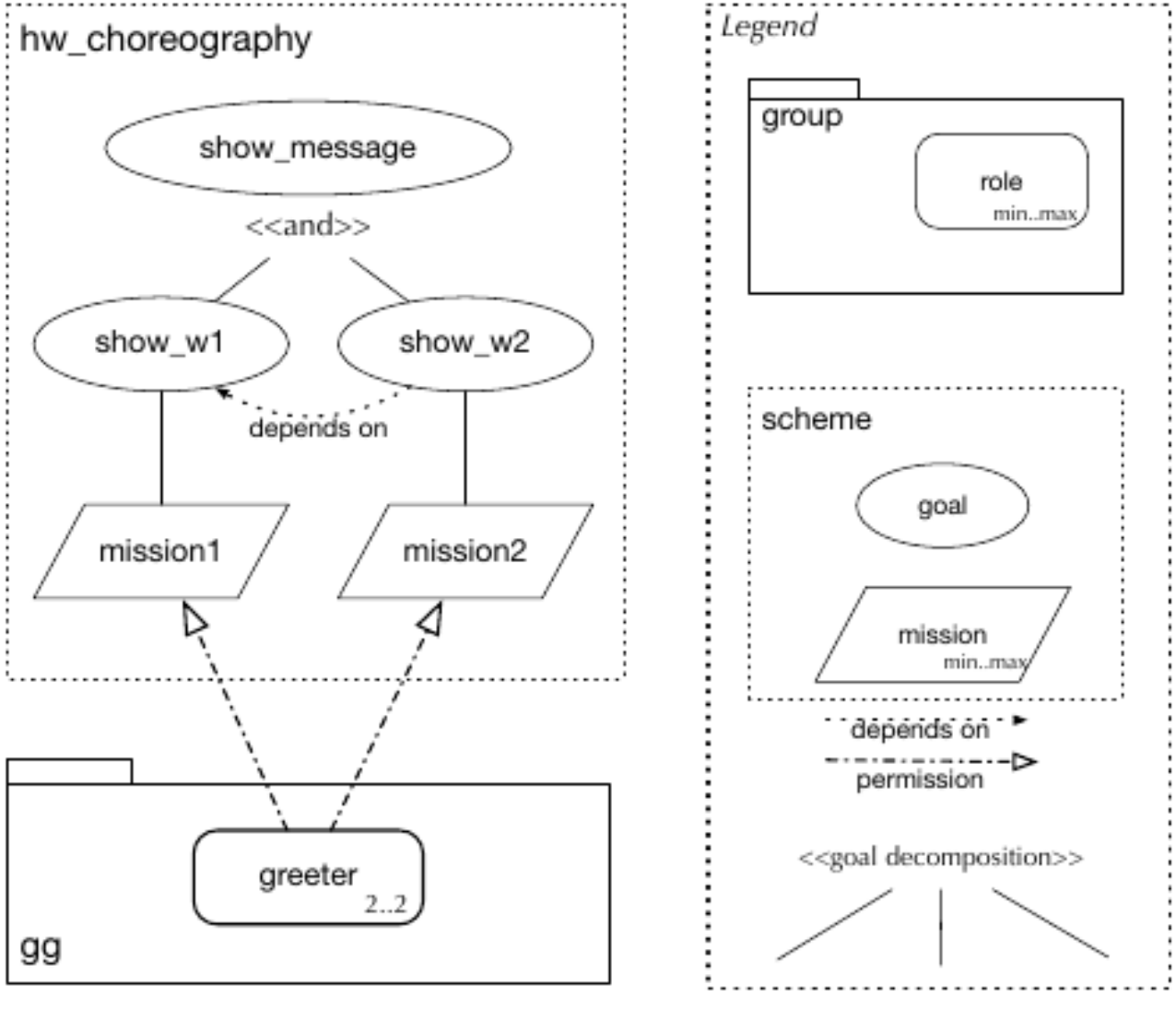
```
[alice] joined workspace room
[alice] focusing on artifact board (at workspace room)
using namespace default
[bob] joined workspace room
[BLACKBOARD] Hello
[BLACKBOARD] World!
```

Виконання файлу програми дає результат, подібний до попередніх, за винятком того, що зараз це артефакт дошки, що роздруковує повідомлення, і ніякого зв'язку між Бобом та Алісою не потрібно.

Організація Hello-World

Тепер ми організовуємо набір агентів для створення повідомлення `"Hello World"`. Як було представлено в попередньому розділі, організація може бути використана для регулювання та координації діяльності агентів. Хоча приклад простий, використання організації полегшує зміну певної схеми координації та регулювання. У нашому прикладі шаблон координації використовується для досягнення мети `show_message`, яка повинна бути досягнута спільною роботою двох агентів і, таким чином, є спільною метою. Щоб відрізнити таку мету від мети агента, ми називаємо її *організаційною метою*.

Щоб відрізнити таку мету від мети агента, ми називаємо її організаційною метою. Ми використовуємо соціальну схему для програмування того, як організаційна мета `show_message` розкладається на підцілі, які призначаються агентам (як показано на малюнку 17.4). Для декомпозиції мета `show_message` має одну підціль для кожного слова повідомлення. Для їх призначення агентам ми створюємо місії, в даному випадку по одній для кожної підцілі. Для того, щоб брати участь у виконанні схеми, агенти повинні взяти участь у місії та досягти відповідних цілей цієї місії. Здійснення до місії - це форма обіцянки групі агентів, які спільно працюють за схемою: «Я обіцяю, що, коли буде потрібно, я виконаю свою частину завдання». Коли агенти виконують усі завдання, схему можна виконати з гарантією того, що, принаймні в принципі, у нас достатньо агентів для роботи над усіма необхідними підцілями.



17.4 Координація за організацією: специфікація Hello-World.

Цей приклад організації також визначає єдину роль, яку будуть грати всі агенти: роль привітання відіграється у типі групи, ідентифікованій `gg` (для «привітальної групи»). Агенти, які виконують цю роль (і тільки вони), мають право виконувати завдання місії.

Реалізація цієї організації написана в XML так:

```
1   <?xml version="1.0" encoding="UTF-8"?>
2
3   <organisational-specification
4     id="hello_world"
5     os-version="0.8"
6
7     xmlns='http://moise.sourceforge.net/os'
8     xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
9     xsi:schemaLocation='http://moise.sourceforge.net/os
10                        http://moise.sourceforge.net/xml/os.xsd' >
11
12  <structural-specification>
13    <group-specification id="gg">
14      <roles>
15        <role id="greeter" max="2"/>
16      </roles>
17    </group-specification>
18  </structural-specification>
19
```

```
<functional-specification>
  <scheme id="hw_choreography">
    <goal id="show_message">
      <plan operator="sequence">
        <goal id="show_w1"/>
        <goal id="show_w2"/>
      </plan>
    </goal>

    <mission id="mission1"  min="1" max="1"> <goal id="show_w1"/> </mission>
    <mission id="mission2"  min="1" max="1"> <goal id="show_w2"/> </mission>
  </scheme>
</functional-specification>

<normative-specification>
  <norm id="norm1"  type="permission" role="greeter" mission="mission1"/>
  <norm id="norm2"  type="permission" role="greeter" mission="mission2"/>
</normative-specification>

</organisational-specification>
```

Агенти відіграють роль привітання та беруть участь у місіях, щоб показати свої слова (боб показує «Hello», а Аліса - «World»). У кожного агента своя місія/мета/- слово, яке слід показати. Як показано на малюнку 17.4, привітальнику дозволено виконувати будь -яку місію, але ми не хочемо, щоб усі агенти брали участь у всіх місіях, які вони можуть. Щоб вирішити це, кожен агент має віру в місію, яку він повинен взяти на себе. Ці переконання є `my_mission` (місія1) для боба та `my_mission` (місія2) для Аліси.

Рішення взяти участь у місії реалізується за таким планом:

```
1  // when the organization gives me permission to
2  // commit to a mission M in scheme S,
3  // do that if it matches the belief my_mission
4  +permission(A,_,committed(A,M,S),_)
5     :  .my_name(A) &  // the permission is for me
6        my_mission(M)  // my mission is M
7     <- commitMission(M).
```

Символ + у рядку 4 означає "у разі віри ..."; код після `:` це умови щодо того, що агент вважає поточною ситуацією, які необхідні для використання плану; а код після `<--` це «дії» (наприклад, дії, які необхідно виконати, і цілі, яких необхідно досягти). Таким чином, цей план ініціюється додаванням переконання, що агент `A` має дозвіл на виконання місії `M` у схемі `S`. Якщо значення змінної `M` в переконанні агента `my_mission (M)` відповідає дозволеній місії `M`, план застосовується для події та агент виконує дії, спрямовані на виконання місії `M`.

Листові цілі соціальної схеми повинні бути досягнуті агентами, і тому вони мають для цього плани:

```
9   // when I have goal show_w1, create subgoal say(...)
10  +!show_w1  <- !say("Hello").
11  +!show_w2  <- !say("World").

12

13  +!say(M) <- writeMsg(M).
```

Символи `+!` у рядку 10 можна прочитати як "у разі досягнення нової мети ...". Код `!say (...)` в тому ж рядку створює нову підціль. Щодо віри в дозвіл, цілі `show_w` ... виходять від організації. Організація інформує агентів про цілі, які вони мають переслідувати, враховуючи поточний стан виконання схеми та зобов'язання агента. У цьому прикладі всі агенти, які беруть участь в організації, мають плани щодо всіх цілей `show_w`; агенти володіють ноу-хау, щоб показати обидва слова, і те, яке слово вони показують, залежить від місії, яку вони взяли на себе.

Коротко кажучи, агенти мають плани реагувати на події, вироблені організацією (нові дозволи та нові цілі), і їм не потрібно чітко координувати між собою за допомогою спілкування; тобто Бобу більше не потрібно надсилати повідомлення Алісі. Для підтримки координації також не потрібне середовище.

Файл програми для цієї реалізації **`Hello-World`** виглядає наступним чином:

```
1  mas hello_world {
2    agent bob : hwa.asl {
3      focus: room.board
4      roles: greeter in ghw   // initial role for bob
5      beliefs: my_mission(mission1) // initial belief
6    }
7    agent alice : hwa.asl {
8      focus: room.board
9      roles: greeter in ghw
10     beliefs: my_mission(mission2)
11   }
12   workspace room {
13     artifact board : tools.Blackboard
14   }
15   organisation greeting : org1.xml {
16     group ghw : gg {
17       responsible-for: shw
18     }
19     scheme shw : hw_choreography
20   }
21 }
```

Як і раніше, у цьому файлі є записи для агентів та робочих областей, але тепер додано організаційний блок. У рядку 19 організаційна сутність створюється на основі файлу XML, що описує тип груп і схем, доступних в організації. Одна сутність групи створюється у рядку 20 (ідентифіковано **ghw**), а одна сутність схеми створюється у рядку 23 (ідентифікується **shw**). У рядку 21 зазначено, що група **ghw** надає агентів для виконання схеми **shw**. Рядки 5 і 11 відводять роль, яка є більш привабливою для наших агентів у групі **ghw**. Рядки 6 та 12 додають переконання в агентах щодо місій, які вони повинні виконувати.

Виконання файлу програми (`.jcm`) відбувається таким чином:

1. Створюється кімната робочого простору та артефактна дошка.
2. Група **`ghw`** та схема **`shw`** створюються та пов'язуються (відповідальні за).
3. Створюються агенти Боб і Аліса, які приєднуються до кімнати робочого простору.
4. Агентам відводиться роль привітання.
5. Граючи цю роль, вони починають вірити

```
permission(bob,_,committed(bob,mission1,shw),_)
permission(bob,_,committed(bob,mission2,shw),_)
permission(alice,_,committed(alice,mission1,shw),_)
permission(alice,_,committed(alice,mission2,shw),_).
```

6. Додавання цих переконань ініціює їх перший план, і вони виконують свої місії. Зображення загального стану системи

показано на малюнку 17.5.

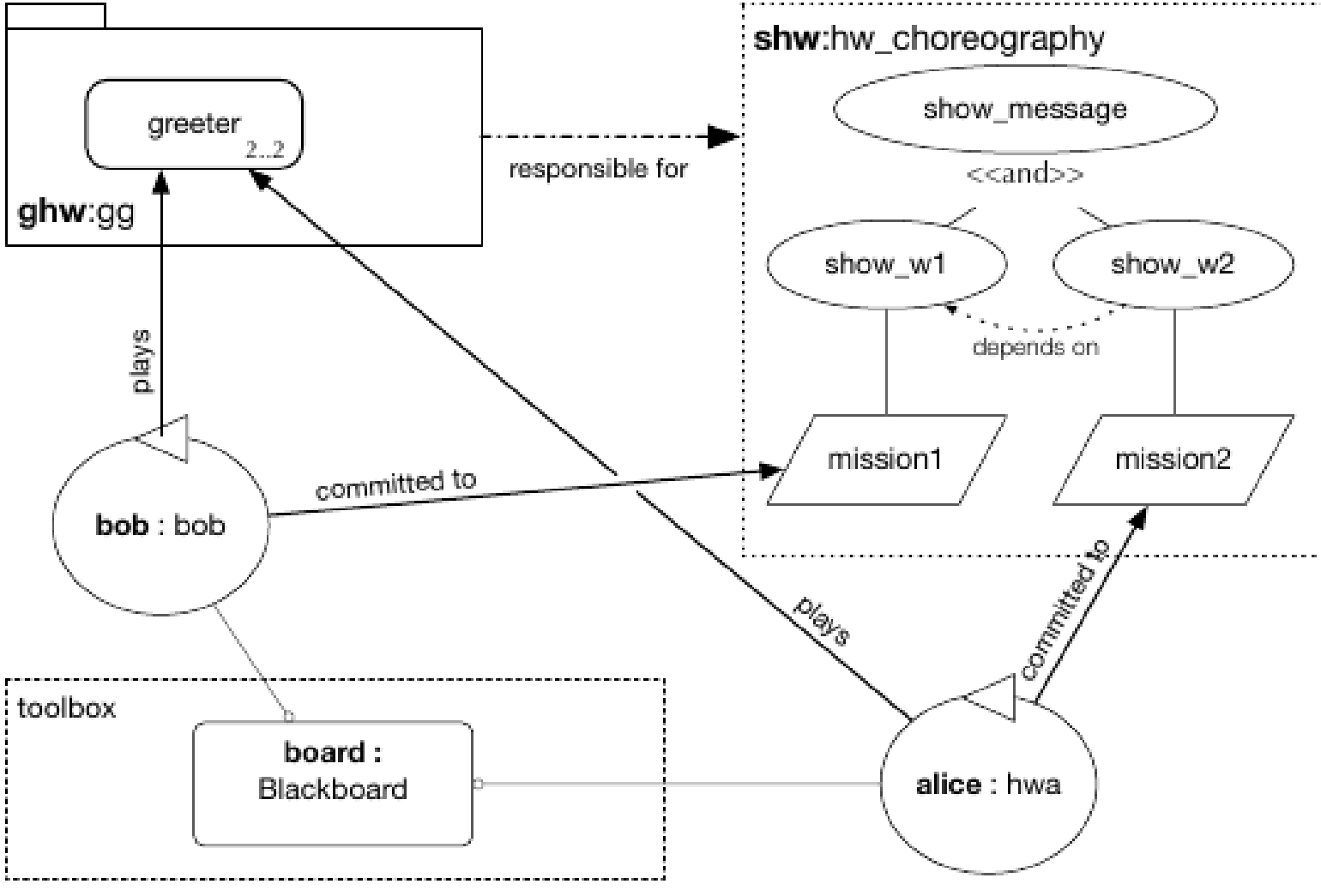
7. Коли агенти виконують свої місії, схема **shw** має достатньо агентів для її виконання, і мета **show_w1** може бути остаточно досягнута.

8. Агент Боб, будучи відданим **місії1**, повідомляється, що мета **show_w1** може бути прийнята, і він це робить; на дошці записується повідомлення **«Hello»**.

9. Потім агенту Алісі повідомляють, щоб він досяг **show_w2**, і він робить це; на дошці написано повідомлення «**World**».

10. Схема закінчена.

Ми можемо помітити узгоджену поведінку: слова завжди показуються у правильному порядку. Більш того, координація реалізується в організації, а не в агентах (в агентах немає коду для координації їх окремих дій, щоб загальна поведінка системи була такою, як очікується).



17.5 Суб’єкти організації Hello-World.

Інсталяція JaCoMo і перші кроки

У нас є різні варіанти початку використання JaCaMo: Eclipse, команди сценарію оболонки, Gradle або Docker.

Параметри сценаріїв Eclipse та оболонки вимагають, щоб ви завантажили та встановили JaCaMo на своєму комп'ютері. Оскільки сценарій оболонки, Gradle та Docker не містять IDE, ви можете або імпортувати проект у робочу область Eclipse (інструкції відображаються сценарієм створення), або скористатися якимось текстовим редактором (ми пропонуємо Atom, див. Нижче).

Eclipse Plugin

Вимоги до програмного забезпечення:

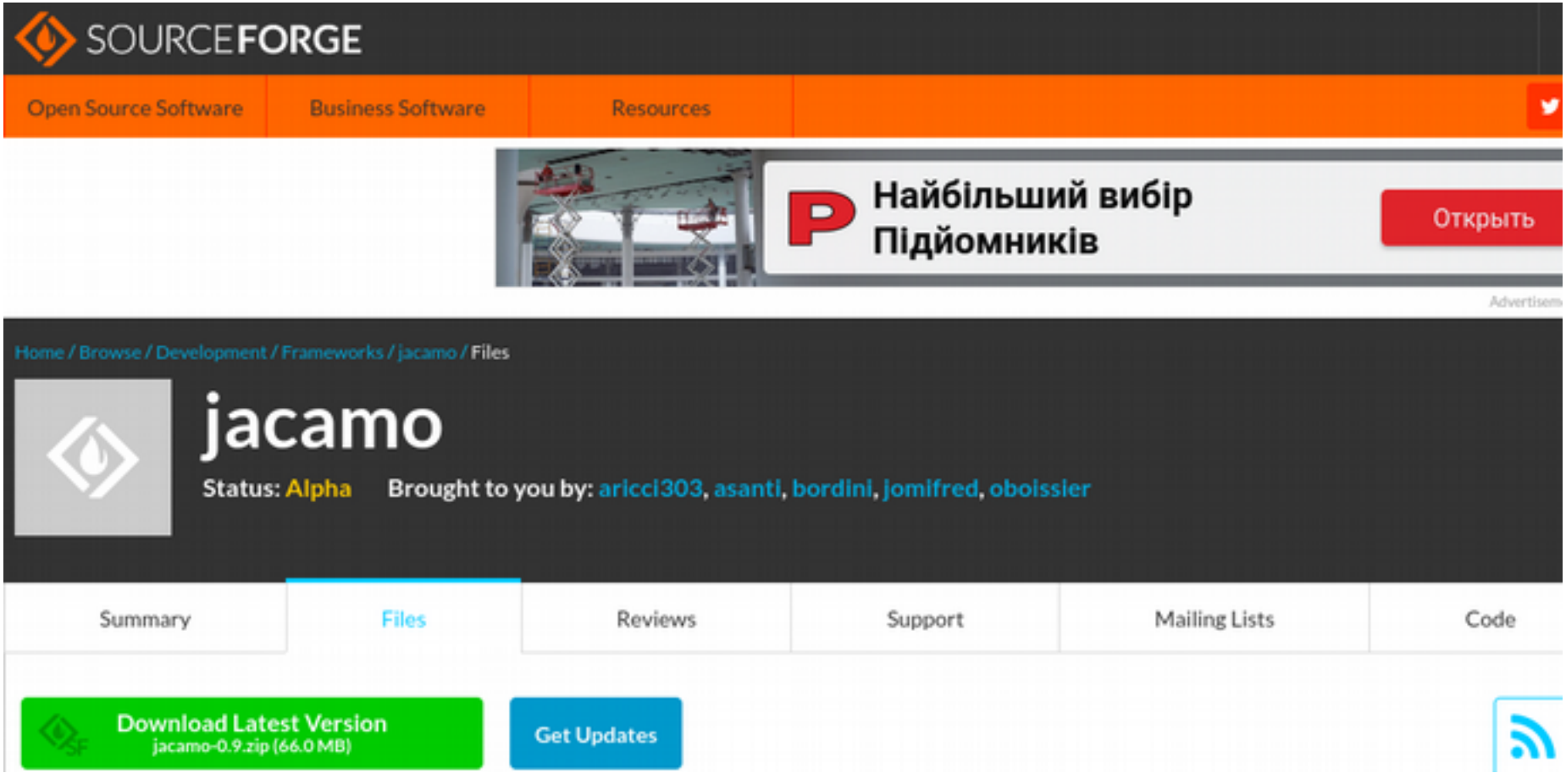
- Java >= 8
- Eclipse Java-EE або Committers >= Photon Release (4.8.0)

Для початку дотримуйтесь інструкцій плагіна JaCaMo eclipse до кроку 12. Кроки 13–17 ілюструють, як створити та запустити нову програму.

Крок 1

Завантажте останню версію JaCaMo за посиланням:

http://sourceforge.net/projects/jacamo/files/



Крок 2

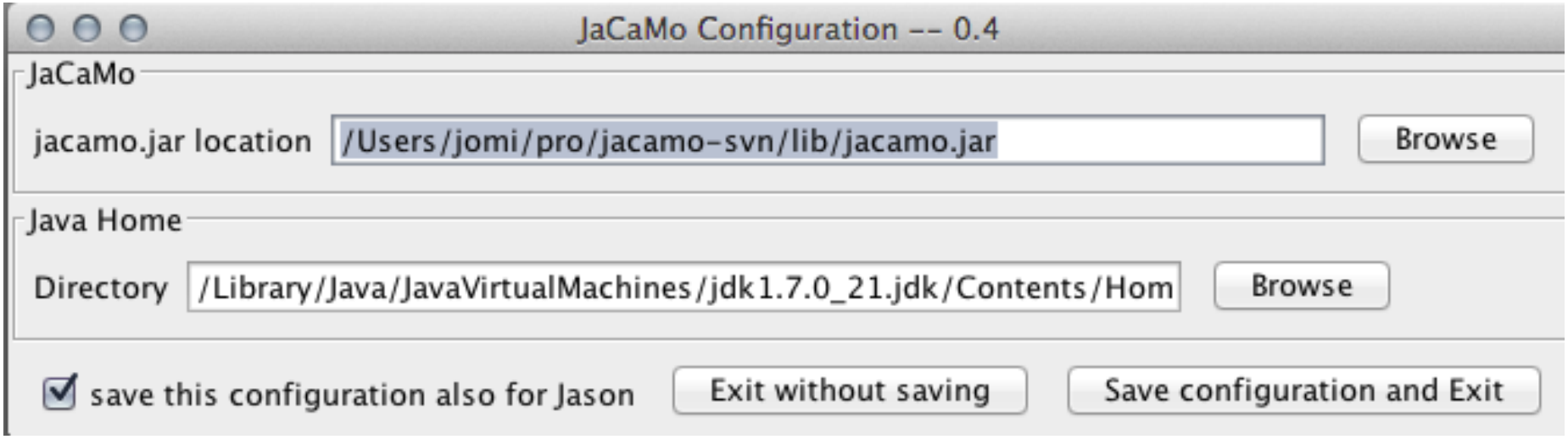
Після завантаження розпакуйте його в будь -якому каталозі вашого комп'ютера.

Крок 3

Навіть якщо раніше ви запускали JaCaMo на своєму комп'ютері, двічі клацніть на ньому файл "`lib/jacamo.jar`". Ви також можете виконати цей файл за допомогою такої команди:

```
java -jar lib/jacamo.jar
```

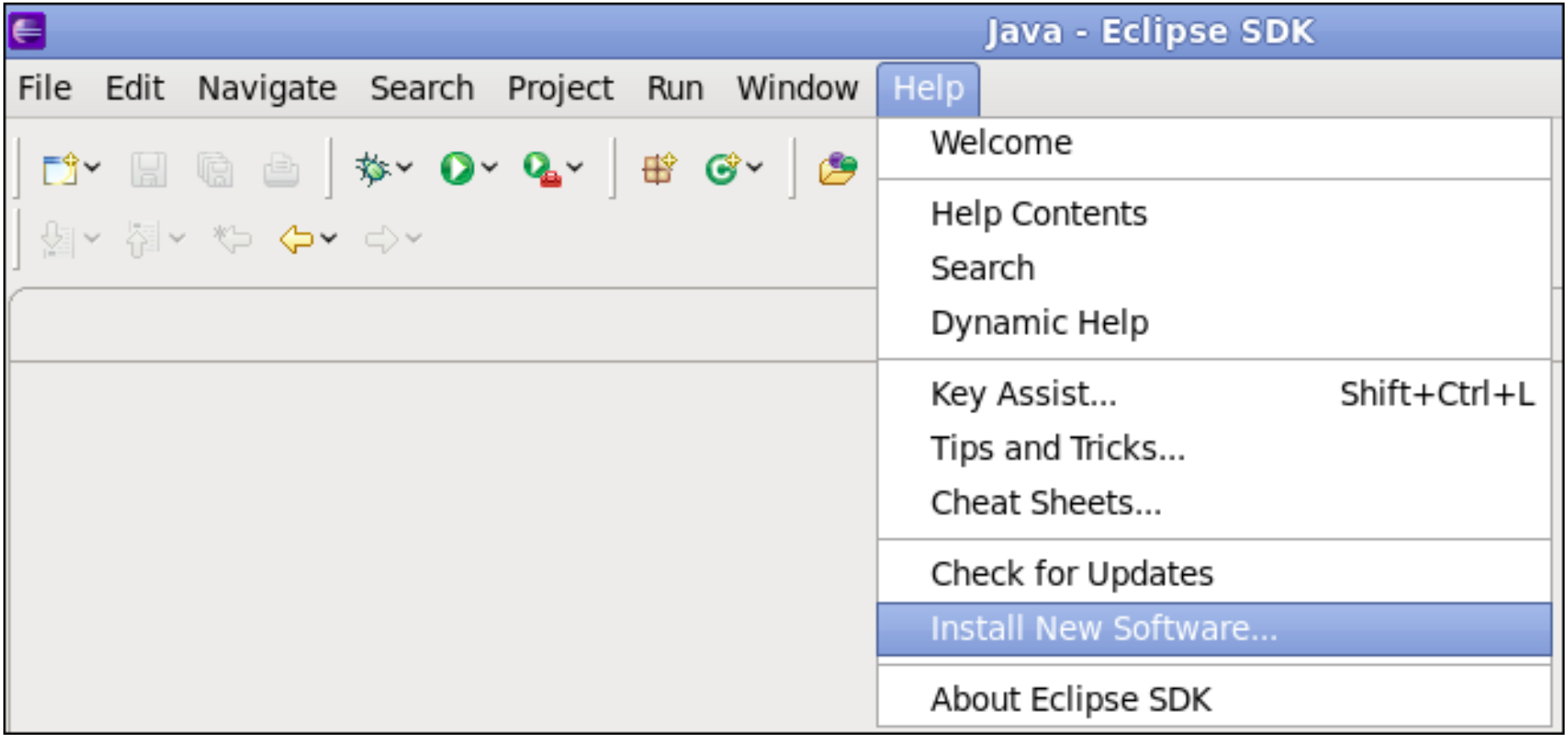
З'явиться наступне вікно. Ми пропонуємо вам змінити лише каталог "Java Home" (зверніть увагу, що він повинен вказувати на встановлення JDK).



*** Якщо ваш доступ до Інтернету проходить через проксі -сервер, натисніть тут http://help.eclipse.org/luna/index.jsp?topic=%2Forg.eclipse.platform.doc.user%2Freference%2Fref-net-preferences.htm , щоб дізнатися, як налаштувати проксі -сервер у вашому Eclipse, перш ніж продовжити встановлення плагіна.

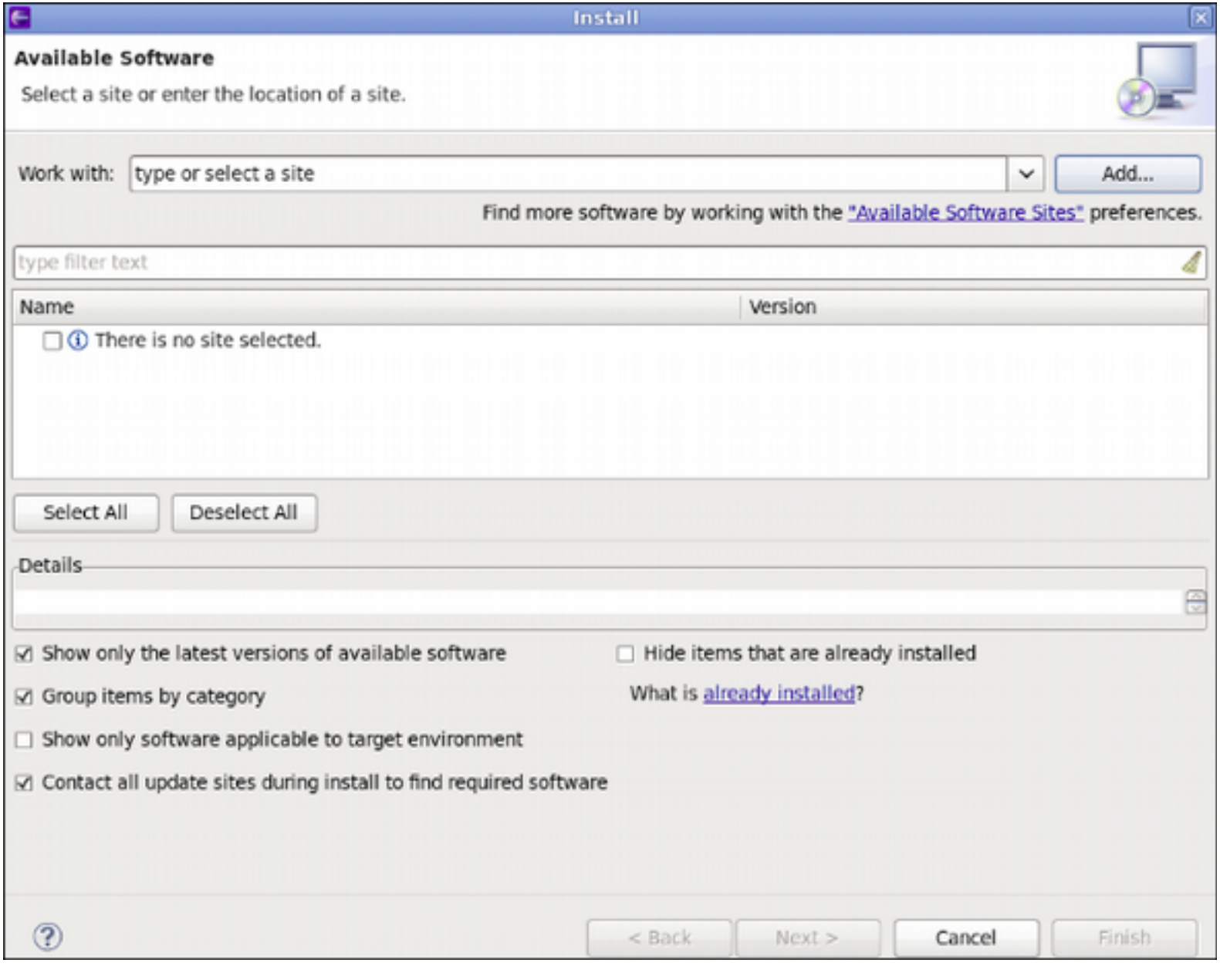
Крок 4

Нарешті, ви можете встановити плагін JaCaMo для Eclipse, відкривши платформу Eclipse і перейшовши до опції "Встановити нове програмне забезпечення ..." у меню "Довідка":


Java - Eclipse SDK
File
Edit
Navigate
Search
Project
Run
Window
Help
Welcome
Help Contents
Search
Dynamic Help
Key Assist... Shift+Ctrl+L
Tips and Tricks...
Cheat Sheets...
Check for Updates
Install New Software...
About Eclipse SDK

Крок 5

Отже, з'явиться наступне вікно.


Install
Available Software
Select a site or enter the location of a site.
Work with:
type or select a site
Add...
Find more software by working with the "Available Software Sites" preferences.
type filter text
Name
Version
There is no site selected.
Select All
Deselect All
Details
Show only the latest versions of available software
Group items by category
Show only software applicable to target environment
Contact all update sites during install to find required software
Hide items that are already installed
What is already installed?
< Back
Next >
Cancel
Finish

Крок 6

Натисніть кнопку «Додати» та заповніть форму, як показано на наступному малюнку. Параметри такі

Name: **`jacamoide`**

Location (for **`JaCaMo >= 0.4`** and **`Eclipse`*** 2020-09 or newer):

http://jacamo.sourceforge.net/eclipseplugin/20x

Location (for **JaCaMo >= 0.4** and **Eclipse*** Juno/Kepler/Luna/Mars/Neon/Oxygen/Photon/2018-2020):

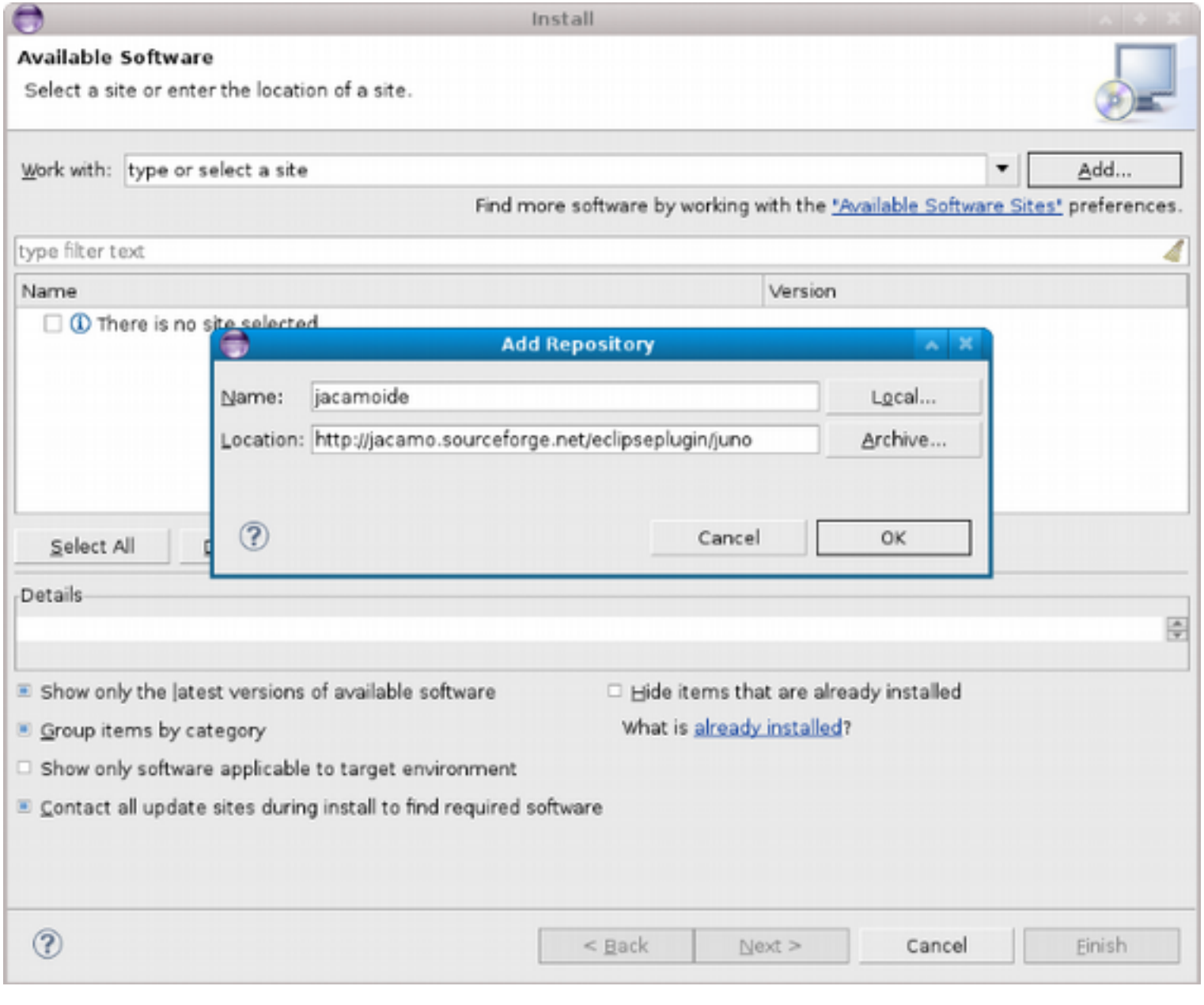
http://jacamo.sourceforge.net/eclipseplugin/juno

Location (for **JaCaMo <= 0.3a** and **Eclipse*** Juno/Kepler/Luna/Mars/Neon/Oxygen/Photon/2018-2020):

http://jacamo.sourceforge.net/eclipseplugin/juno_old

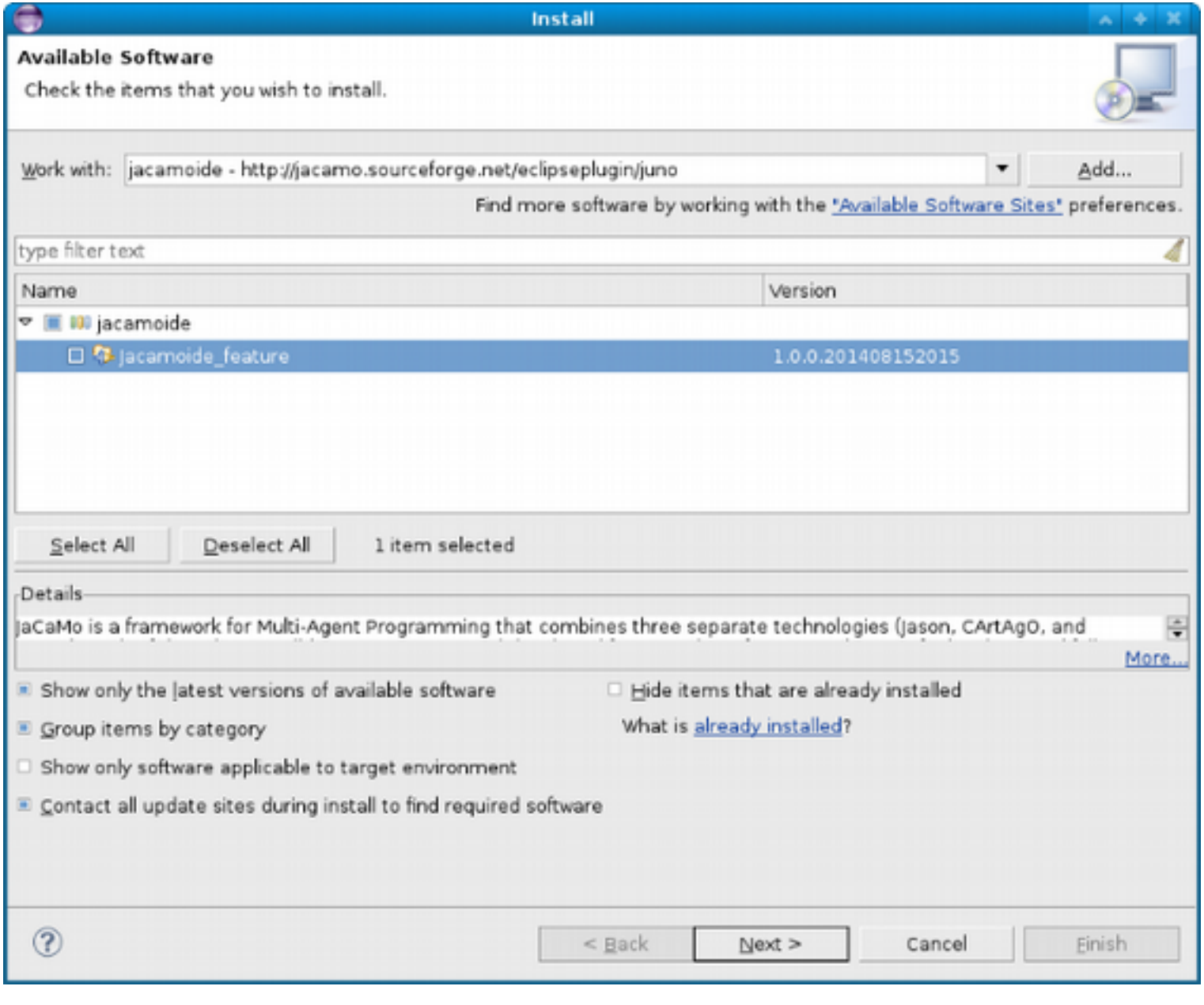
(*) Лише Java-EE або Committers.

Для завершення натисніть кнопку "OK".


Install
Available Software
Select a site or enter the location of a site.
Work with:
type or select a site
Add...
Find more software by working with the "Available Software Sites" preferences.
type filter text
Name
Version
There is no site selected
Add Repository
Name:
jacamoide
Local...
Location:
http://jacamo.sourceforge.net/eclipseplugin/juno
Archive...
Cancel
OK
Select All
Details
Show only the latest versions of available software
Group items by category
Show only software applicable to target environment
Contact all update sites during install to find required software
Hide items that are already installed
What is already installed?
< Back
Next >
Cancel
Finish

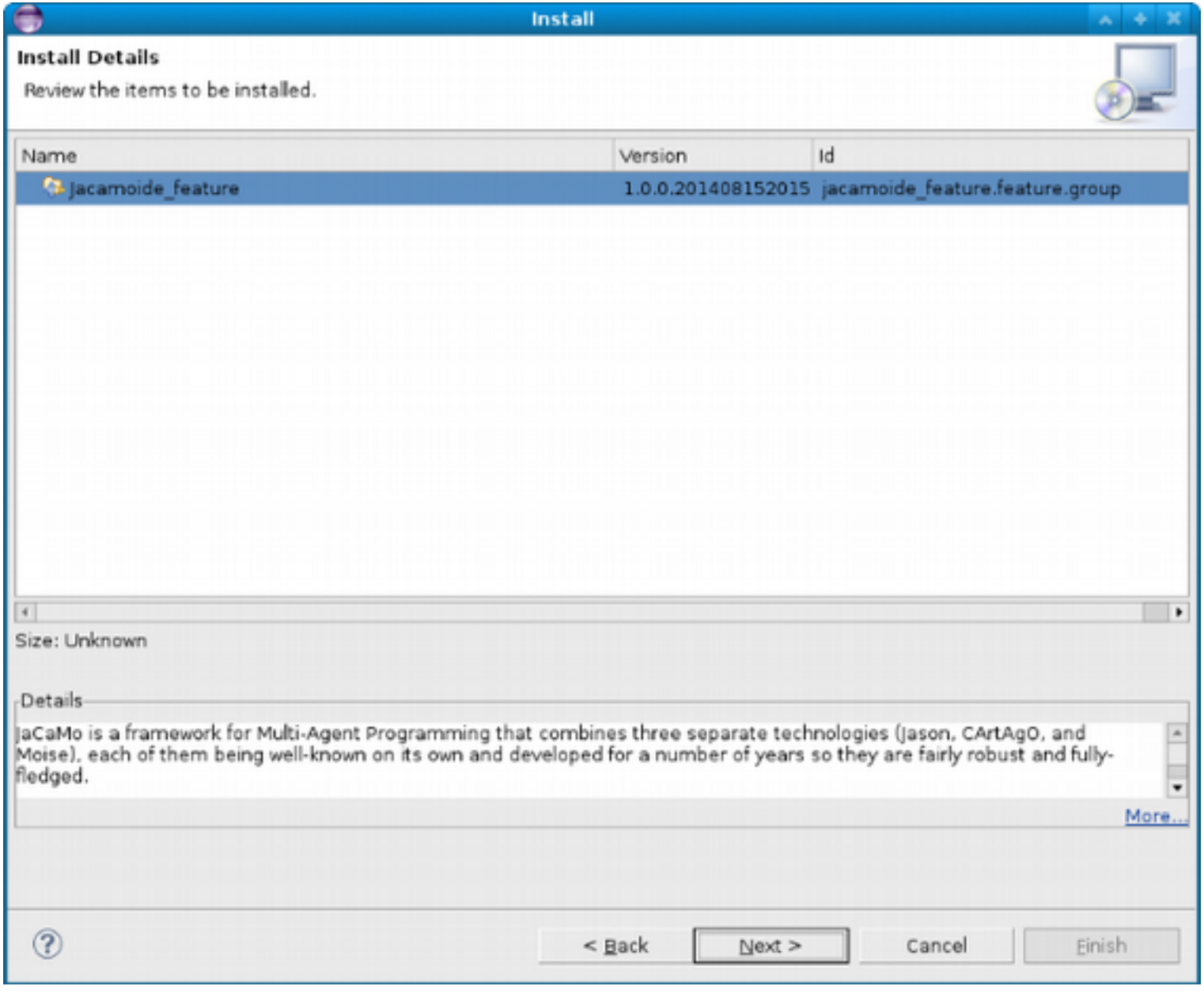
Крок 7

Позначте опцію "jacamoide", а потім натисніть кнопку "наступний". Отже, вам доведеться почекати хвилину, поки Eclipse шукатиме залежності.


Install
Available Software
Check the items that you wish to install.
Work with: jacamoide - http://jacamo.sourceforge.net/eclipseplugin/juno
Add...
Find more software by working with the "Available Software Sites" preferences.
type filter text
Name
Version
jacamoide
jacamoide_feature
1.0.0.201408152015
Select All
Deselect All
1 item selected
Details
JaCaMo is a framework for Multi-Agent Programming that combines three separate technologies (Jason, CArtAgO, and
More...
Show only the latest versions of available software
Hide items that are already installed
Group items by category
What is already installed?
Show only software applicable to target environment
Contact all update sites during install to find required software
< Back
Next >
Cancel
Finish

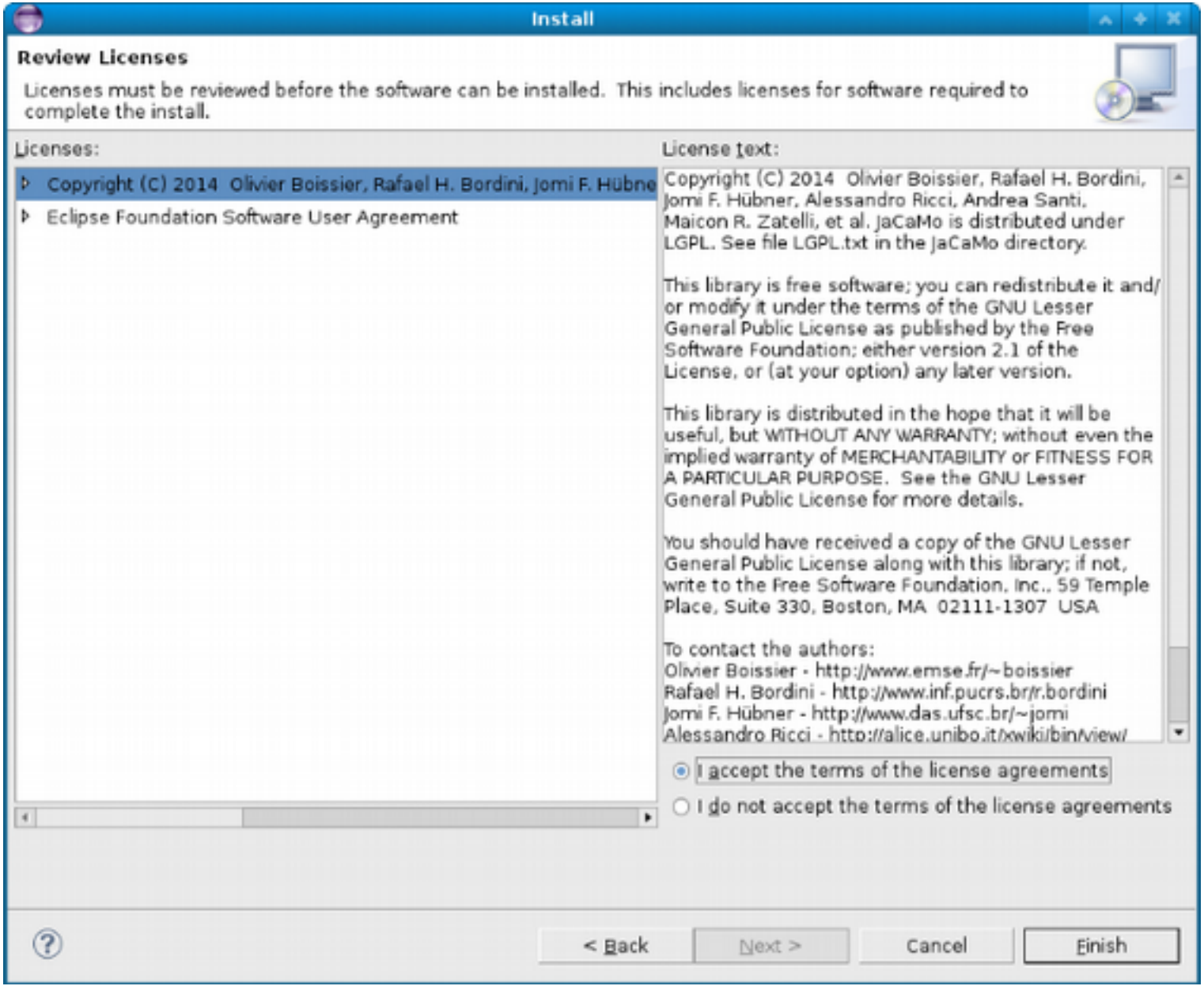
Крок 8

У наступних вікнах просто знову натисніть кнопку «далі».


Install
Install Details
Review the items to be installed.
Name
Version
Id
Jacamoide_feature
1.0.0.201408152015
jacamoide_feature.feature.group
Size: Unknown
Details
JaCaMo is a framework for Multi-Agent Programming that combines three separate technologies (Jason, CArtAgO, and Moise), each of them being well-known on its own and developed for a number of years so they are fairly robust and fully-fledged.
More...
< Back
Next >
Cancel
Finish

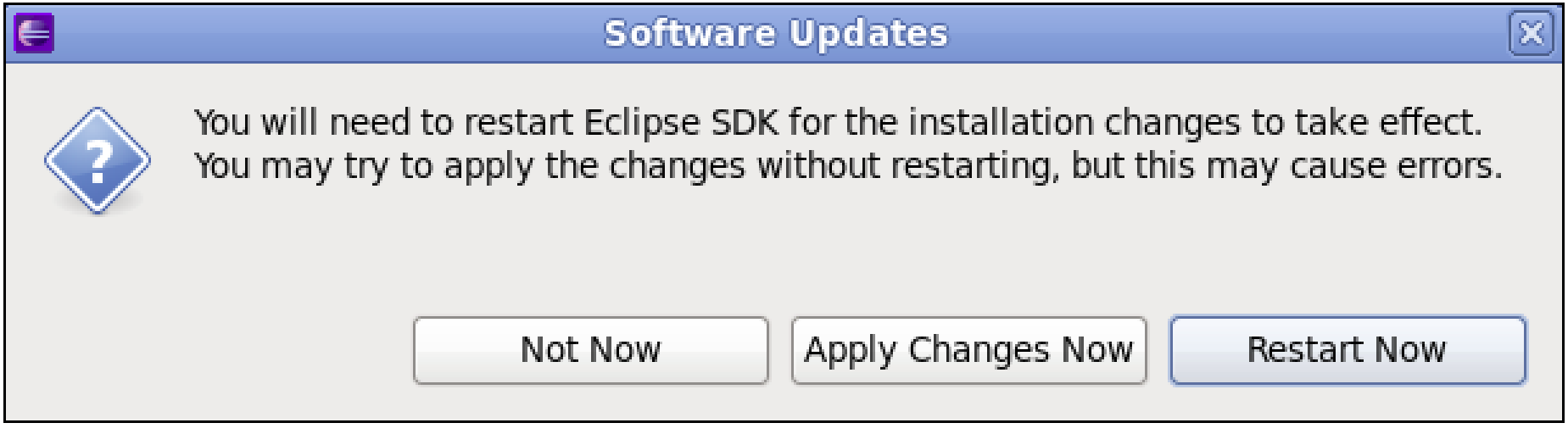
Крок 9

Останнє вікно, яке вам буде показано, стосується ліцензії. Поставте прапорець біля пункту "Я приймаю умови ліцензійних угод", а потім натисніть кнопку "закінчити". Потім інсталяція продовжується, це може зайняти кілька хвилин, тому зачекайте.


Install
Review Licenses
Licenses must be reviewed before the software can be installed. This includes licenses for software required to complete the install.
Licenses:
Copyright (C) 2014 Olivier Boissier, Rafael H. Bordini, Jomi F. Hübne
Eclipse Foundation Software User Agreement
License text:
Copyright (C) 2014 Olivier Boissier, Rafael H. Bordini, Jomi F. Hübner, Alessandro Ricci, Andrea Santi, Maicon R. Zatelli, et al. JaCaMo is distributed under LGPL. See file LGPL.txt in the JaCaMo directory.
This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.
This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.
You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
To contact the authors:
Olivier Boissier - http://www.emse.fr/~boissier
Rafael H. Bordini - http://www.inf.pucrs.br/r.bordini
Jomi F. Hübner - http://www.das.ufsc.br/~jomi
Alessandro Ricci - http://alice.unibo.it/xwiki/bin/view/
I accept the terms of the license agreements
I do not accept the terms of the license agreements
< Back
Next >
Cancel
Finish

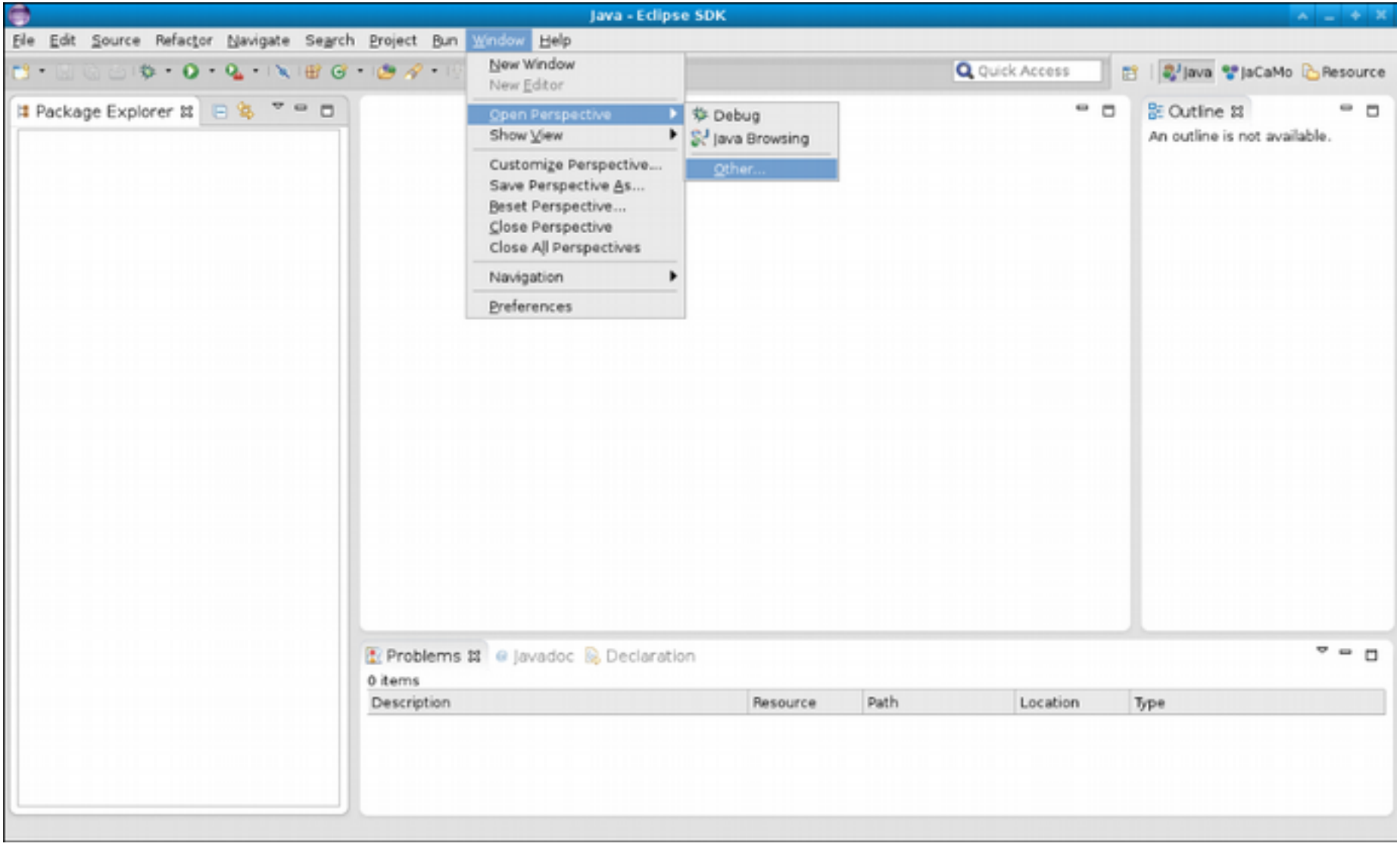
Крок 10

В кінці цього процесу буде показано вікно для завершення установки. Виберіть опцію «Перезапустити зараз».



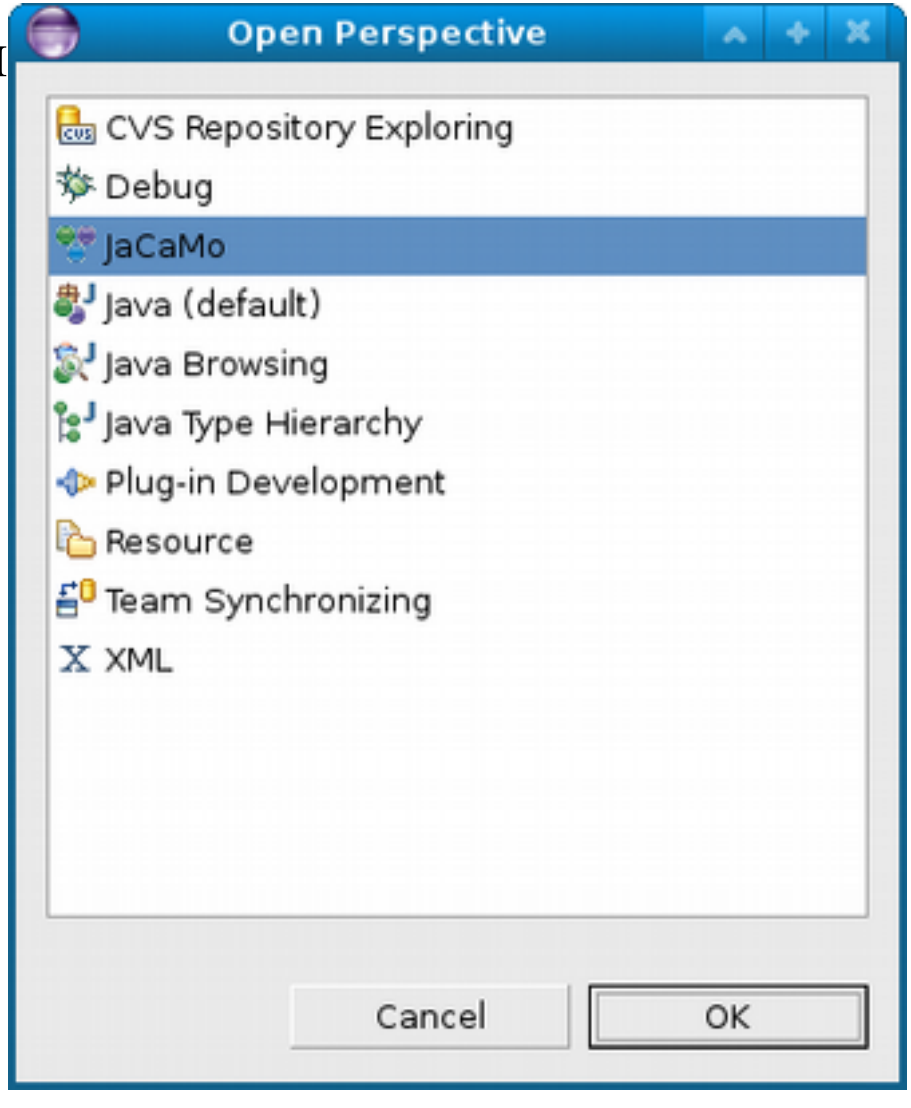
Крок 11

Тепер ви можете відкрити перспективу JaCaMo. Це можна зробити в меню (Вікно> Відкрити перспективу> Інше)


Java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help
New Window
New Editor
Open Perspective
Show View
Customize Perspective...
Save Perspective As...
Reset Perspective...
Close Perspective
Close All Perspectives
Navigation
Preferences
Debug
Java Browsing
Other...
Quick Access
Java
jaCaMo
Resource
Package Explorer
Outline
An outline is not available.
Problems
Javadoc
Declaration
0 items
Description
Resource
Path
Location
Type

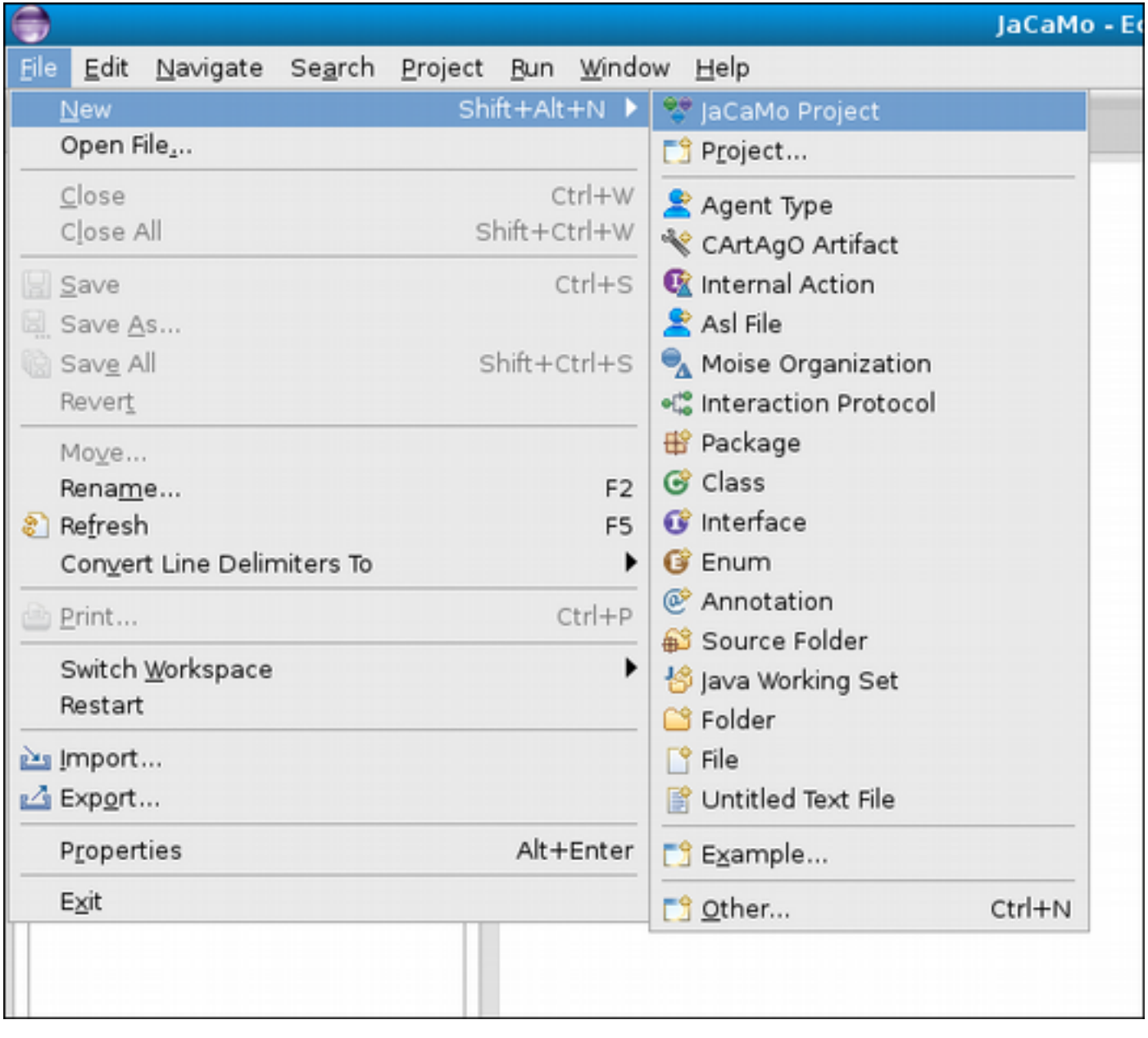
Крок 12

Потім ви можете вибрати JaCaMo і натиснути «OK».



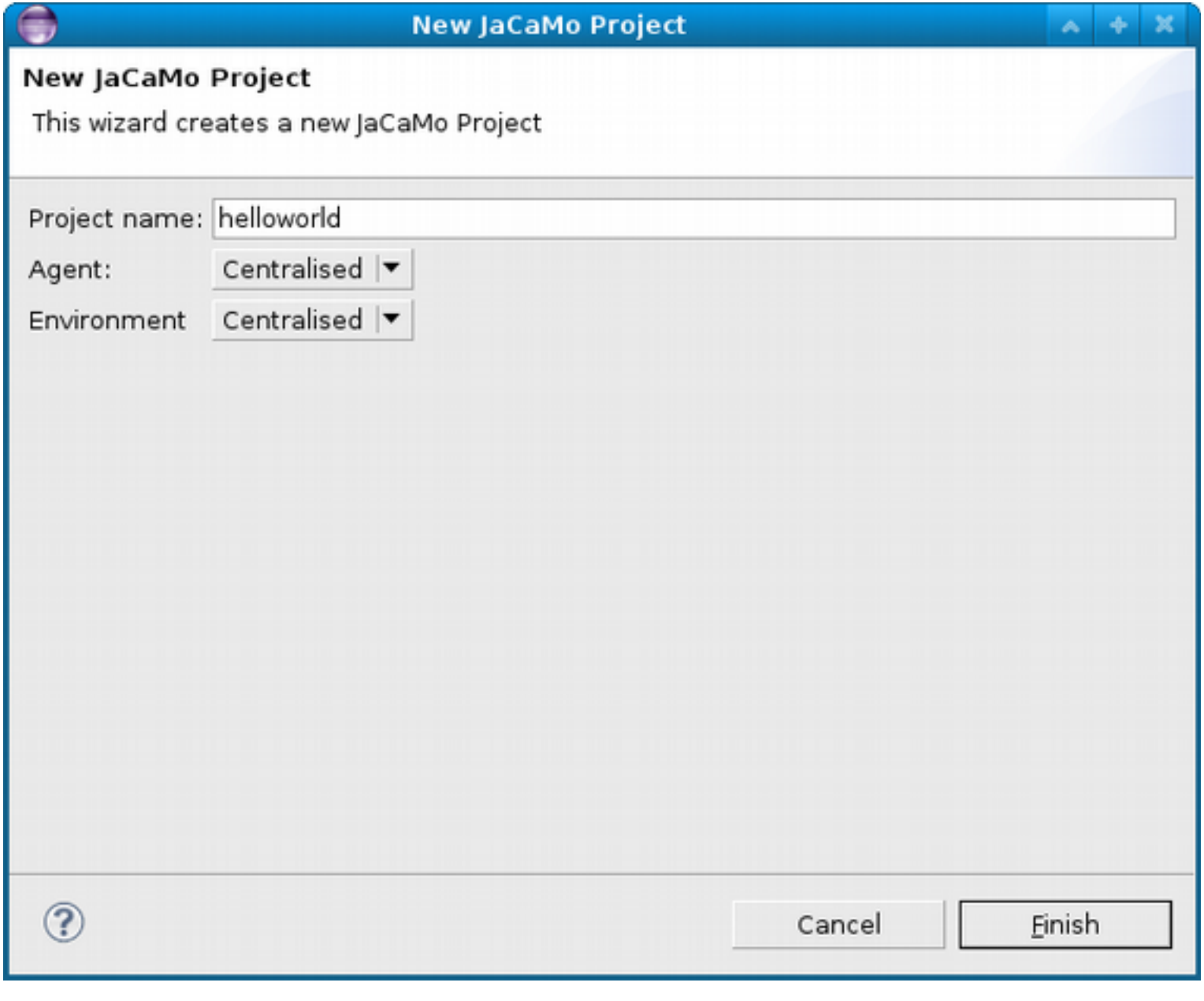
Крок 13

Нарешті, все готово. Для того, щоб перевірити встановлення плагіна, ми пропонуємо створити простий проект hello world. Це можна зробити в меню (Файл> Створити> Проект JaCaMo) або (Файл> Створити> Інше> JaCaMo> Проект JaCaMo).


JaCaMo - E
File
Edit
Navigate
Search
Project
Run
Window
Help
New Shift+Alt+N
Open File...
Close Ctrl+W
Close All Shift+Ctrl+W
Save Ctrl+S
Save As...
Save All Shift+Ctrl+S
Revert
Move...
Rename... F2
Refresh F5
Convert Line Delimiters To
Print... Ctrl+P
Switch Workspace
Restart
Import...
Export...
Properties Alt+Enter
Exit
JaCaMo Project
Project...
Agent Type
CArtAgO Artifact
Internal Action
Asl File
Moise Organization
Interaction Protocol
Package
Class
Interface
Enum
Annotation
Source Folder
Java Working Set
Folder
File
Untitled Text File
Example...
Other... Ctrl+N

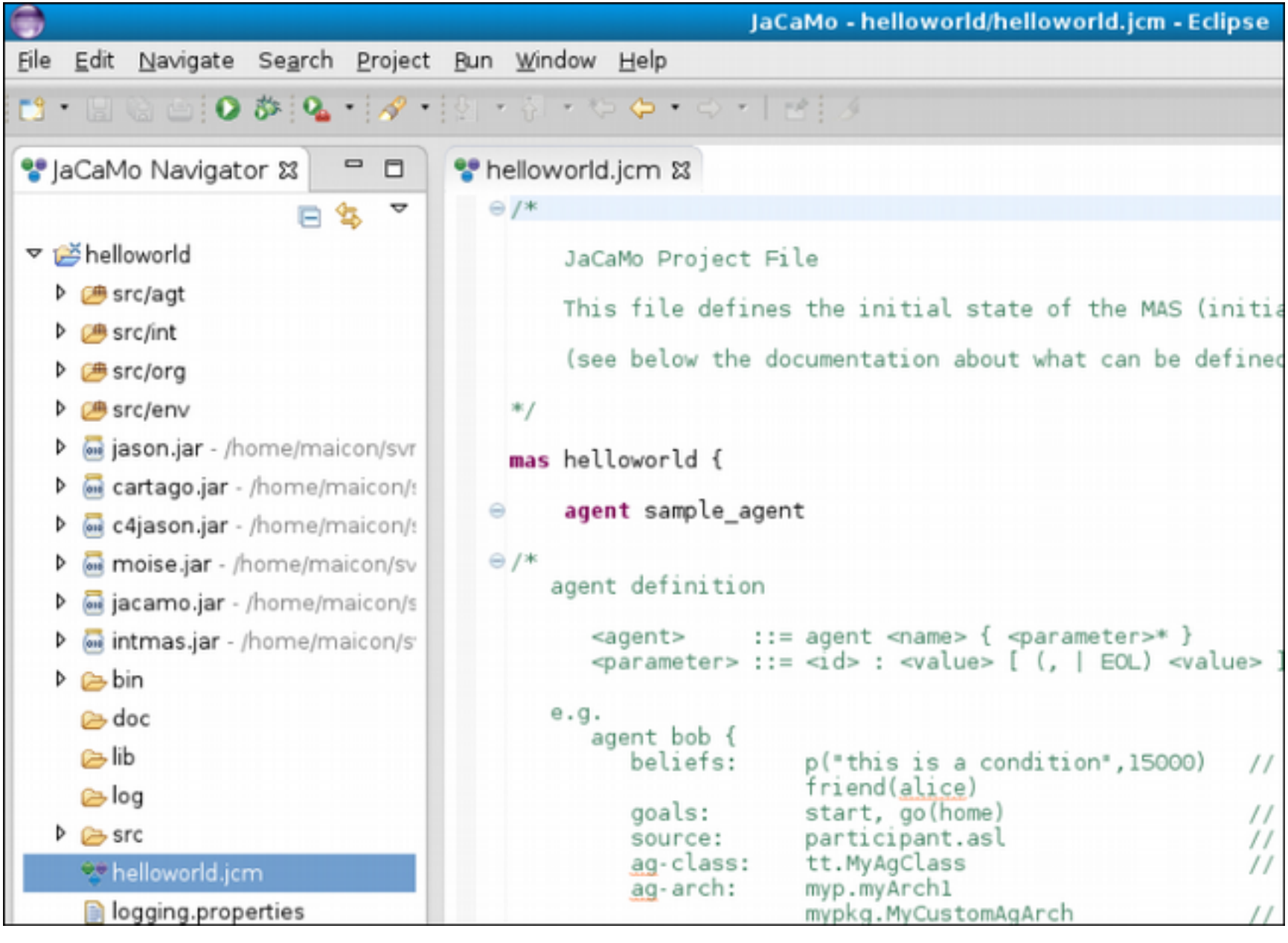
Крок 14

Заповніть поле "Назва проекту" і натисніть кнопку "Готово".


New JaCaMo Project
New JaCaMo Project
This wizard creates a new JaCaMo Project
Project name:
helloworld
Agent:
Centralised
Environment
Centralised
Cancel
Finish

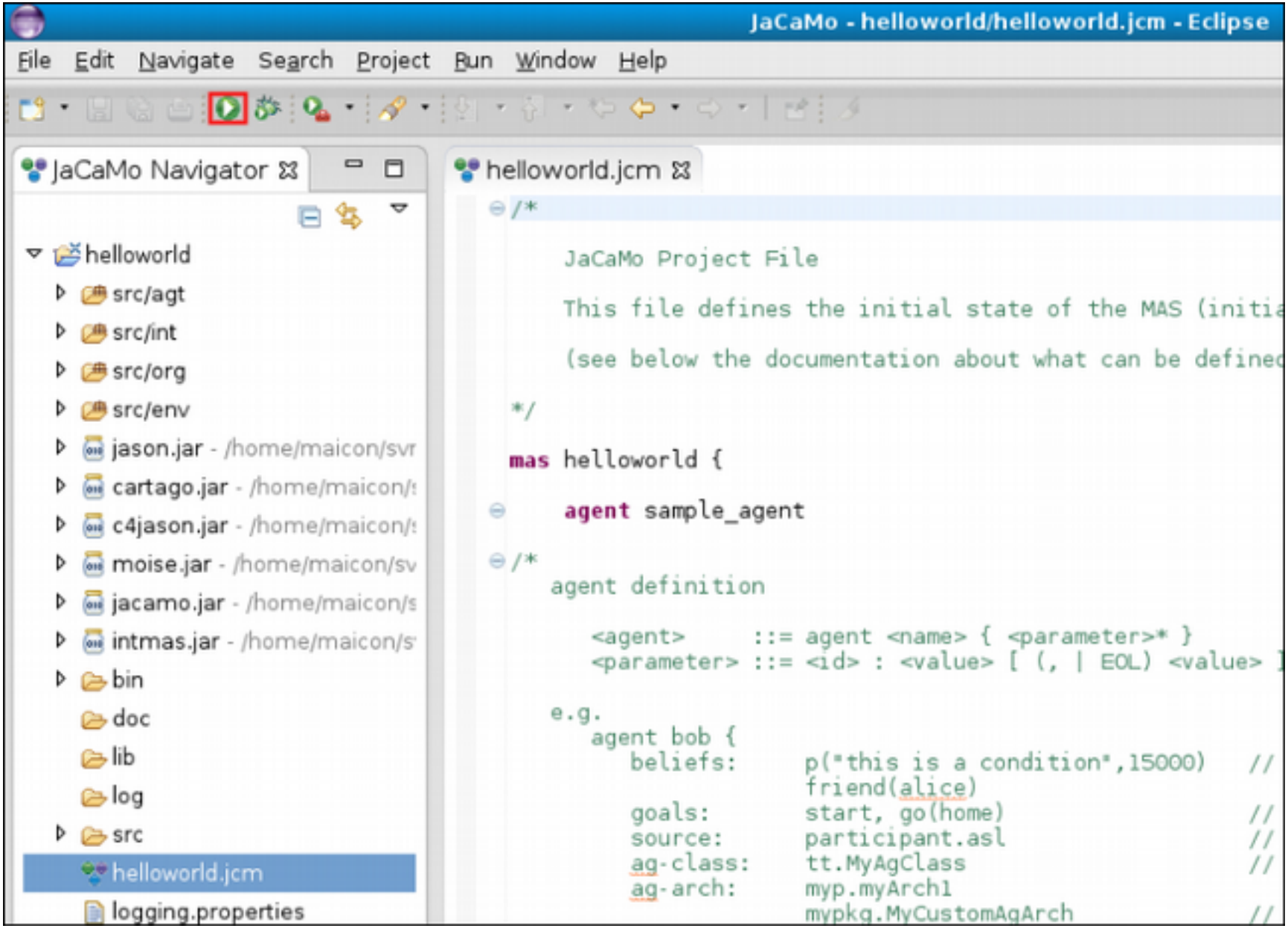
Крок 15

Якщо все в порядку, у вас буде створено ваш перший проект!


JaCaMo - helloworld/helloworld.jcm - Eclipse
File Edit Navigate Search Project Run Window Help
JaCaMo Navigator
helloworld
src/agt
src/int
src/org
src/env
jason.jar - /home/maicon/svr
cartago.jar - /home/maicon/s
c4jason.jar - /home/maicon/s
moise.jar - /home/maicon/sv
jacamo.jar - /home/maicon/s
intmas.jar - /home/maicon/s
bin
doc
lib
log
src
helloworld.jcm
logging.properties
helloworld.jcm
/*
JaCaMo Project File
This file defines the initial state of the MAS (initia
(see below the documentation about what can be define
*/
mas helloworld {
agent sample_agent
/*
agent definition
<agent> ::= agent <name> { <parameter>* }
<parameter> ::= <id> : <value> [(, | EOL) <value>]
e.g.
agent bob {
beliefs: p("this is a condition",15000) //
friend(alice)
goals: start, go(home) //
source: participant.asl //
ag-class: tt.MyAgClass //
ag-arch: myp.myArch1
mypkg.MyCustomAgArch //

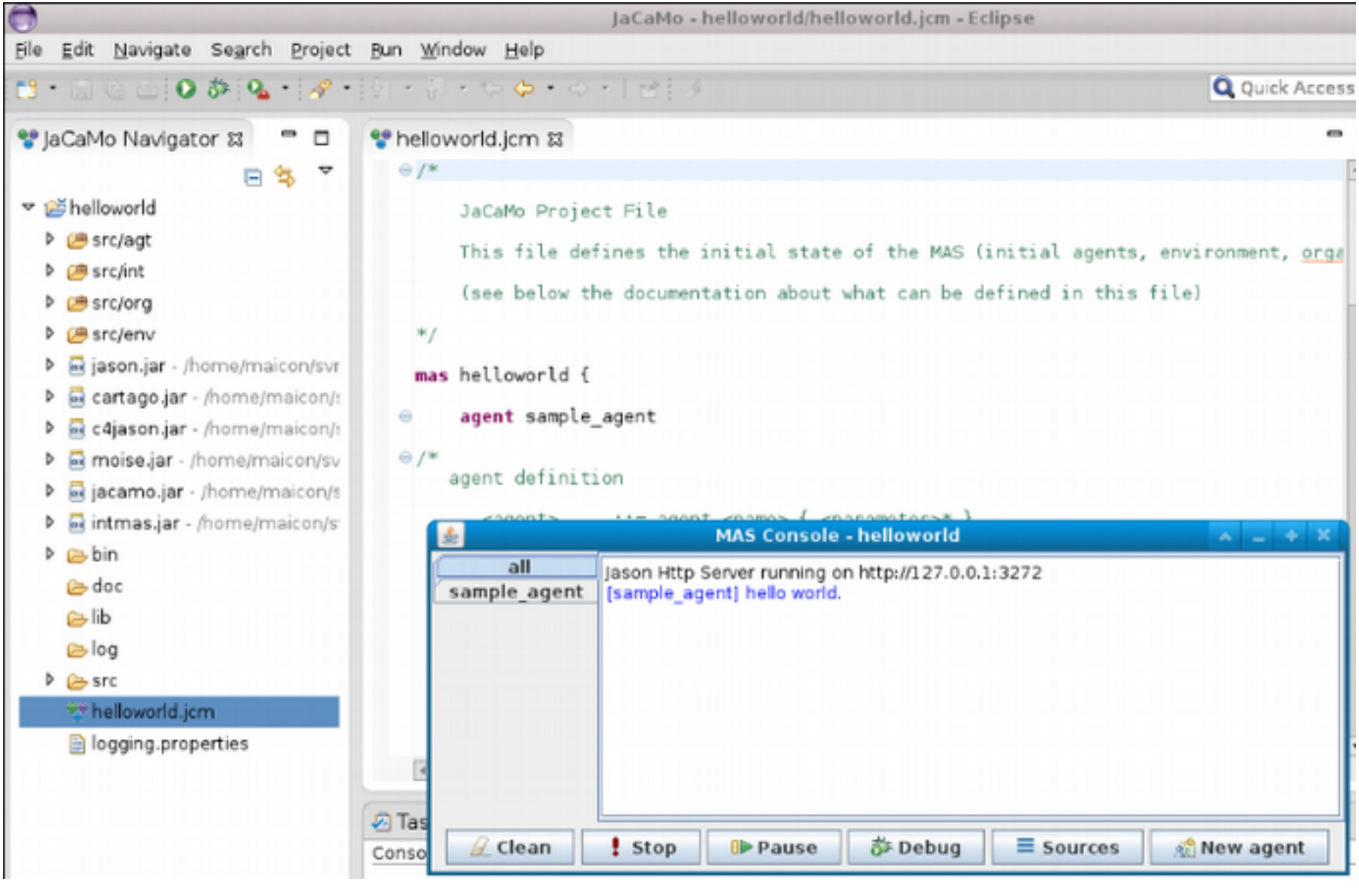
Крок 16

Тепер ви можете запустити програму, натиснувши кнопку Виконати.


JaCaMo - helloworld/helloworld.jcm - Eclipse
File Edit Navigate Search Project Run Window Help
JaCaMo Navigator
helloworld
src/agt
src/int
src/org
src/env
jason.jar - /home/maicon/svr
cartago.jar - /home/maicon/s
c4jason.jar - /home/maicon/s
moise.jar - /home/maicon/sv
jacamo.jar - /home/maicon/s
intmas.jar - /home/maicon/s
bin
doc
lib
log
src
helloworld.jcm
logging.properties
helloworld.jcm
/*
JaCaMo Project File
This file defines the initial state of the MAS (initia
(see below the documentation about what can be define
*/
mas helloworld {
agent sample_agent
/*
agent definition
<agent> ::= agent <name> { <parameter>* }
<parameter> ::= <id> : <value> [(, | EOL) <value>]
e.g.
agent bob {
beliefs: p("this is a condition",15000) //
friend(alice)
goals: start, go(home) //
source: participant.asl //
ag-class: tt.MyAgClass //
ag-arch: myp.myArch1
mypkg.MyCustomAgArch //

Крок 17

Результатом буде повідомлення "привіт світу" на екрані.


JaCaMo - helloworld/helloworld.jcm - Eclipse
File Edit Navigate Search Project Run Window Help
Quick Access
JaCaMo Navigator
helloworld
src/agt
src/int
src/org
src/env
jason.jar - /home/maicon/svr
cartago.jar - /home/maicon/
c4jason.jar - /home/maicon/
moise.jar - /home/maicon/sv
jacamo.jar - /home/maicon/
intmas.jar - /home/maicon/s
bin
doc
lib
log
src
helloworld.jcm
logging.properties
helloworld.jcm
/*
JaCaMo Project File
This file defines the initial state of the MAS (initial agents, environment, orga
(see below the documentation about what can be defined in this file)
*/
mas helloworld {
agent sample_agent
/*
agent definition
MAS Console - helloworld
all
sample_agent
Jason Http Server running on http://127.0.0.1:3272
[sample_agent] hello world.
Clean
Stop
Pause
Debug
Sources
New agent

Gradle

Вимоги:

Java> = 8

Щоб створити новий додаток, виконайте наведені нижче дії.

Примітка. Перший запуск програми триває довше після завантаження JaCaMo.

Unix

```
curl -s -O http://jacamo.sourceforge.net/nps/np1.0.zip
unzip np1.0.zip
./gradlew --console=plain
```

Windows

1. Завантажьте
   http://jacamo.sourceforge.net/nps/np1.0.zip

2. **`Unzip`**

3. Запустіть **`gradlew.bat`**

Порада: Замість **`np1.0.zip`** ви можете використовувати **`npss.zip`** для створення програми JaCaMo на основі поточної версії знімка.

Вам буде запропоновано ввести ідентифікатор програми, після чого будуть показані інструкції щодо її запуску.

Приклад виводу:

```
> Task :run
JaCaMo 1.0 built on Thu Jun 25 21:12:43 BRT 2020

Enter the identification of the new application:
helloworld
Creating path /Users/jomi/tmp/helloworld

You can run your application with:
    cd /Users/jomi/tmp/helloworld
    ./gradlew -q --console=plain

or (if you have JaCaMo scripts installed)
    jacamo /Users/jomi/tmp/helloworld/helloworld.jcm

an Eclipse project can be created using
    'Existing Gradle Project' from Eclipse menu
File/Import
```

```
or
    ./gradlew eclipse
```

Якщо ви хочете запустити програму JaCaMo, яка не була створена з файлом **`build.gradle`**, ви можете завантажити шаблон тут https://raw.githubusercontent.com/jacamo-lang/jacamo/master/src/main/resources/templates/build.gradle і замінити

`<VERSION>` by the required JaCaMo release (e.g. `1.0`)

`<PROJECT-FILE>` by your .jcm file (e.g. `hello.jcm`).

Команди скрипту Shell

Вимоги:

Java> = 8

Щоб використовувати сценарії для створення та запуску програм JaCaMo, ви можете завантажити JaCaMo звідси або клонувати з GitHub. Щоб налаштувати ці сценарії:

Переконайтеся, що у вас правильно встановлено JACAMO_HOME (ця змінна вказує на каталог із бібліотеками підкаталогів та сценаріями JaCaMo), а потім налаштуйте **`PATH`**.

Якщо ви завантажили та розпакували JaCaMo у локальну папку, скажімо `/home/bob/jacamo`, налаштування виконується за допомогою таких команд:

```
export JACAMO_HOME=/home/bob/jacamo
export PATH=$JACAMO_HOME/scripts:$PATH
```

у випадку, якщо ви вирішили клонувати JaCaMo з GitHub:

```
git clone https://github.com/jacamo-lang/jacamo.git
cd jacamo
./gradlew config
export JACAMO_HOME=`pwd`/build
export PATH=$JACAMO_HOME/scripts:$PATH
```

Переконайтеся, що у вас JAVA_HOME правильно встановлено (ця змінна вказує на каталог Java Development Kit (JDK)), наприклад:

```
export JAVA_HOME=/Library/Java/JavaVirtualMachines/jdk-12.0.2.jdk/Contents/Home
```

Доступні сценарії:

1. Створіть новий додаток, ідентифікований **helloworld** за допомогою

```
jacamo-new-project ../somewhere/helloworld
```

Приклади агента, артефакту та організації розміщуються в папці src. Якщо ім'я програми не повідомляється, користувача просять надати його. Другий необов'язковий аргумент **--console** і створює додаток, де повідомлення друкуються в консолі замість відкриття графічного інтерфейсу консолі MAS.

2. Скомпілюйте та запустіть додаток за допомогою

```
jacamo ../somewhere/helloworld/helloworld.jcm
```

3. Запустіть додаток за допомогою

```
jacamo-run ../somewhere/helloworld/helloworld.jcm
```

4. Щоб створити єдиний файл `jar` з усіма ресурсами для запуску програми:

```
jacamo-jar helloworld.jcm
```

яким можна керувати

```
java -jar helloworld.jar
```

Script	Arguments	Examples

jacamo-new-project	[*application name*]
	
	[, `--console`]
		`jacamo-new-project`
		
		`jacamo-new-project helloworld`
		
		`jacamo-new-project helloworld --console`

jacamo	*application name*	`jacamo helloworld.jcm`
jacamo-run	*application name*	`jacamo-run helloworld.jcm`
jacamo-jar	*application name*	`jacamo-jar helloworld.jcm`

Docker

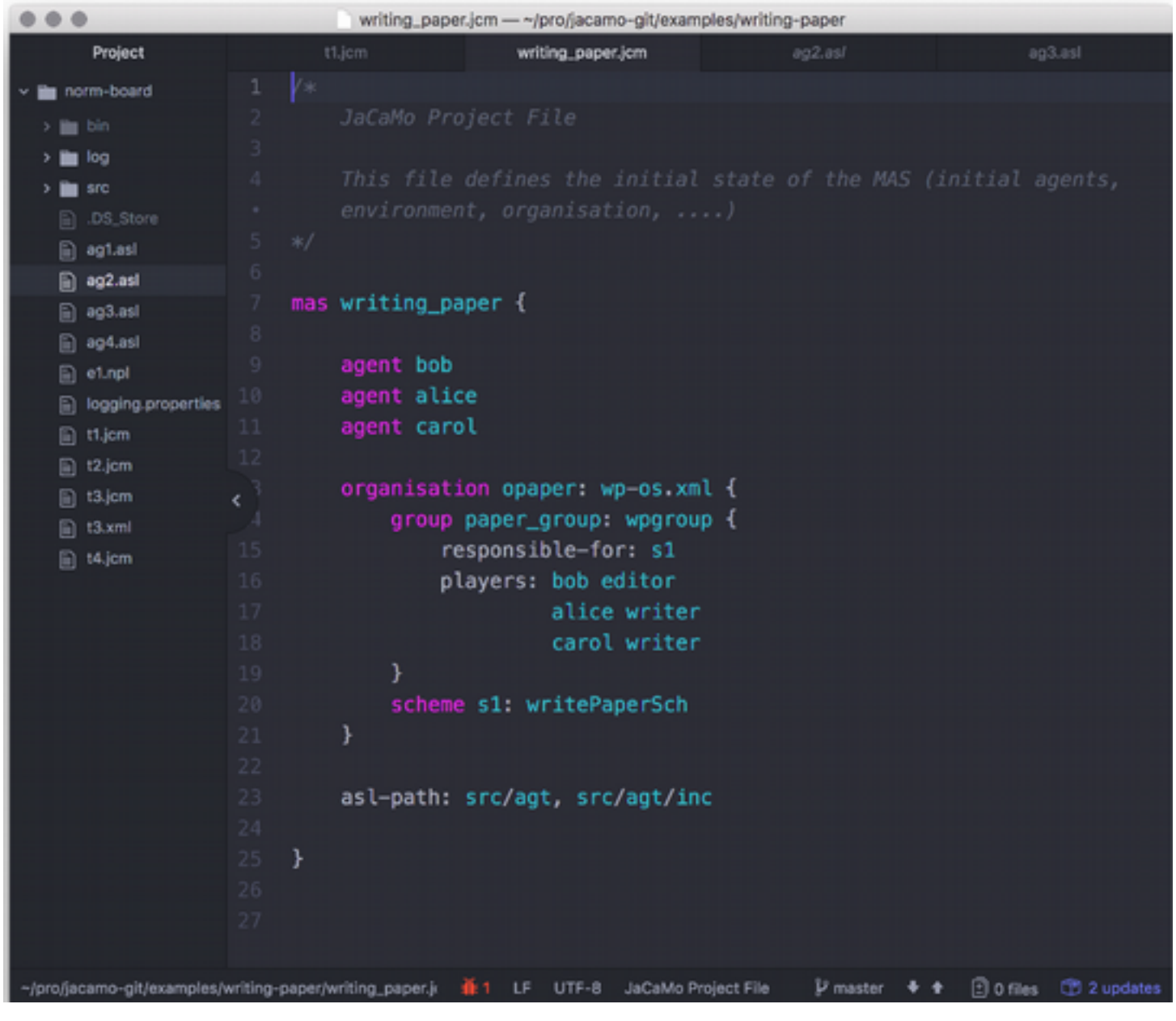
Вимоги:

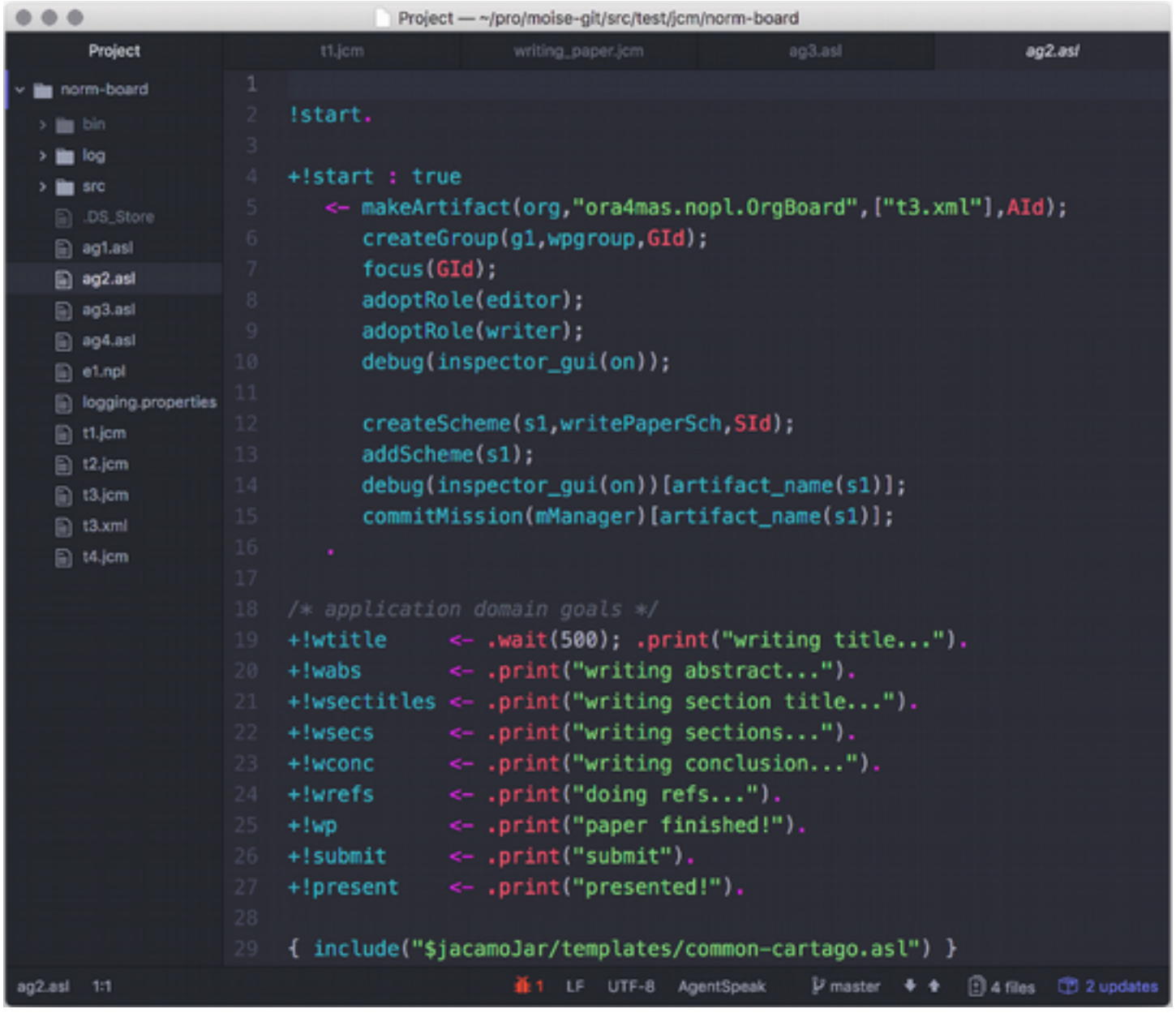
- Docker

Інструкції щодо створення та запуску програми JaCaMo за допомогою зображень Docker див. на сторінці JaCaMo-Docker - https://github.com/jacamo-lang/docker .

Atom

Для IDE, легшої за Eclipse, ми рекомендуємо Atom - **https://atom.io/** . Наступні пакети забезпечують підсвічування синтаксису для проектів AgentSpeak (мова Jason) та JCM: language-jcm **https://atom.io/packages/language-jcm** та language-agentspeak **https://atom.io/packages/language-agentspeak** .


writing_paper.jcm — ~/pro/jacamo-git/examples/writing-paper
Project
norm-board
bin
log
src
.DS_Store
ag1.asl
ag2.asl
ag3.asl
ag4.asl
e1.npl
logging.properties
t1.jcm
t2.jcm
t3.jcm
t3.xml
t4.jcm
t1.jcm
writing_paper.jcm
ag2.asl
ag3.asl
/*
JaCaMo Project File
This file defines the initial state of the MAS (initial agents,
environment, organisation, ...)
*/
mas writing_paper {
agent bob
agent alice
agent carol
organisation opaper: wp-os.xml {
group paper_group: wpgroup {
responsible-for: s1
players: bob editor
alice writer
carol writer
}
scheme s1: writePaperSch
}
asl-path: src/agt, src/agt/inc
}
~/pro/jacamo-git/examples/writing-paper/writing_paper.j
1
LF
UTF-8
JaCaMo Project File
master
0 files
2 updates


Project — ~/pro/moise-git/src/test/jcm/norm-board
Project
t1.jcm
writing_paper.jcm
ag3.asl
ag2.asl
norm-board
bin
log
src
.DS_Store
ag1.asl
ag2.asl
ag3.asl
ag4.asl
e1.npl
logging.properties
t1.jcm
t2.jcm
t3.jcm
t3.xml
t4.jcm
!start.
+!start : true
<- makeArtifact(org,"ora4mas.nopl.OrgBoard",["t3.xml"],AId);
createGroup(g1,wpgroup,GId);
focus(GId);
adoptRole(editor);
adoptRole(writer);
debug(inspector_gui(on));
createScheme(s1,writePaperSch,SId);
addScheme(s1);
debug(inspector_gui(on))[artifact_name(s1)];
commitMission(mManager)[artifact_name(s1)];
.
/* application domain goals */
+!wtitle <- .wait(500); .print("writing title...").
+!wabs <- .print("writing abstract...").
+!wsectitles <- .print("writing section title...").
+!wsecs <- .print("writing sections...").
+!wconc <- .print("writing conclusion...").
+!wrefs <- .print("doing refs...").
+!wp <- .print("paper finished!").
+!submit <- .print("submit").
+!present <- .print("presented!").
{ include("$jacamoJar/templates/common-cartago.asl") }
ag2.asl 1:1
1
LF
UTF-8
AgentSpeak
master
4 files
2 updates

Завдання:

1. Встановити інструментарії для роботи з JaCaMo:

Eclipse,

команди сценарію Shell,

Gradle,

Docker,

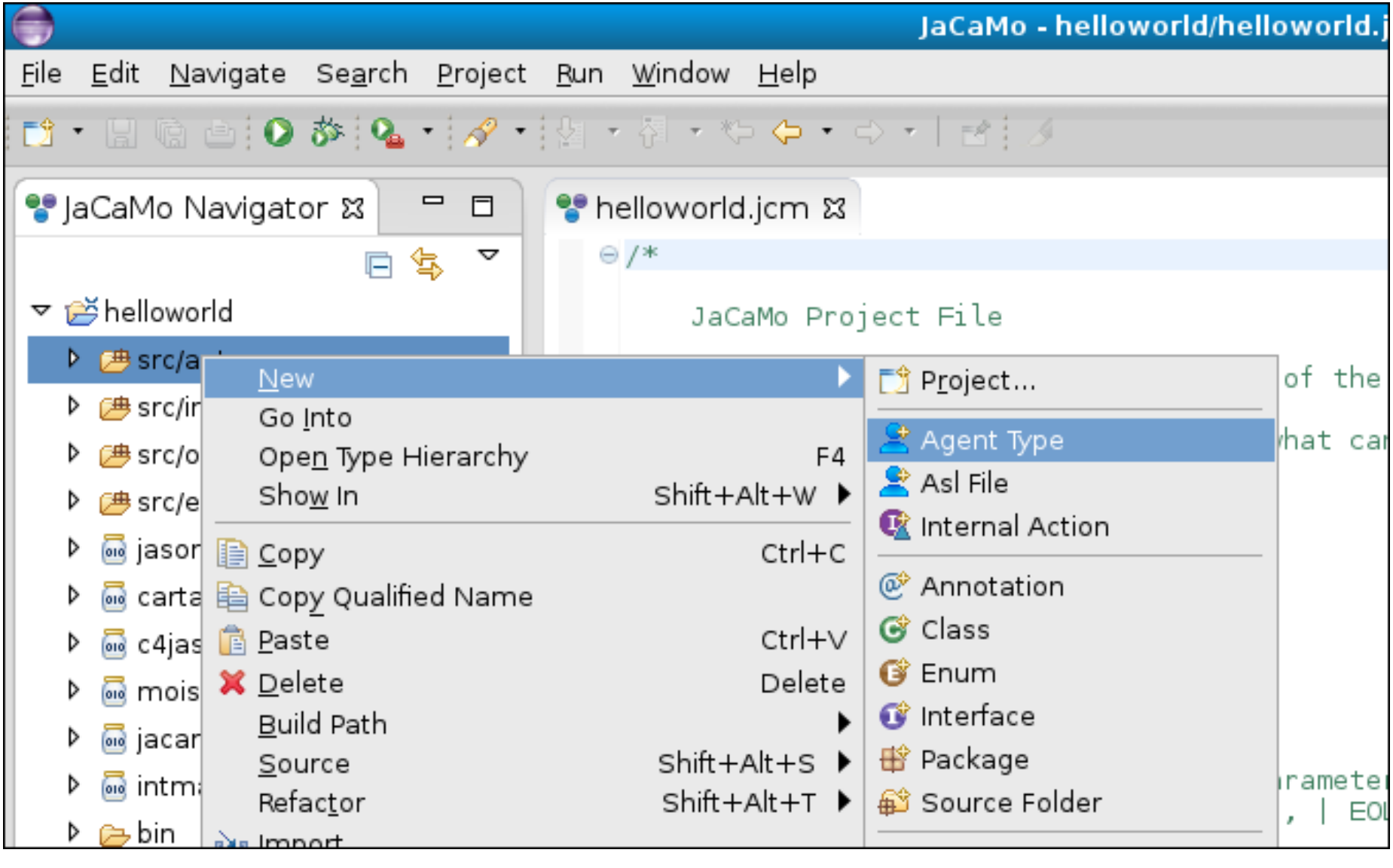
Atom.

2. Підготуйте звіт з скріншотами про виконані кроки.

Створення агентів на платформі JaCaMo

Крок 1

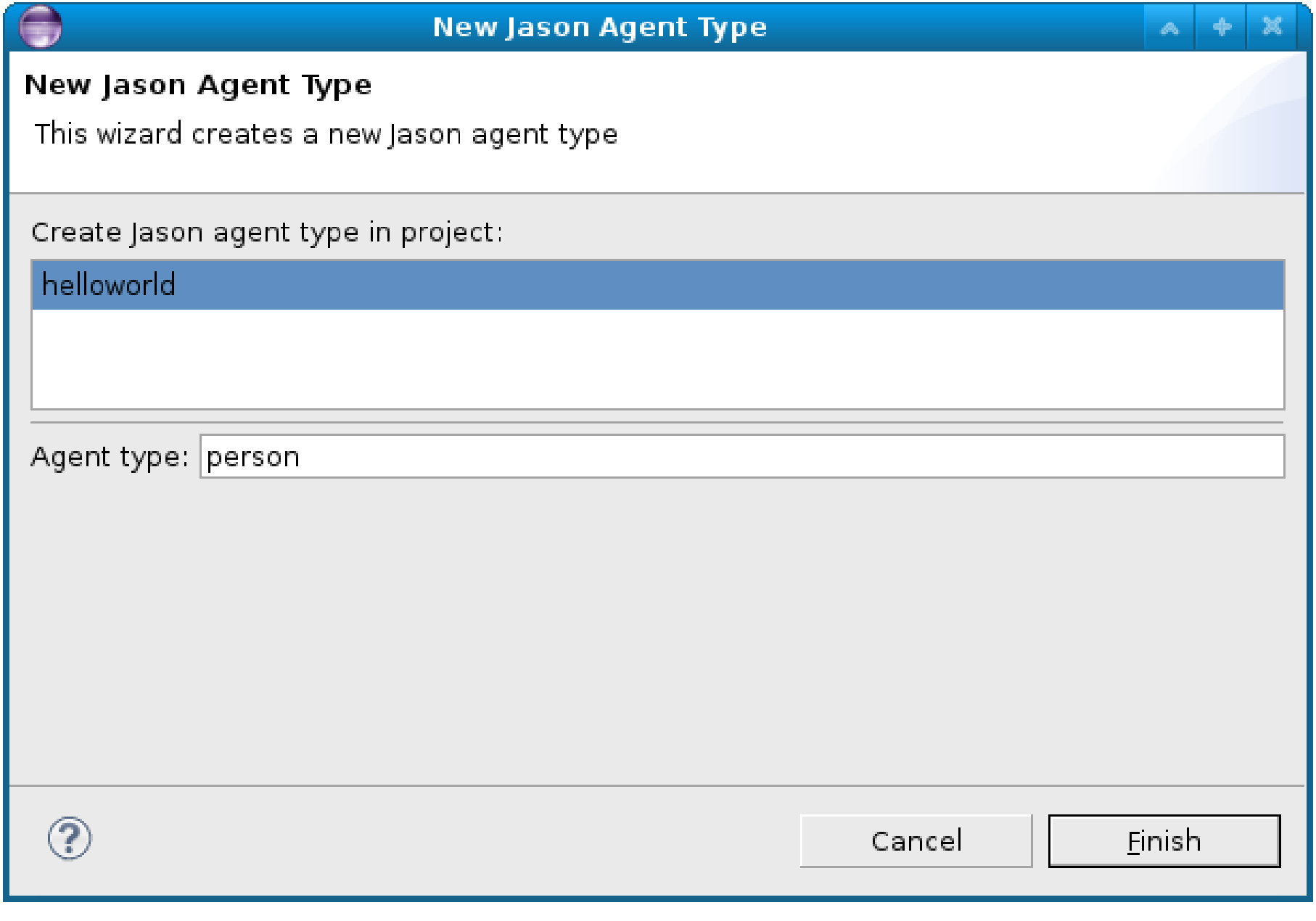
Натисніть правою кнопкою на вихідну папку з назвою "**`src/agt`**" і перейдіть до опції **Створити> Тип агента**.


JaCaMo - helloworld/helloworld.j
File Edit Navigate Search Project Run Window Help
JaCaMo Navigator
helloworld.jcm
/*
JaCaMo Project File
helloworld
src/a
src/ir
src/o
src/e
jason
carta
c4jas
mois
jacar
intma
bin
New
Go Into
Open Type Hierarchy F4
Show In Shift+Alt+W
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Delete Delete
Build Path
Source Shift+Alt+S
Refactor Shift+Alt+T
Import
Project...
Agent Type
Asl File
Internal Action
Annotation
Class
Enum
Interface
Package
Source Folder
of the
hat car

Крок 2

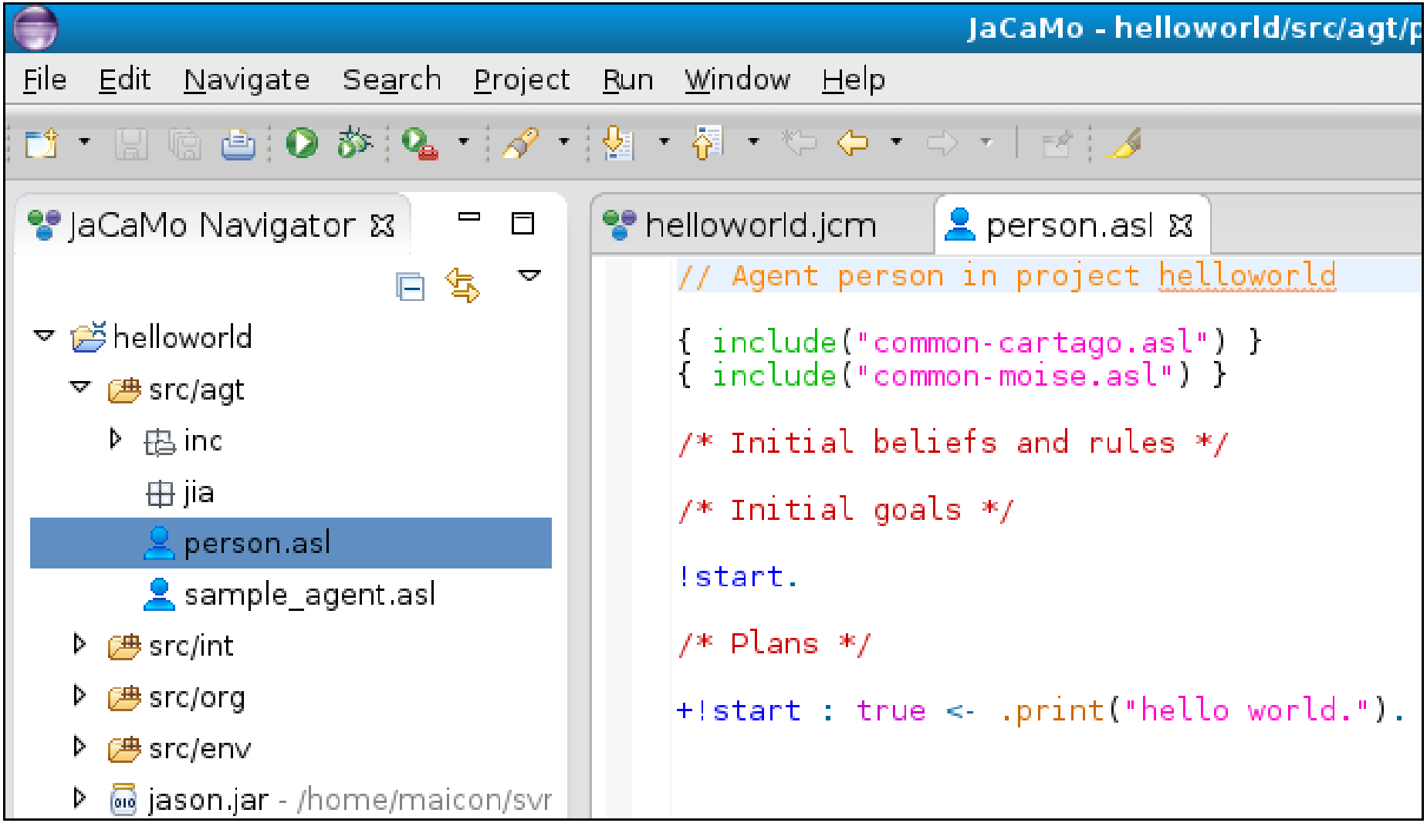
Заповніть форму. Єдине обов’язкове поле - це ім’я типу агента.

Після цього натисніть кнопку «Готово».


New Jason Agent Type
New Jason Agent Type
This wizard creates a new Jason agent type
Create Jason agent type in project:
helloworld
Agent type:
person
Cancel
Finish

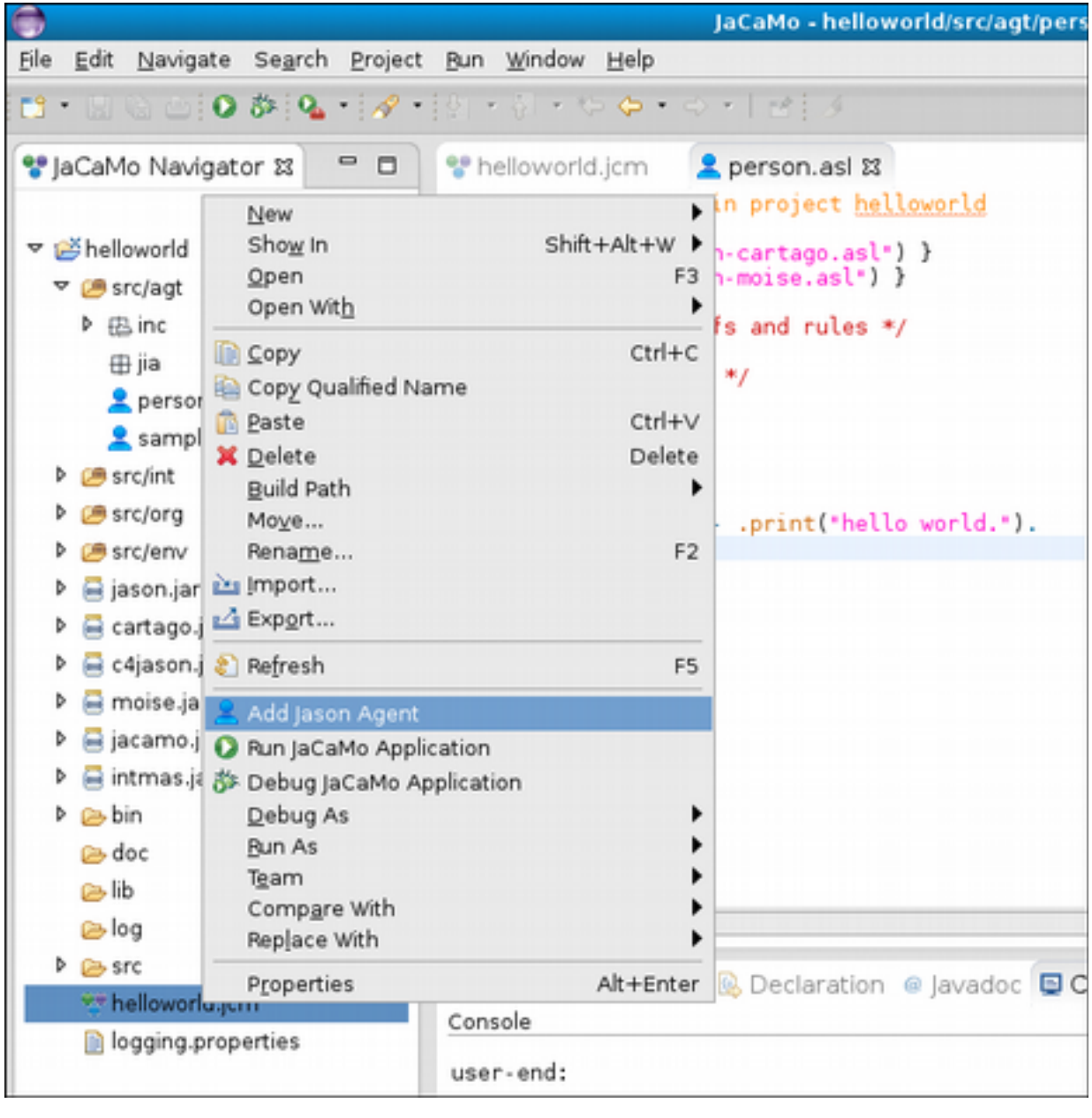
Крок 3

Буде створено тип агента, і з’явиться наступне вікно.


JaCaMo - helloworld/src/agt/p
File Edit Navigate Search Project Run Window Help
JaCaMo Navigator
helloworld
src/agt
inc
jia
person.asl
sample_agent.asl
src/int
src/org
src/env
jason.jar - /home/maicon/svr
helloworld.jcm
person.asl
// Agent person in project helloworld
{ include("common-cartago.asl") }
{ include("common-moise.asl") }
/* Initial beliefs and rules */
/* Initial goals */
!start.
/* Plans */
+!start : true <- .print("hello world.").

Крок 4

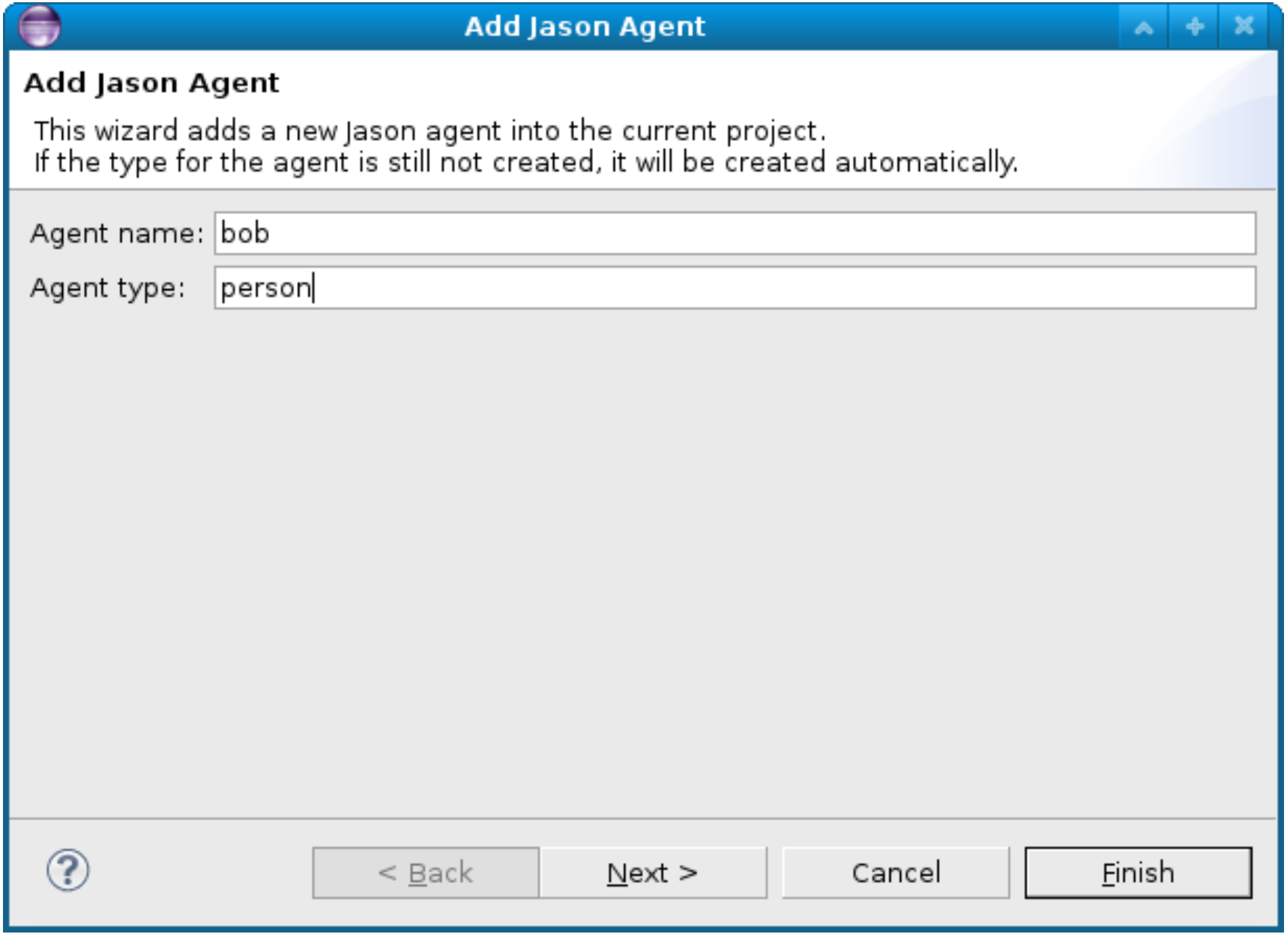
Нові агенти можна створювати за допомогою щойно створеного типу агента. Натисніть на файл проекту за допомогою правої кнопки та перейдіть до опції Додати файл Джейсона.

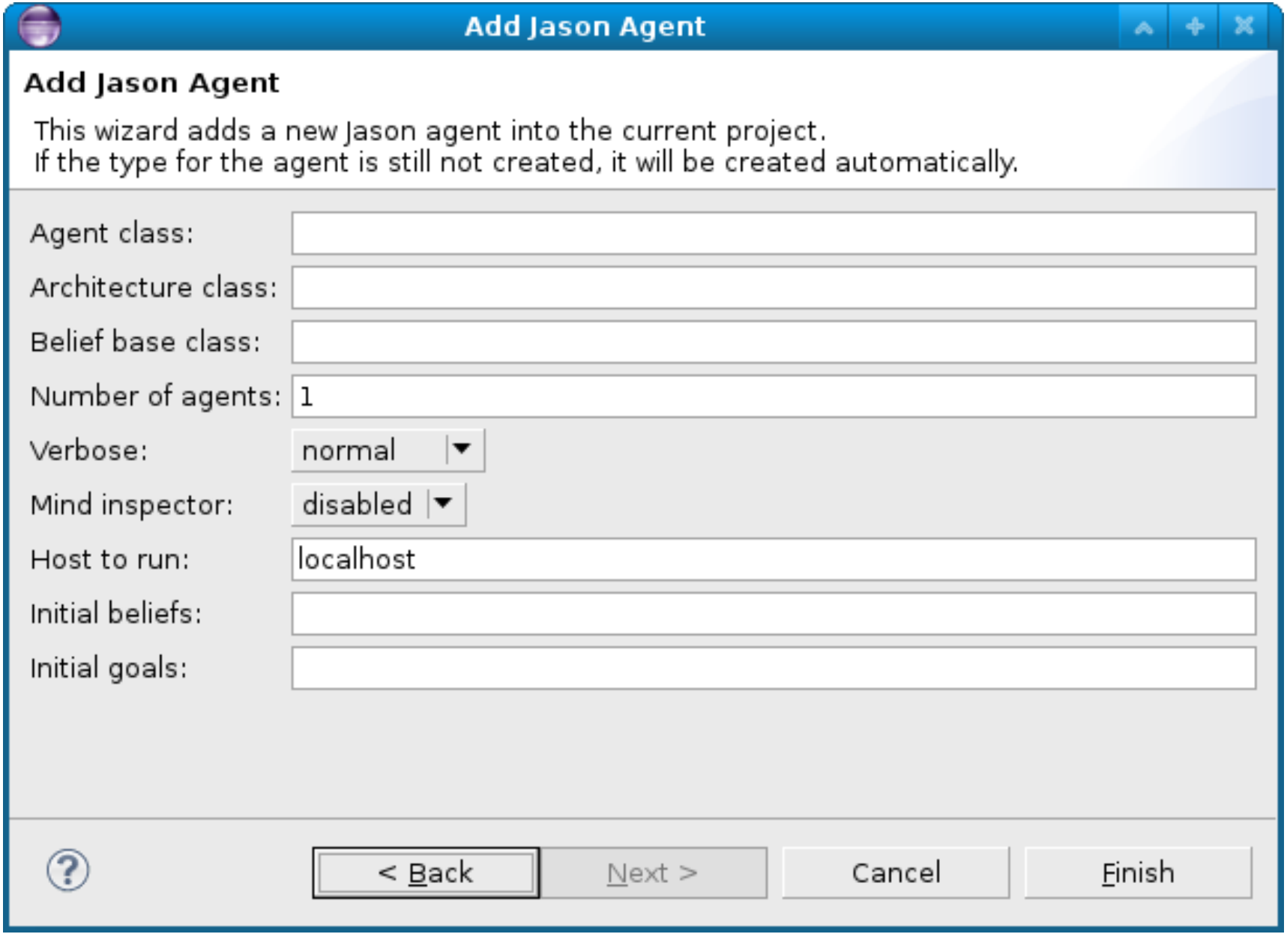

JaCaMo - helloworld/src/agt/pers
File Edit Navigate Search Project Run Window Help
JaCaMo Navigator
helloworld.jcm
person.asl
helloworld
src/agt
inc
jia
src/int
src/org
src/env
bin
doc
lib
log
src
logging.properties
New
Show In Shift+Alt+W
Open F3
Open With
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Delete Delete
Build Path
Move...
Rename... F2
Import...
Export...
Refresh F5
Add Jason Agent
Run JaCaMo Application
Debug JaCaMo Application
Debug As
Run As
Team
Compare With
Replace With
Properties Alt+Enter
in project helloworld
-cartago.asl") }
-moise.asl") }
fs and rules */
*/
.print("hello world.").
Declaration
Javadoc
Console
user-end:

Крок 5

Заповніть форму. Єдиними обов’язковими полями є ім’я та тип агента.

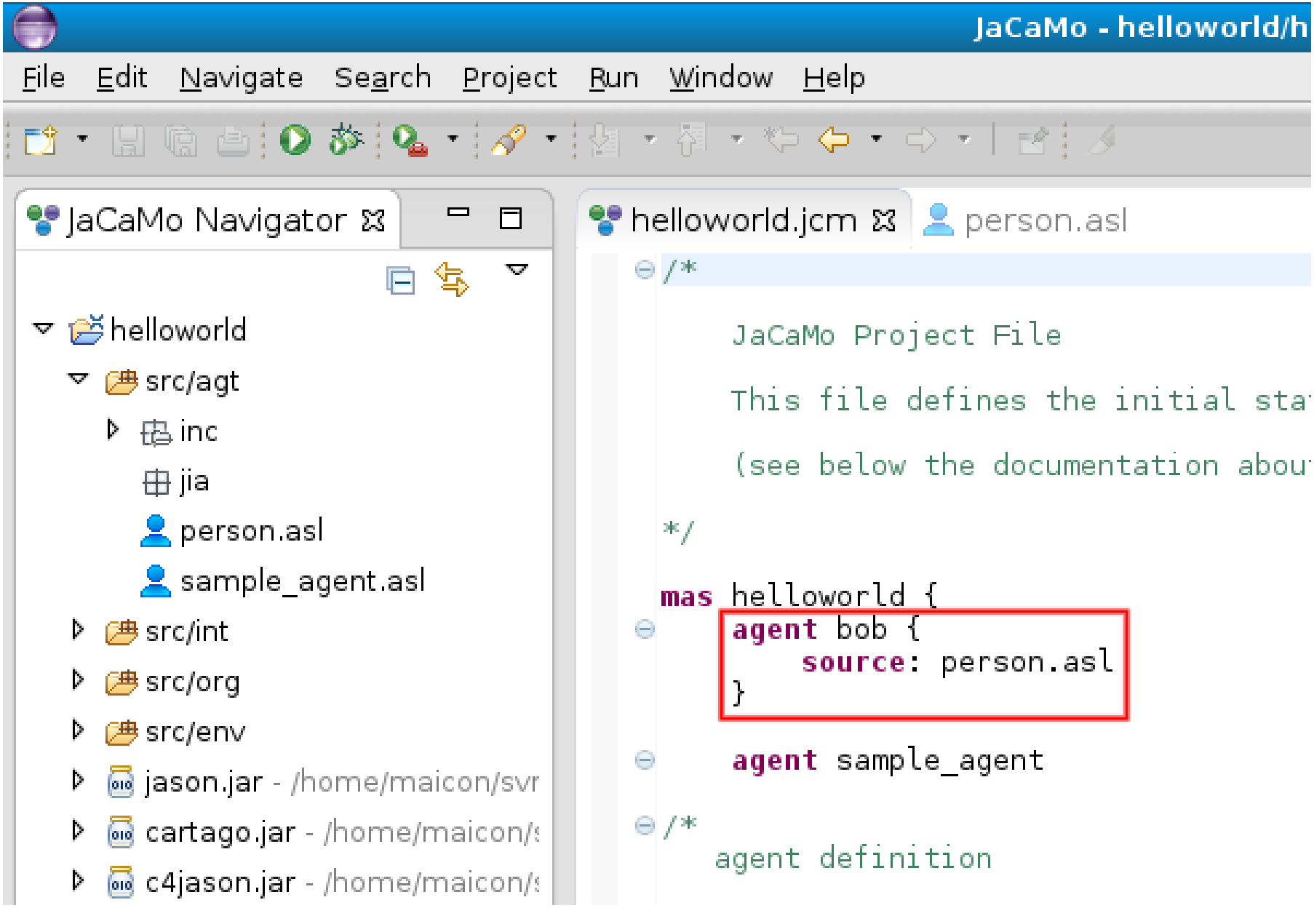
Після цього натисніть кнопку "Готово", якщо більше не буде виконуватися конфігурація, інакше натисніть кнопку "Далі" для розширеної конфігурації.


Add Jason Agent
Add Jason Agent
This wizard adds a new Jason agent into the current project.
If the type for the agent is still not created, it will be created automatically.
Agent name:
bob
Agent type:
person
< Back
Next >
Cancel
Finish


Add Jason Agent
Add Jason Agent
This wizard adds a new Jason agent into the current project.
If the type for the agent is still not created, it will be created automatically.
Agent class:
Architecture class:
Belief base class:
Number of agents:
1
Verbose:
normal
Mind inspector:
disabled
Host to run:
localhost
Initial beliefs:
Initial goals:
< Back
Next >
Cancel
Finish

Крок 6

Новий агент буде автоматично доданий у файл проекту.


JaCaMo - helloworld/h
File Edit Navigate Search Project Run Window Help
JaCaMo Navigator
helloworld
src/agt
inc
jia
person.asl
sample_agent.asl
src/int
src/org
src/env
jason.jar - /home/maicon/svr
cartago.jar - /home/maicon/s
c4jason.jar - /home/maicon/s
helloworld.jcm
person.asl
/*
JaCaMo Project File
This file defines the initial sta
(see below the documentation abou
*/
mas helloworld {
agent bob {
source: person.asl
}
agent sample_agent
/*
agent definition

Завдання:

2.1 Змініть реалізацію трьох версій програми Hello-World: координація за допомогою спілкування, координація за допомогою середовища, координація за організацією (див. у лекції 2). Ці зміни складаються з

а) додавання нового повідомлення з трьома словами «Привіт, чудовий світ» та третього агента для обробки цього нового слова;

б) друк слів повідомлення у зворотному порядку;

в) паралельний друк слів повідомлення (підказка: у файлі XML, де визначено специфікацію організації, варіанти оператора плану - це послідовність, вибір та паралель); та

г) друкувати повідомлення знову і знову, як тільки воно буде надруковане.

Для кожної з цих змін оцініть, яку версію (координація за допомогою спілкування з агентами, координація за допомогою середовища або координація за організацією) найлегше реалізувати.

2.2 Змініть агентів так, щоб вони могли надрукувати повідомлення «Hello World» різними мовами, та запровадіть механізм для легкої зміни мови.

2.3 У нормативних специфікаціях організації замініть дозвіл на зобов'язання та зверніть увагу на різницю у виконанні.

2.4 У структурній специфікації створіть дві ролі (враховуючи друк "Hello World", як це зроблено в лекції 2) або три ролі (враховуючи друк "Hello Wonderful World", як це було зроблено у першій вправі), по одній для кожного слова повідомлення.

Література

Olivier Boissier, Rafael H. Bordini, Jomi F. Hübner, Alessandro Ricci, Andrea Santi. Multi-agent oriented programming with JaCaMo. - Science of Computer Programming 78 (2013) 747–761.

Rafael C. Cardoso and Angelo Ferrando. A Review of Agent-Based Programming for Multi-Agent Systems. - Computers 2021, 10, 16. - https://doi.org/10.3390/computers10020016 .

Multi-Agent Programming: Languages, Platforms and Applications. - © 2005 by Springer. - ISBN-10: 0-387-24568-5

Engineering Multi-Agent Systems. 6th International Workshop, EMAS 2018. Stockholm, Sweden, July 14–15, 2018. - © Springer Nature Switzerland AG 2019. - https://doi.org/10.1007/978-3-030-25693-7

Rafael H. Bordini · Mehdi Dastani · Jürgen Dix · Amal El Fallah Seghrouchni, Ed. Multi-Agent Programming: Languages, Tools and Applications. - © Science+Business Media, LLC 2009. - DOI 10.1007/978-0-387-89299-3

Наступна лекція буде присвячена програмуванню інтелектуальних агентів в системі JaCoMo