Мультиагентно-орієнтоване програмування

3-й рівень навчання, осінь 2021

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

Лекція 4

Вимір середовищ в JaCaMo

Після введення виміру агента в попередній лекції, ми тепер поступово збагачуємо наш набір абстракцій програмування, розглядаючи вимір середовища, який використовується в MAOP для моделювання ресурсів та інструментів, які використовуються агентами як першокласних концепцій. По-перше, ми представляємо концепцію артефакту, яка є основною цеглою, яку можна використовувати для проектування модульних, динамічних і компонованих середовищ — як артефакти в середовищі людини — обговорюючи відповідні абстракції програмування, доступні в JaCaMo. Потім ми бачимо, як можна запрограмувати агентів, щоб створювати, використовувати та спостерігати за артефактами, щоб зрештою формувати власне середовище відповідно до своїх потреб і цілей.

Нарешті, ми описуємо, як складні середовища на основі артефактів можуть бути структуровані з робочими просторами, концепція, введена для моделювання топології багатоагентної системи, можливо розподіленої по декількох вузлах Інтернету.

Концепція середовища використовується в МАОР як абстракція програмування для моделювання ресурсів та інструментів, які агенти можуть створювати, спільно використовувати та використовувати для виконання своєї роботи. Ефективну аналогію для розуміння точки зору дають людські робочі середовища (рис. 4.1). Щоб виконувати свою роботу, люди (агенти) не тільки спілкуються, але й використовують ресурси та інструменти, доступні в робочому середовищі, що часто є основоположним для того, щоб їх дії були ефективними або неефективними. Крім того, метою їхньої роботи часто є якийсь артефакт, який вони поступово будують, можливо, спільно. Аналогічно, в МАОР ми вводимо рівень абстракції, який дає можливість інкапсулювати в MAS ті послуги та ресурси, які належним чином не моделюються як когнітивні агенти, не будучи тоді ні автономними, ні проактивними (орієнтованими на ціль).

Простим прикладом є дошка. Іншим прикладом є база даних або спільна база знань.

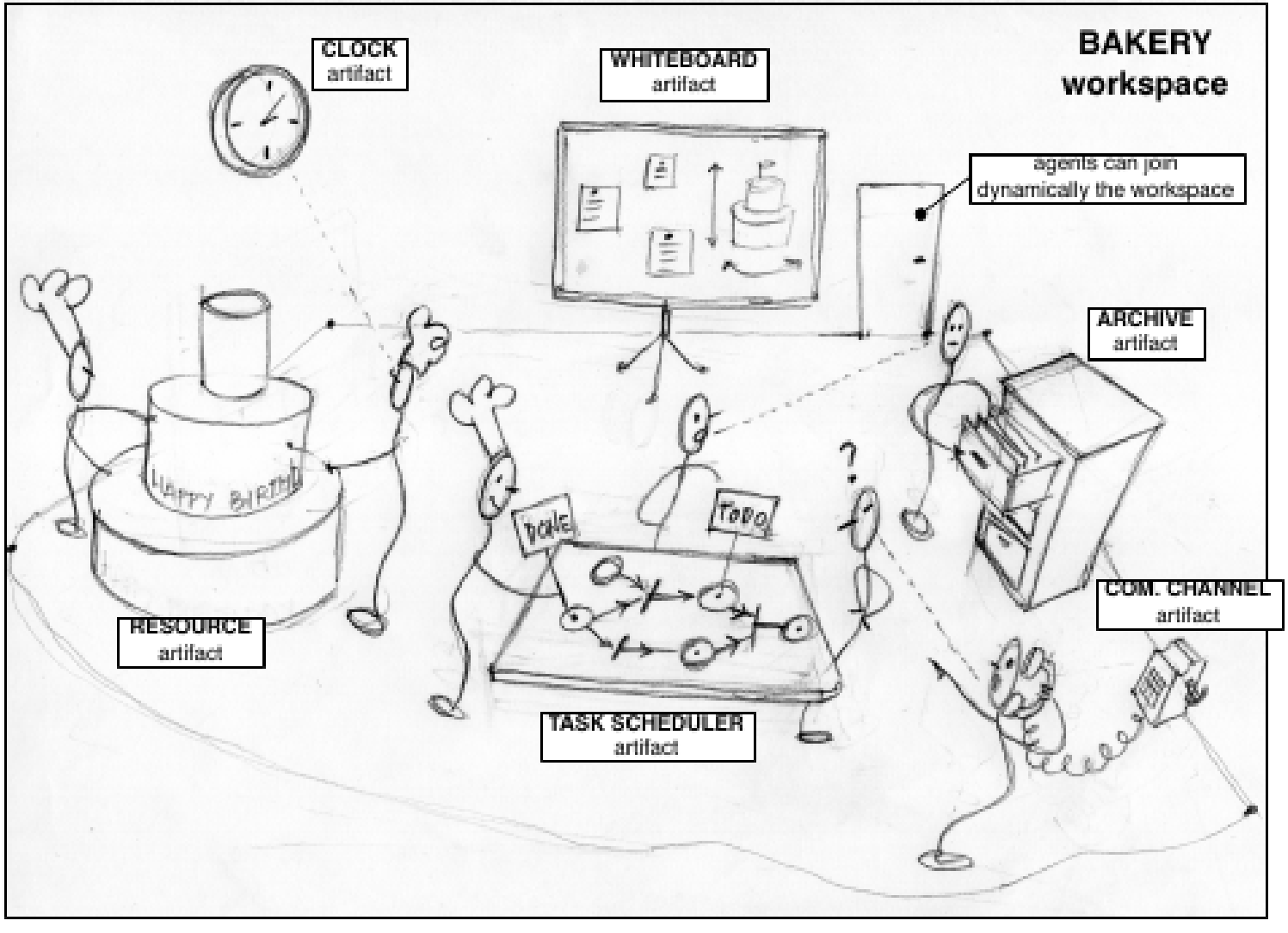

Рис. 4.1 Агенти та артефакти в сценарії пекарні.

На малюнку 4.1 знову показані ключові концепції виміру середовища, представлені в лекції 1, відповідно до концептуальної моделі агентів і артефактів (A&A). В A&A середовище моделюється як робочі простори, до яких агенти можуть приєднатися, щоб поділитися та працювати з артефактами. Термін «артефакт» було взято з теорії діяльності та розподіленого пізнання, щоб пригадати властивості, які таке поняття має в контексті людського середовища. Абстракція артефакту вводиться як одиниця структури та організації середовища.

З точки зору дизайнера MAS, артефакти є основними будівельними блоками — або, радше, першокласною абстракцією — для проектування та розробки середовища агентів; з точки зору агента, артефакти — це першокласні сутності свого світу, які вони можуть створювати, виявляти, ділитися і в кінцевому підсумку використовувати та спостерігати для виконання своєї діяльності та досягнення своїх цілей. Якщо агенти є базовими елементами для проектування автономної та орієнтованої на ціль/завдання частини MAS, то артефакти є основними сутностями для організації неавтономної, функціонально-орієнтованої (з точки зору агента) її частини. Термін функція використовується тут для позначення цільової мети.

Тоді артефакти є природною абстракцією для моделювання та реалізації існуючих обчислювальних об’єктів і механізмів, які часто вводяться в розробку MAS, що представляють спільні ресурси, такі як спільні сховища даних або координаційні носії, такі як дошки, кортежні простори та менеджери подій. Крім цього, аналогічно людському контексту, артефакти можуть бути спеціально розроблені для покращення того, як агенти виконують завдання, шляхом розподілу дій у часі (попереднє обчислення) і агентів (розподілене пізнання), а також шляхом зміни способу, яким люди виконують свою діяльність.

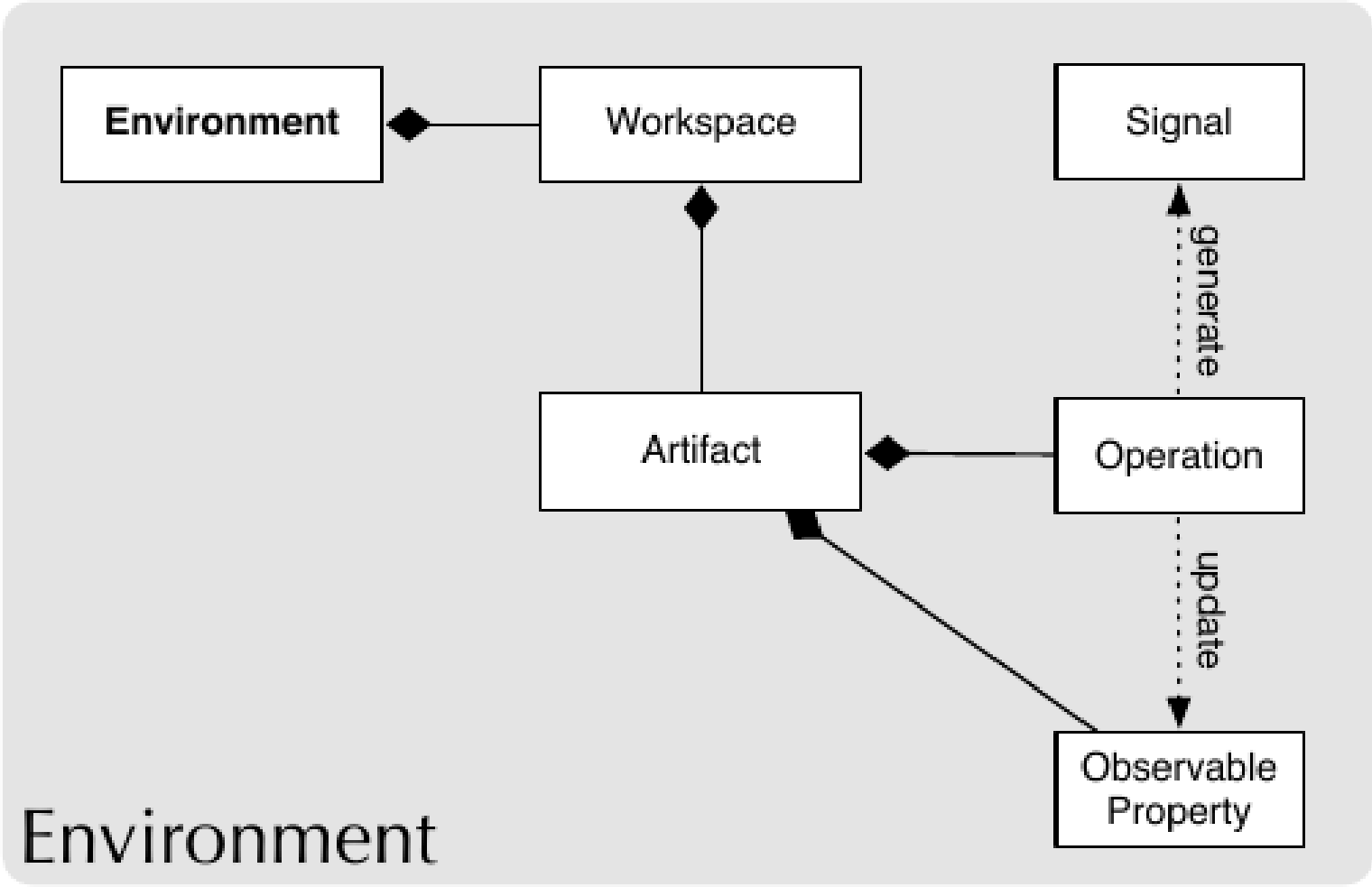

Рис. 4.2 Основні поняття у вимірі середовища.

A&A також вводить абстракцію робочого простору, щоб структурувати та організувати загальний набір артефактів (і агентів) у MAS з топологічної точки зору. Робоча область — це логічний контейнер агентів та артефактів, що забезпечує логічне уявлення про місцевість та розміщення для агентів, визначивши область для взаємодій і

спостережуваність подій, а також для набору пов'язаних дій, що здійснюються групою агентів, використовуючи деякий набір артефактів. Складний MAS може бути організований як набір робочих просторів, розподілених між кількома вузлами мережі, при цьому агенти, можливо, приєднуються до кількох робочих просторів одночасно.

Як підсумок, рис. 4.3 показує концепції середовища, представлені на даний момент в цьому розділі, і пов'язує їх з виміром агента, представленим у попередній лекції. Важливо підкреслити загальність концепцій A&A. Хоча в цій дисципліні ми використовуємо конкретну модель програмування та API, надані JaCaMo (як введено з наступного розділу), поняття (автономних) агентів та (неавтономних) робочих просторів середовища спільного використання артефактів є досить загальними для проектування та реалізації мульти- агентні системи.

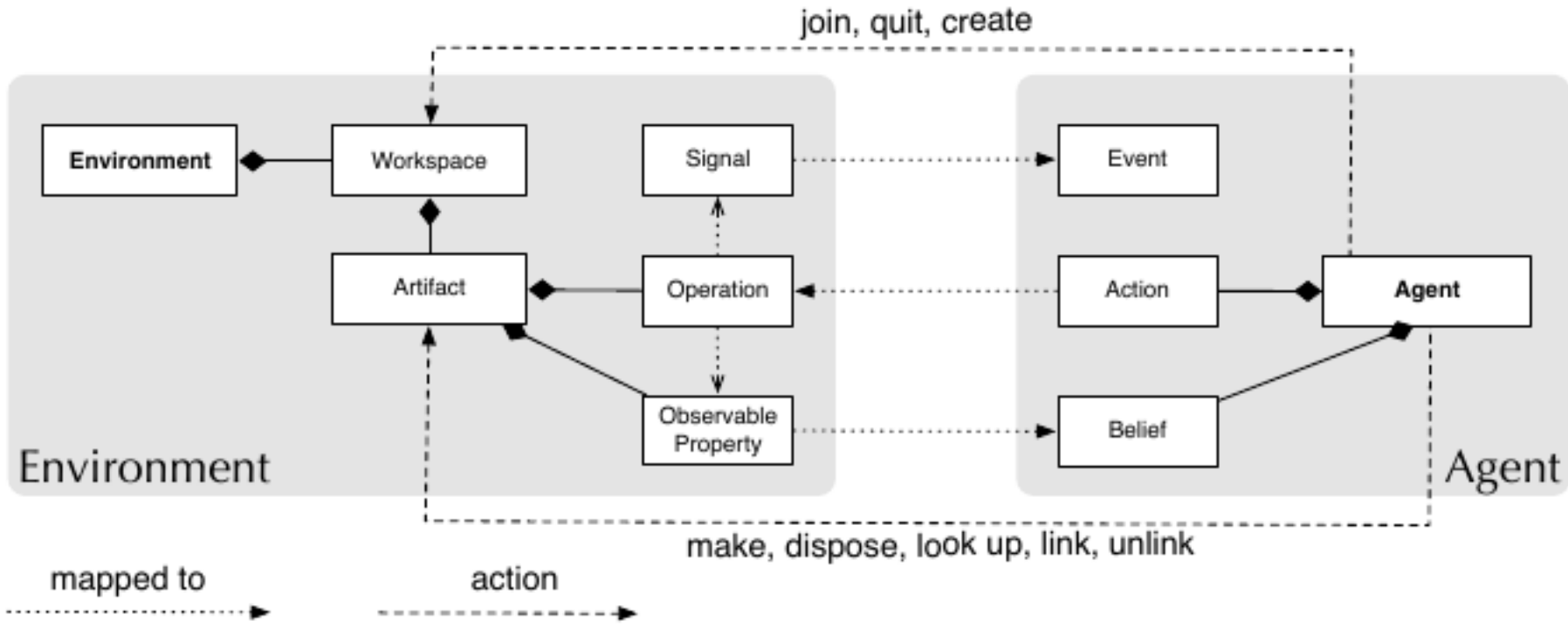

Рис. 4.3 Карта концепцій середовища масштабування та їх зв'язки з виміром агента.

Абстракції середовища

Якщо абстракція агента призначена для ефективного моделювання автономних, проактивних, орієнтованих на ціль обчислювальних об’єктів, то абстракція артефактів призначена для ефективного моделювання функціонально-орієнтованих, неавтономних обчислювальних об’єктів, які призначені для використання (спостереження або контролю) агентами для виконують свою роботу. На малюнку 4.4 показана концептуальна модель, запропонована в A&A і прийнята в JaCaMo для представлення артефактів як обчислювальних сутностей. Функціональні можливості артефакту визначаються в термінах операцій, що відповідають — з точки зору агента — діям, наданим агентам, які використовують цей артефакт. Кожна операція позначається міткою та списком вхідних параметрів.

```
class Counter extends Artifact {

  void init(){
    defineObsProp("count",0);
  }

  @OPERATION void inc(){
    ObsProperty p = getObsProperty("count");
    p.updateValue(p.intValue() + 1);
    signal("tick");
  }
}
```

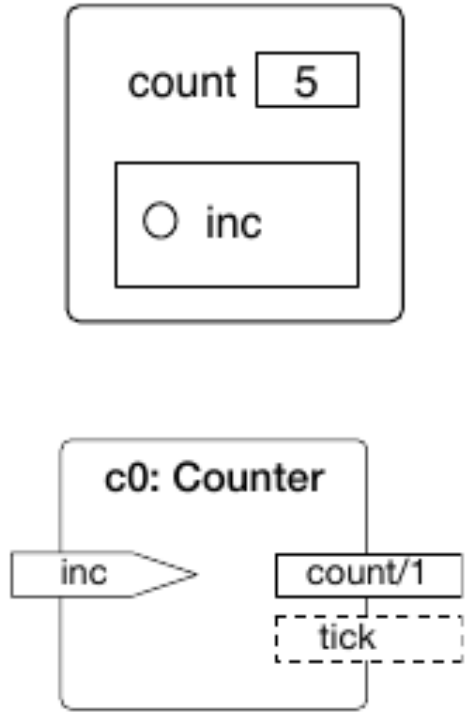

Рис. 4.4 Простий приклад артефакту **`Counter`**. На лівій стороні знаходиться вихідний код Java класу шаблону артефакту. Праворуч є два різних уявлення — абстрактне (зверху) і більш детальне (знизу), — які використовуються в цій дисципліні.

З точки зору програмування, подібно до об’єктів в ООП, артефакти є екземплярами шаблонів артефактів. У JaCaMo шаблон артефакту може бути реалізований як клас Java, що розширює вже існуючий клас, який називається Artifact і доступний в API, і використовує базовий набір анотацій Java та існуючих методів для визначення елементів структури та поведінки артефакту. Як приклад, малюнок 4.4 показує реалізацію простого артефакту лічильника, який відстежує підрахунок.

Операції є основними одиницями для структурування функціональних можливостей артефактів і можуть бути атомарними або процесом, що включає послідовність атомарних обчислювальних кроків. Артефакт лічильника має одну операцію, яка називається inc, для збільшення значення лічильника.

Спостережувані властивості представляють спостережуваний стан артефакту, який може бути сприйнятий агентами, які спостерігають цей артефакт. У нашому прикладі артефакт Counter має одну спостережувану властивість під назвою **`count`**, яка відстежує значення **`count`**. Значення властивості count збільшується на операцію **`inc`**. Кожного разу, коли спостережуваний стан артефакту змінюється операцією, агенти, які спостерігають за артефактом, можуть сприймати подію, що відповідає новому стану. Артефакт також може мати прихований, неспостережуваний стан, який може бути закодований, наприклад, у термінах приватних полів екземпляра.

На додаток до спостережуваних властивостей, артефакт може генерувати сигнали, які є спостережуваними подіями, які можуть бути сприйняті агентами, які використовують або спостерігають артефакт. Сигнали можуть використовуватися для представлення будь-якої ситуації, умов або повідомлення і не обов'язково пов'язані з властивостями, які можна спостерігати, щоб передаватися спостережним агентам.

У прикладі артефакт **`Counter`** генерує сигнал, який називається **`tick`**, щоразу, коли кількість збільшується. Крім спостережуваних властивостей і сигналів, агент може отримувати зворотний зв'язок від артефакту в термінах вихідних параметрів в операціях, тобто параметрів, значення яких призначене для встановлення під час виконання операції. З точки зору агента, ці параметри представляють зворотний зв'язок щодо дії.

Нижче наведено приклад:

```
class Calculator extends Artifact {
void init(){
defineObsProperty("last_result",0);
}
@OPERATION void add(double a, double b,
OpFeedbackParam<Double> result){
double res = a + b;
getObsProperty("last_result").updateValue(res)
;
result.set(res);
}
}
```

Операція додавання в артефакті калькулятора має результат вихідного параметра, який встановлюється операцією як сума двох операндів **`a`** і **`b`**, що передаються як звичайні параметри. Параметризований клас **`OpFeedbackParam`** використовується для представлення вихідних параметрів (яких може бути більше одного для однієї операції). У прикладі властивість **`last_result`**, що спостерігається, також використовується для відстеження результату останньої виконаної операції. Це лише простий приклад того, як зворотний зв'язок дії та спостережувані властивості можна використовувати разом.

Нарешті, для підтримки композиції артефакти можуть бути пов’язані між собою, щоб уможливити взаємодію між артефактами. Це здійснюється за допомогою інтерфейсів зв’язку, які аналогічні інтерфейсам артефактів у реальному світі (наприклад, підключення навушників у MP3-плеєр або використання пульта дистанційного керування для телевізора). Зв’язування також підтримується для артефактів, що належать до різних робочих областей, можливо, що знаходяться на різних вузлах мережі.

Основна систематика артефактів

Подібно до об’єктів та інструментів, які використовуються людьми, артефакти можна класифікувати відповідно до функціональних можливостей, які вони надають. У літературі було визначено три основні категорії (Ricci et al. 2006 - *Ricci, Alessandro, Mirko Viroli, and Andrea Omicini. 2006. Programming MAS with artifacts. In Programming Multi-Agent Systems, eds. Rafael H. Bordini, Mehdi M. Dastani, Jürgen Dix, and Amal El Fallah Seghrouchni, 206–221. Springer.*):

• Артефакти ресурсів Це найбільш загальний і поширений тип артефакту, який представляє певний специфічний тип ресурсу, яким можуть користуватися агенти. Прикладом є простий лічильник, згаданий у цій лекції; інший є артефактом спільної бази знань.

• Артефакти координації Ці артефакти спеціально розроблені для забезпечення функціональних можливостей координації, дозволяючи та керуючи певним чином взаємодією між агентами. Приклади варіюються від механізмів синхронізації — аналогічних, наприклад, бар'єрів і семафорів у паралельному програмуванні — до класних дошок, аукціонних машин і механізмів робочого процесу, до артефактів, які використовуються для керівництва організації.
• Граничні артефакти. Ці артефакти дозволяють агентам взаємодіяти з людьми-користувачами і, загалом, з будь-яким актором або системою, яка є зовнішньою щодо MAS; прикладом є графічний інтерфейс.

Розробка та реалізація артефактів координації та кордонів, зокрема, може вимагати використання більш досконалих механізмів, наданих API артефактів, для синхронізації виконання операцій — аналогічно умовним змінним у моніторах — і керування виконанням асинхронних обчислень, взаємодіючи із зовнішніми потоками. . Ці механізми представлені в наступному.

Робота з артефактами з точки зору агента

Набір дій, доступних агентам для роботи з артефактами, можна розділити на три основні групи:

1. дії зі створення/пошуку/утилізації артефактів;
2. дії з використання артефактів, тим самим ініціюючи операції та спостерігаючи властивості та сигнали; і
3. дії для зв'язування/від'єднання артефактів.

Далі опишемо ці дії більш детально.

Створення та виявлення артефактів Ми починаємо зі створення та виявлення артефактів. Артефакти призначені для створення, виявлення та, можливо, утилізації агентами під час виконання; це основний спосіб, за допомогою якого модель підтримує динамічну розширюваність (крім модульності) середовища. На малюнку 4.5 (нижче) ми показуємо простий приклад, у якому користувацький агент створює артефакт і використовує його, виконуючи операції, а агент-спостерігач виявляє існування цього артефакту та реагує на зміни спостережуваних властивостей.

Дія **`makeArtifact(Name,Template,Params,Id)`** створює екземпляр нового артефакту під назвою **`Name of type Template`** всередині робочої області, отримуючи як зворотний зв'язок дії його ідентифікатор **`Id`**. Логічне ім'я ідентифікує артефакт всередині робочої області. Артефакти, що належать до різних робочих областей, можуть мати однакове логічне ім'я, тому на додаток до логічного імені кожен артефакт має унікальний ідентифікатор, згенерований системою і повернутий як зворотний зв'язок щодо дії. Як показано на малюнку 4.5 (ліворуч), у плані для цілі **`! create_and_use`** агент створює артефакт під назвою **`c0`** як екземпляр **`Counter`** шаблону артефакту, передаючи у якості початкового параметра значення **`10`**. Подвійно, щоб **`makeArtifact`**, **`disposeArtifact(Id)`** використовується для видалення артефакту з робочої області.

Виявлення артефакту стосується можливості отримати ідентифікатор артефакту, розташованого в робочому просторі, надавши його логічну назву або опис типу (тобто шаблон, який використовується для його створення). З цією метою **`lookupArtifact(Name,Id)`** отримує унікальний ідентифікатор артефакту з його логічною назвою. На малюнку 4.5 (права частина) план цілі **`!discover_and_observe`** використовується іншим агентом (відмінним від агента, який створив артефакт), щоб шукати артефакт під назвою **`c0`** і спостерігати за ним (зосереджуючи увагу на ньому). З цією метою він спершу намагається отримати ідентифікатор артефакту за допомогою підцілі **`!locate_count`**, у якій він багаторазово виконує пошук артефакту, поки артефакт не буде знайдено.

Виконання операцій над артефактами Для агентів використання артефакту по суті включає два аспекти: можливість виконувати операції, які фактично перераховані в інтерфейсі використання артефактів, і здатність сприймати інформацію, яку можна спостерігати за артефактом, у термінах спостережуваних властивостей і сигналів.

Для першого аспекту, з точки зору агента, операції з артефактами представляють зовнішні дії, надані агентам середовищем. При виконанні дії `op(Params)` запускається відповідна операція, що забезпечується артефактом у робочій області. Як показано на малюнку 4.5 (ліворуч), користувальницький агент створює артефакт лічильника з назвою `c0`, а потім збільшує його, виконуючи дію inc, яка запускає відповідну операцію inc над артефактом. Насправді, якщо більше одного артефакту в робочих областях, до якого приєднався агент, забезпечує операцію, ідентифікатор артефакту-цілі операції можна вказати за допомогою анотації `[artifact_id(Id)]` після дії. Якщо анотація не вказана і є кілька артефактів, які забезпечують операцію, вибирається один (недетермінований, з точки зору агента).

Дія, що виконується агентом, буде успішною, якщо відповідна операція завершується успішно; навпаки, дія завершується, якщо або зазначена операція на даний момент не включена в інтерфейс використання артефакту, або якщо під час виконання операції сталася помилка, тобто сама операція не вдалася. Успішно завершивши своє виконання, операція може генерувати деякі результати, які можна повернути агенту як зворотний зв'язок щодо дії. Виконуючи дію, намір агента-охоплювача призупиняється, доки не буде отримана подія, яка повідомляє про завершення дії (з успіхом або невдачею). Отримавши подію завершення дії, виконання дії завершується, а відповідний план відновлюється.

Варто зауважити, що навіть якщо намір призупинено, агент не блокується, і цикл міркування агента може тривати на обробці сприйняття та виконанні дій, пов’язаних з іншими планами. Таким чином, у середовищах на основі артефактів репертуар зовнішніх дій, доступних для агента, визначається набором артефактів, які в даний момент заповнюють середовище. Це означає, що репертуар дій може бути динамічним, оскільки набір артефактів може динамічно змінюватися самими агентами, створюючи нові артефакти або видаляючи наявні артефакти.

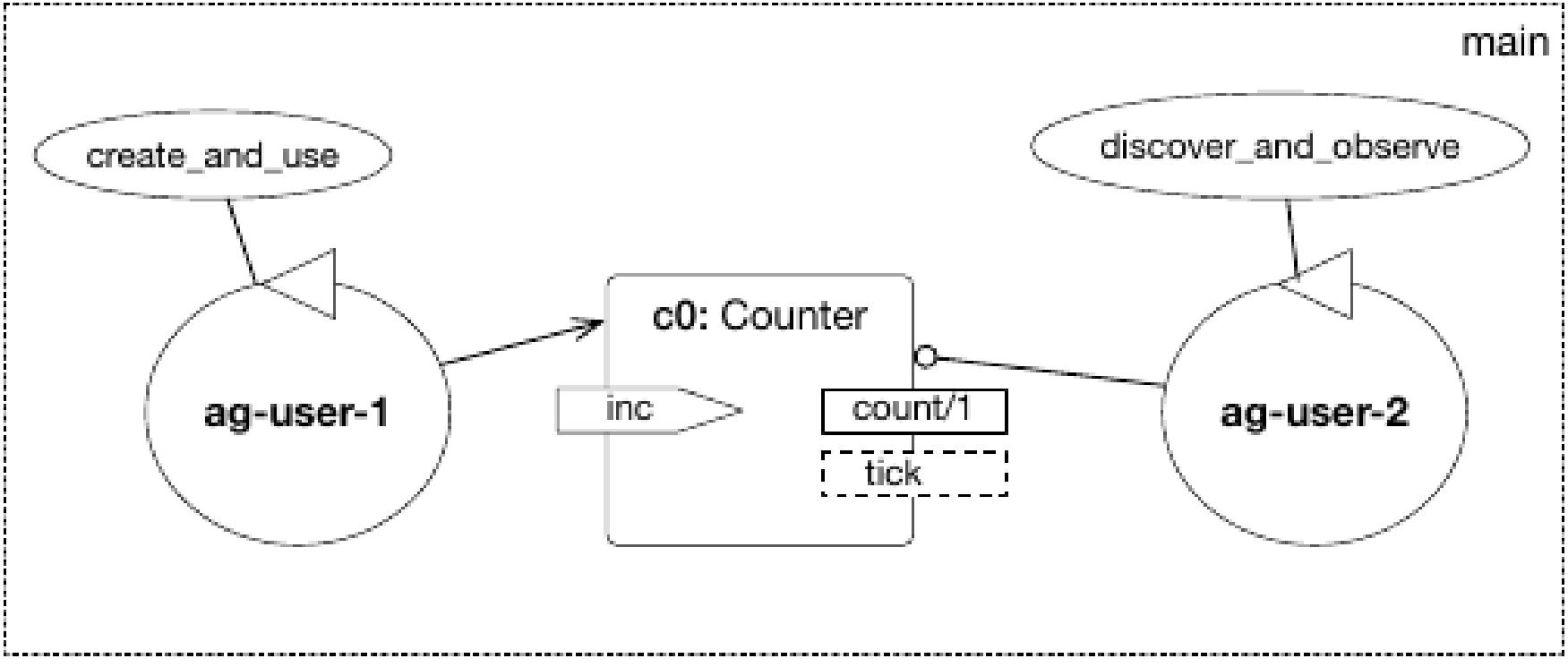

Рис. 5.5 Агенти, які використовують і спостерігають за артефактом. У верхній частині є діаграма для прикладу користувача та спостерігача. Ліворуч показано, як користувальницький агент створює та використовує артефакт лічильника під назвою c0. Праворуч показаний агент-спостерігач, який відкриває артефакт c0, спостерігає за ним і реагує на зміни його властивості count спостережуваного.

```
/* user agent */

!create_and_use.

+!create_and_use : true <-
  makeArtifact("c0",
        "Counter",[10],Id);
  inc;
  inc [artifact_id(Id)].
```

```
/* observer agent */

!discover_and_observe.

+!discover_and_observe <-
  !locate_count(Id);
  focus(Id).

+count(V) <-
  println("count:  ",V).

+tick <- println("tick!").

+!locate_count(Id) <-
  lookupArtifact("c0",Id).
-!locate_count(Id) <-
  .wait(10);
  !locate_count(Id).
```

Попередньо визначені артефакти

Навіть основні дії, такі як **`makeArtifact`**, подібні до будь-якої іншої зовнішньої дії; тобто вони рівномірно доступні для агентів, оскільки існує артефакт, що показує їх як операції. Зокрема, кожна робоча область за замовчуванням оснащена артефактом під назвою робоча область, який надає основні функції для створення, пошуку та керування артефактами. Більше того, він надає набір спостережуваних властивостей, які дають змогу агентам знати основні динамічна інформація про робочу область, наприклад набір доступних артефактів (через переконання артефакту форми (назва, шаблон, ідентифікатор)).
Артефакт робочої області — це один із попередньо визначених артефактів, які за замовчуванням створюються в робочих областях як основні інструменти, які агенти можуть використовувати у своїй діяльності.

Серед інших попередньо визначених артефактів

• артефакт **`console`**, який забезпечує операції взаємодії зі стандартним введенням/виводом (наприклад, дія **`println`**, використана в прикладах);

• артефакт **`blackboard`**, який реалізує просту дошку (на основі кортежу), корисну для координації агентів.

Повний опис цих артефактів є частиною документації, доступної на веб-сайті платформи.

Спостереження за артефактами. Крім виконання операцій над артефактами, їх використання зазвичай вимагає вміння спостерігати за ними. Агент може почати сприймати спостережувані властивості та сигнали, згенеровані артефактом, виконавши дію `focus(Id,Filter)`, вказавши ідентифікатор артефакту для спостереження і, за бажанням, фільтр для подальшого вибору підмножини спостережуваних властивостей і сигналів, у яких агент цікавиться. Подвійно для фокусування надається дія `stopFocus(Id)` для випадку, коли агент більше не хоче спостерігати певний артефакт.

Під час спостереження за артефактом спостережувані властивості артефакту відображаються безпосередньо в переконаннях в основі переконань агента. Кожного разу, коли спостережувана властивість сфокусованого артефакту оновлюється, «за кадром» артефакт генерує подію, яка автоматично сприймається агентом, і відповідне переконання оновлюється, генеруючи подію зміни віри. Це дає можливість писати плани, які реагують на зміни спостережуваних властивостей. На малюнку 4.5 (праворуч) агент-спостерігач спочатку виявляє лічильник `c0`, а потім починає спостерігати за артефактом і реагує щоразу, коли оновлюється переконання про кількість спостережуваних властивостей `count(V)`. Подія також генерується, коли переконання, пов'язане з властивістю, що спостерігається, створюється вперше (тобто, коли фокусна дія успішно виконується).

Сигнали, навпаки, не пов'язані з спостережуваними властивостями; вони схожі на повідомлення, створені на стороні артефакту, які асинхронно обробляються на стороні спостерігача. Відповідно, жодні переконання щодо сигналів не зберігаються за замовчуванням. Агент може сфокусувати (спостерігати) одночасно кілька артефактів, навіть одного виду. Щоб відрізнити спостережувані властивості з такою ж назвою, але від різних артефактів, переконання про спостережувані властивості анотуються конкретним ідентифікатором артефакту, який є джерелом цього переконання. Зокрема, кожне переконання про спостережувану властивість артефакту анотується **`artifact_id(Id)`**, **`artifact_name(Name)`** і **`Workspace(Id)`**, що містить інформацію про унікальний ідентифікатор артефакту, його назву та ідентифікатор. робоча область, де вона розміщена, відповідно.

Ці анотації можна використовувати, наприклад, для написання планів агентів:

```
+count(V) [artifact_name("counter")] <- ...
+count(V) [artifact_id(Id)] <- /* use the Id */...
```

Важливі зауваження щодо семантики спостереження:

• **Повнота спостереження.** Модель така, що жодні події не можна втратити. Тобто для кожного спостережуваного стану s, створеного артефактом, тобто для кожного нового значення v, яке можна спостерігати щодо спостережуваної властивості, цей стан сприймається кожним агентом, який спостерігає артефакт, і генеруються відповідні внутрішні події.

• **Упорядкування подій.** Стани та події, які генеруються артефактом, сприймаються кожним агентом, який спостерігає за артефактом, у тому ж порядку, в якому вони генеруються. І навпаки, порядок між подіями, створеними різними артефактами, не визначається.

• **Атомне сприйняття.** Якщо дві спостережувані властивості змінюються під час виконання однієї операції, то їх зміна сприймається за допомогою одного сприйняття, і відповідні переконання агента оновлюються в одному циклі міркувань, генеруючи кілька внутрішніх подій, які можуть бути оброблені в різних наступних циклах міркувань.

Зв'язування артефактів Нарешті, ми опишемо дії зв'язування артефактів. Артефакти в людському середовищі часто створюються для з’єднання між собою, щоб об’єднати їх функціональні можливості. Аналогічно, артефакти тут можуть бути пов’язані між собою агентами, щоб дозволити одному артефакту (з’єднувому) виконувати операції над іншим артефактом (зв’язаним, який має бути зв’язаним шляхом надання належного інтерфейсу посилання). Щоб з’єднати два артефакти, агентам доступна дія **`linkArtifacts(LinkingArId, LinkedArId,Port)`**.

На стороні артефакту зв’язку поняття порту використовується для (опосередковано) посилання на зв’язаний артефакт у коді артефакту. Після того, як артефакти пов’язані разом, артефакт зв’язку може виконувати операції над артефактом, приєднаним до деякого порту, за допомогою примітиву **`execLinkedOp(Port,OpName,OpArgs)`**. Малюнок 4.6 (ліворуч) показує приклад артефакту зв'язування, який визначає порт **`linkedCount`** та використовує його в операціях **`test`** і **`test2`** для виконання операцій над зв'язаним артефактом.

```
public class LinkingArtifact
      extends Artifact {

 private static final String
         linkedCount = "linkedCount";

 void init(){
   definePort(linkedCount);
 }

 @OPERATION
 void test(){
   execLinkedOp(linkedCount,"inc");
 }

 @OPERATION
 void test2(OpFeedbackParam<Integer> v){
   execLinkedOp(linkedCount,"getValue", v);
   log("back from linked op.: "+v.get());
 }
}
```

```
public class LinkableArtifact
       extends Artifact {

  int count;

  void init(){
    count = 0;
  }

  @LINK
  void inc(){
    count++;
  }

  @LINK
  void getValue(
       OpFeedbackParam<Integer> v){
    v.set(count);
  }
}
```

```
!test_link.

+!test_link
  <-  makeArtifact("myArtifact","LinkingArtifact",[],Id1);
      makeArtifact("count","LinkableArtifact",[],Id2);
      linkArtifacts(Id1,"linkedCount",Id2);
      println("artifacts linked: going to test");
      test;
      test2(V);
      println("value ",V).
```

Рис. 4.6 (дивись на попередньому слайді). Зв’язування артефактів. Ліворуч – приклад артефакту зв’язування. Порт з іменем **`linkedCount`** визначено і використовується для ініціювання виконання операцій над пов’язаним артефактом за допомогою операцій **`test`** і **`test2`**. Праворуч наведено приклад зв’язуваного артефакту. Операції, анотовані **`@LINK`**, можуть бути викликані шляхом зв'язування артефактів. Унизу агент створює два артефакти, пов’язуючи їх разом і виконуючи операції над артефактом зв’язування.

На стороні зв’язаного артефакту відкривається інтерфейс зв’язку, який визначає набір операцій, які можна виконати шляхом зв’язування артефактів. Інтерфейс посилання визначається шляхом анотування за допомогою **@LINK** тих операцій артефакту, які можуть бути пов’язані іншими артефактами.

Малюнок 4.6 (праворуч) показує приклад артефакту, який можна підключити, що відкриває пару операцій, анотованих **@LINK**. Семантика виконання операції зв'язування така ж, як і операцій, що виконуються агентами: запит на операцію, що виконується артефактом зв'язування, призупиняється до тих пір, поки операція над зв'язаним артефактом не буде виконана, незалежно від того, чи буде вона успішно або невдало.

На малюнку 4.6 (внизу) показано приклад коду агента, який створює та зв’язує два артефакти (**`LinkingArtifact`** і **`LinkableArtifact`**), а потім взаємодіє з артефактом зв’язування, опосередковано виконуючи операції також над пов’язаним.

Модульність—Інкапсуляція—Повторне використання

Модель середовища, що використовується в JaCaMo, має багато найважливіших функцій, які очікуються від мов програмування та методологій програмної інженерії. Наприклад, артефакти є природним засобом модульності, оскільки артефакти можуть пов’язуватися з іншими артефактами за допомогою пов’язаних операцій.

Артефакт – це передусім механізм інкапсуляції, оскільки всі операції та спостережувані властивості, концептуально пов’язані з сутністю артефакту, яку сприймають агенти, реалізуються в межах однієї конструкції артефакту. Ще одна важлива особливість – можливість повторного використання;

Зайве говорити, що один і той же шаблон артефакту буде корисний у багатьох різних системах, які розробник може захотіти реалізувати — уявіть собі стіл, дошку, кавоварку чи будь-який інший реальний артефакт; і буде легко побачити, як часто одні й ті самі артефакти корисні в багатьох різних контекстах, і те саме стосується артефактів у різних мультиагентних системах.

Структурування артефактів у робочій області

Як згадувалося на початку лекції, в моделі середовища, прийнятій в JaCaMo, артефакти збираються в робочі області, визначаючи топологію середовища. Робоче середовище можна розуміти як логічне місце, що містить артефакти, а також як контекст роботи для діяльності агентів. Щоб отримати доступ до артефактів робочого простору та використати його, тобто поділитися цим контекстом роботи, агент повинен спочатку приєднатися до нього. Агент може приєднатися і працювати в кількох робочих просторах, а кілька агентів можуть працювати в одному робочому просторі одночасно.

За замовчуванням MAS містить одну робочу область. Складне середовище, однак, може бути структуровано в термінах кількох робочих просторів, організованих ієрархічно, подібно до файлових систем; див. малюнок 4.7. Існує коренева робоча область, яка за замовчуванням називається main, але можна використовувати більш конкретні імена; наприклад, на малюнку 4.7 коренева робоча область називається **`my_bakery`**. Кожна робоча область може мати одну або кілька дочірніх робочих областей, але лише одну батьківську. Як і у файлових системах, логічні шляхи можна використовувати для посилання на робочу область, наприклад, **`/my_bakery/cake_room`**.

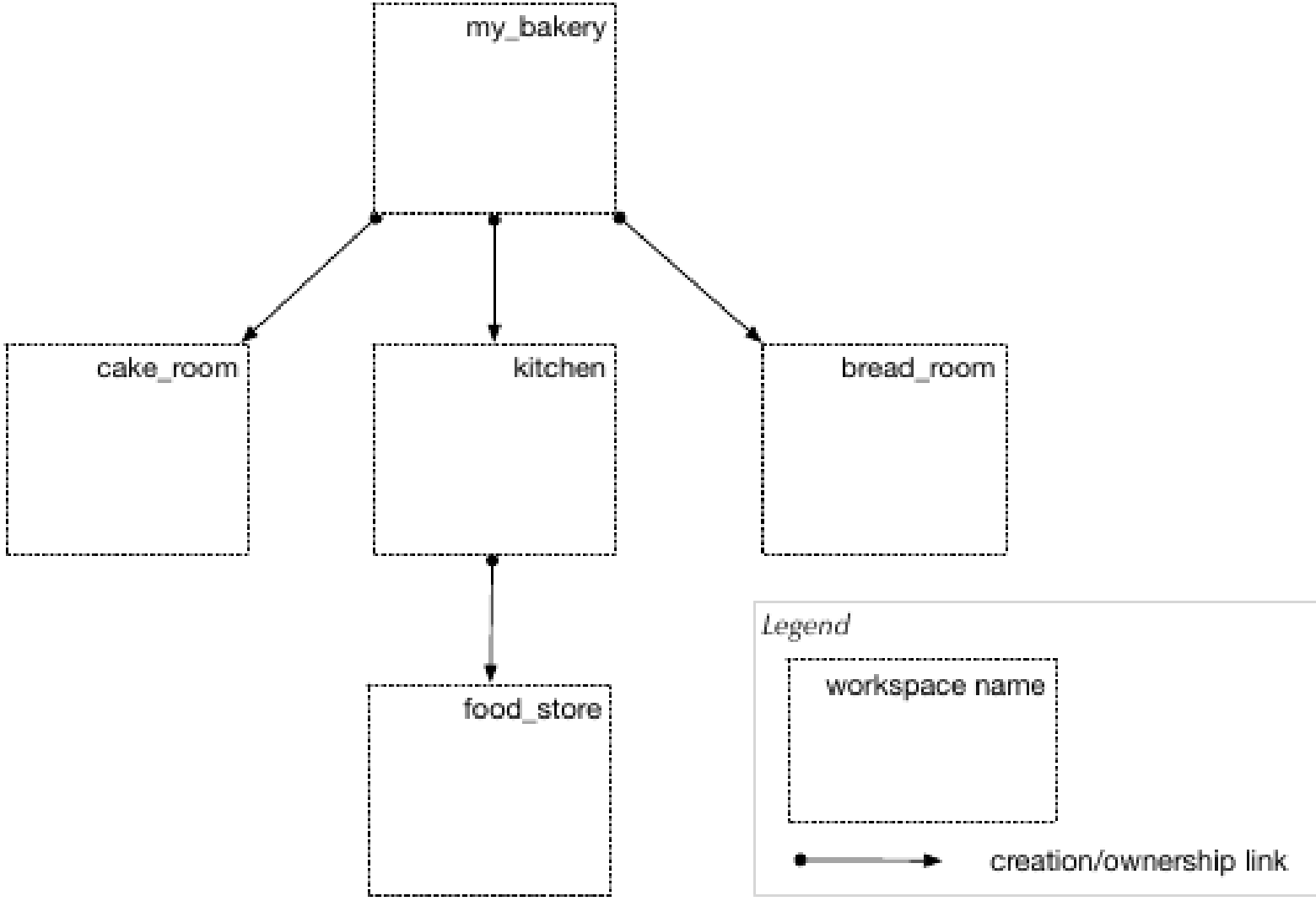

Рис. 5.7 Огляд розподілених середовищ.

На стороні агента доступні деякі дії для роботи з робочими областями. По-перше, агенти породжуються або входять в MAS у певному робочому просторі (їхньому домашньому робочому просторі), яке зазвичай не змінюється протягом їхнього життя в межах цього MAS, за винятком мобільних агентів. Після введення MAS агент може працювати одночасно в кількох робочих просторах MAS, просто приєднавшись до них.

Дію **`joinWorkspace`** можна використовувати для приєднання до будь-якої робочої області MAS, вказавши повне ім'я шляху до робочої області та отримавши її унікальний ідентифікатор як зворотний зв'язок дії. Прикладом є **`joinWorkspace("/my_bakery/cake_room",Id)`**. Після приєднання робочої області агент може взаємодіяти з усіма артефактами в цьому робочому просторі. Щоб вийти з робочої області, передбачено дію **`quitWorkspace(Id)`**.

Окрім приєднання, агент може створити нову робочу область за допомогою дії **`createWorkspace`**, вказуючи ідентифікатор батьківського робочого простору та логічне ім'я, яке буде використовуватися для нового дочірнього робочого простору. Робочі області також можна видалити за допомогою дії **`removeWorkspace`**, вказуючи повне ім'я (шлях) робочої області, яку потрібно видалити. Нарешті, щоб створити зв'язок доступу між двома робочими областями, існує дія **`linkWorkspaces(From,To,Name)`**, де **`From`** – це шлях до робочої області, де створено посилання, **`To`** – це шлях робочої області, яку потрібно зв'язати, і **`Name`** — це мітка посилання доступу.

Показано простий приклад вихідного коду агента, який працює з робочими просторами:

```
+!test_workspaces
      <- createWorkspace("/main/w0");
           joinWorkspace("/main/w0",W0);
           println("hello in ",W0);
           createWorkspace("w1");
           joinWorkspace("w1",W0);
           println("hello in ",W1).
```

Агент створює пару робочих областей — **w0** і **w1** — а потім приєднується до них, друкуючи повідомлення на своєму артефакті **console** (який за замовчуванням доступний у кожній робочій області).

Кілька моментів заслуговують додаткової уваги та пояснень. Перший пункт стосується того, як посилається на другу робочу область **`w1`**; що стосується файлових систем, замість абсолютного використовується відносний шлях — відносні шляхи не починаються з **`/`**. Тобто, аналогічно поняттю поточного каталогу при використанні оболонок операційної системи, в даному випадку існує поняття поточної робочої області, яка відповідає останньому приєднаному робочому простору (у випадку, якщо агент працює одночасно в кількох робочих просторах). Відносні шляхи вирішуються відносно поточної робочої області. У прикладі, коли агент у MAS завантажується, за замовчуванням він приєднується до кореневого робочого простору (за замовчуванням називається **`main`**).

Створюючи та приєднуючи робочу область **`w0`**, **`w0`** стає поточною робочою областю. Тоді робоча область **`w1`** створюється як дочірня для **`w0`**, оскільки для посилання на робочу область використовується відносний шлях **`w1`**.

Це еквівалентно **`createWorkspace("/main/w0/w1")`**.

Варто зазначити, що, як і у файлових системах, .. можна використовувати для посилання на батьківську робочу область, тому **`createWorkspace("../w1")`** створив би робочу область всередині **`main`**.

Другий пункт, пов’язаний з першим, стосується виконання операцій над артефактами без вказівки ідентифікатора артефакту (або цільової робочої області). У прикладі дія println посилається на відповідну операцію, надану артефактом консолі. Правдоподібним питанням тут може бути «Якщо агент працює в кількох робочих просторах, який конкретний артефакт консолі використовується, коли запитується дія **`println`**?» Відповідь полягає в тому, що використовується поточна робоча область; тобто, коли дію (операцію) вказано без включення ідентифікатора артефакту, поточна робоча область неявно вважається цільовою, а потім артефакт, що забезпечує цю операцію, розглядається в цій робочій області.

Тому в прикладі перше привітальне повідомлення друкується артефактом консолі в робочій області **`w0`**, тоді як друге привітальне повідомлення друкується артефактом консолі в робочій області **`w1`**.

Поточна робоча область призначена для представлення поточного контексту роботи агента і, з цієї причини, вона пов’язана з поточним наміром у виконанні; тобто кожен намір може мати власну поточну робочу область, і якщо у агента одночасно виконується кілька намірів, поточна робоча область автоматично перемикається відповідно до того, який намір виконується.

Наступна лекція буде присвячена програмуванню агентів і їх середовищ на платформі JaCaMo.