УДК 681.3.06

П.І. Андон, В.О. Дерецький

ЗАСОБИ КООРДИНАЦІЇ АГЕНТІВ У ПОШУКОВИХ АРХІТЕКТУРАХ WEB

Досліджується підхід до тематичного пошуку інформації на засадах агентних технологій з використанням онтологічних мов опису та координації поведінки агентів. Для реалізації мети застосування (задачі) агент відповідно до зазначеного підходу включає об'єкти, що забезпечують розподілене керування, і прикладні динамічно співробітничаючі об'єкти. Розглядаються модель та застосування тематичного пошуку інформації, технологія і засоби специфікації поводження агента, використовувані в мультиагентній системі, а також описана архітектура платформи для реалізації зазначених моделей.

Вступ

Відомо, що Web є великим репозиторієм інформації і сервісів будь-якого напрямку та призначення. В той же час така характеристика шкодить Web-технологіям. Якщо необхідно знайти точну інформацію, то на пошук і перегляд великої кількості Web-сторінок витрачається значна кількість часу. Безумовно, відповідні пошукові машини Інтернету допомагають прискорити процес інформаційного пошуку, але відсутність контексту для пошукових слів зменшує його ефективність. Ці проблеми частково вирішуються за допомогою адаптації XML-технології, яка розроблена та розвивається для забезпечення опису структури та семантики даних, але Web-сайти в більшості створюються з використанням засобів HTML.

Інша задача пов'язана з формуванням знань користувача в процесі пошуку інформації. Пошукові машини формують і впорядковують знайдений матеріал на основі пошукових слів (образів) користувача. Вищий пріоритет отримують сторінки, що містять знайдені слова, але контекст може зовсім не цікавити користувача. Пошукові слова пов'язані з множиною смислів. Саме контекст може бути помічником для визначення та вибору певного значення пошукового слова і визначення рангу знайдених результатів.

Для вирішення цих та інших семантико-пошукових задач використання агентного підходу дає нові якісні результати.

Агент визначається як комп'ютерна програма, здатна розумно діяти від імені користувача (іншої програми), для виконання задачі [1]. Агенти, подібно людям, співробітничають так, щоб їх сукупність могла поєднувати зусилля для досягнення поставленої мети. Агенти володіють наступними основними властивостями:

автономність — здатність функціонувати самостійно і здійснювати контроль за своїм станом і своїми діями;

інтелектуальна проактивність — спроможність діяти не просто у відповідь на зміни навколишнього середовища, а здатність брати ініціативу (формувати нові цілі);

мобільність — здатність до пересування для подальшого функціонування в різних операційних оточеннях;

адаптивність — спроможність вчитися.

На відміну від програми об'єктно-орієнтованого програмування агент як програма має власне керування у вигляді не тільки виконуваного коду і стану, але й виклику (початку виконання програми агента). Іншими словами, агенти самі визначають, коли і як їм діяти.

Для забезпечення здатності до взаємодії автономних агентів у відкритому і розподіленому середовищах необхідні стандартні механізми та специфікації. Мінімальні вимоги до таких механізмів наступні:

наявність засобів, якими агенти можуть зв'язуватися один з одним так, щоб мати змогу обмінюватися інформацією, делегуючи задачі керування, тощо;

засоби, за допомогою яких агенти можуть визначати один одного;

способи унікальної ідентифікації агента;

засоби взаємодії з користувачами;

засоби переміщення від однієї платформи до іншої.

Модель агента, за рекомендаціями FIPA, забезпечує нормативну структуру, у межах якої агенти можуть бути розгорнуті і працювати. FIPA-специфікація встановлює логічну модель для створення, реєстрації, місця розташування, зв'язку, переміщення та вилучення агентів.

У мультиагентних системах (МАС), складених з кількох автономних агентів, переговори є ключовою формою взаємодії, що дозволяє групам агентів досягти взаємної угоди по відношенню до деякого представлення чи мети плану. Процес переговорів агентів може ґрунтуватися на різних моделях, таких, як аукціони, протоколи контрактної мережі, дискусії тощо. В напрямку дослідження переговорів виділяють три головні теми [2]:

1) предмети переговорів — діапазон проблем, з яких повинно бути досягнуто згоди. Це можуть бути як прості проблеми (визначення цін на товар різних виробників), так і складні (наприклад, встановлення цін на товар заданої якості, планування часу доставки товару та ін.). У найпростішому випадку завдання переговорів полягає у визначенні структури та змісту угоди агентів. В результаті переговорів домагаються прийняти чи відхилити пропозицію. Існують і більш складні випадки переговорів, наприклад зміна значення проблеми в предметі переговорів через зустрічні пропозиції або зміна структури предмета переговорів (додаткові гарантії, інші терміни постачання тощо);

2) протоколи переговорів — набір правил, керуючих взаємодією агентів. Вони визначають склад припустимих типів учасників (наприклад, агентів, які ведуть переговори або мають до них відношення), фіксують стан переговорів, ініціюють події, що викликають переходи в інші стани і виконують дії учасників у визначених станах;

3) моделі міркування агентів — засоби, що забезпечують ухвалення рішення, за допомогою якого учасники намагаються досягти свою ціль. Складність моделі міркування визначається використовуваним протоколом, типом предмета у переговорів і діапазоном операцій, що можуть бути виконані.

Для практичних застосувань так само важлива низкорівнева фундаментальна інфраструктура механізму переговорів. Наприклад, будь-який протокол переговорів повинен враховувати можливість утрати повідомлень, вибування одного з партнерів переговорів тощо. Для успішного завершення переговорів усі сторони (агенти), які беруть у цьому участь, повинні ясно розуміти правила протоколу.

Очевидно, що комунікаційна мова може обмежувати міркування між агентами переговорів. Наприклад, мова KQML [3] накладає обмеження, за якого очікується, що агент, надсилаючи запит, сам вирішить, як одержувач повинен його обробити.

Однією з задач розробки протоколу є досягнення бажаного результату за мінімальну кількість комунікацій.

На даний час FIPA та іншими організаціями ведуться роботи зі стандартизації набору протоколів, на яких буде заснована координація взаємодії агентів [4]. Щоб дозволити агентам взаємодіяти один з одним, розроблено ряд мов їхнього спілкування [1, 5]. Ці мови забезпечують набір спеціальних повідомлень (performatives), заснованих на мовних актах. Хоча такі performatives можуть характеризувати типи повідомлення, практично немає ефективних мов для вираження змісту повідомлення, який дозволяє агентам "розуміти" один одного.

Однією з важливих у застосуваннях МАС є проблема розподілу ресурсів. Для її вирішення використову-

ються прийняті в бізнесі економічно обґрунтовані підходи та ринкові механізми, що базуються на основних формальних моделях.

У задачах, де агенти використовуються для керування обмеженими ресурсами (праця, сировина, товари, гроші), для укладання угод у переговорах передбачається, що ціни й інші параметри моделі загально відомі. На аукціонах є центральний аукціоніст, через якого здійснюється координація. Отже, агентам для обміну необхідна тільки мінімальна кількість інформації.

Інший клас задач — це пошукові задачі, в моделях яких загальновідомими вважаються значення (смисл) пошукових образів.

У наступних розділах описані моделі та підходи створення агентів і МАС на основі засобів керування та координації для пошукових технологій. Використовується онтологічна мова опису та координації інтелектуальних агентів COOL — Control Oriented Ontology Language [6, 7].

COOL у МАС забезпечує створення концептуальної моделі координації у вигляді певних структур — "розмов", що включають комунікативні дії між агентами. Ця модель дозволяє представити об'єкти і структури керування та побудувати МАС.

COOL успішно використовується в застосуваннях, розроблених для промислових підприємств, зокрема для рішення задач інтеграції ланцюгів постачань [6]. У даній статті пропонується використання засобів COOL у пошукових архітектурах.

Функціонально COOL слід розглядати як [7]

- мову, що встановлює стандарти координації між агентами, що співробітничають;
- оболонку для виконання координованих дій у МАС;
- інструмент для проектування, експериментування і перевірки правильності протоколів координації;
- інструмент для поетапного розвитку (наповнення) знань про координацію агентів.

COOL дозволяє задати безліч агентів, що співробітничають взаємодіючи один з одним шляхом використання заданих протоколів координації, названих класами розмови (conversation classes). У результаті такого створення породжується діаграма станів, у якій переходи від одного стану до іншого визначені в структурах, які називаються правилами розмови (conversation rules). Кожне правило може бути виконано тільки за певних умов — отриманих повідомлень від інших агентів. Результат виконання правила веде до зміни стану розмови і викликає деяку дію. Кожен агент забезпечується різними типами планів координації відповідно до тієї ролі, що він буде грати в застосуванні.

Однією з головних задач у МАС є досягнення координованого поводження між агентами. В ідею розробки COOL був закладений той факт, що проблема координації виникає за наявності знань про процеси, які відбуваються при взаємодії агентів. Рішення цієї проблеми знаходиться в компетенції всієї МАС, а не окремо взятого агента.

COOL містить кілька припущень щодо способу, яким агенти зможуть досягти координованого поводження:

- автономні агенти мають власні плани, згідно яких вони переслідують кожен свою мету;
- індивідуальні плани агентів і знання про мультиагентне середовище, у якому вони знаходяться, точно представляють взаємодії з іншими агентами. Припускається, що ці взаємодії відбуваються за допомогою обміну повідомленнями;
- агенти не можуть передбачити точну поведінку інших агентів, але можуть виділити класи альтернативних поводжень, що очікуються. Отже, плани агентів являють собою умовні вираження над можливими діями і/чи реакціями інших агентів;
- плани агентів можуть бути незавершеними чи неточними, і необхідні знання для їхнього розширення або виправлення. Точні плани можуть

стати доступними тільки протягом виконання. З цієї причини агенти здатні розширювати і змінювати свої плани в процесі виконання.

Найбільш важливою конструкцією мови є класи розмови, що визначають стани й асоційовані правила для отримання повідомлення, модифікації локального стану, перевірки умови виконання і передачі повідомлення. Агенти мови COOL можуть володіти декількома класами розмови для керування взаємодіями з іншими агентами. Екземпляри класів розмови, що називаються розмовами (conversations), зберігають свій стан виконання стосовно класу розмови. Агенти можуть мати одночасно кілька активних розмов, і існують механізми керування, що дозволяють їм припиняти розмову при чеканні інших розмов, щоб досягти визначених стадій. Таким чином, агенти можуть динамічно створювати ієрархії розмов.

1. Тематичний пошук інформації з використанням програмних агентів

Запропонований підхід є розвитком досліджень у галузі пошукових технологій [8, 9] з використанням агентних методів і засобів опрацювання семантики текстової інформації.

Взаємодію агентів розглянемо на прикладі архітектури пошукової системи, що містить:

агент теми (Topic Agent);

агент контексту (Context Agent);

пошуковий агент (Search Agent).

Агент контексту формує контекст поточної Web-сторінки за допомогою алгоритму класифікації, що використовує список тематичних категорій і для кожної категорії список упорядкованих пошукових слів та словосполучень. Такий список тематичних категорій складається попередньо, а взаємодія з ним забезпечується агентом теми. Кожна сторінка індексується, для неї формуються вагові коефіцієнти слів за частотними показниками. Отримані значення нормалізуються відповідно до розмірів сторінки. Розраховується ступінь подібності сторінки до тематичної категорії за формулою

$$Sim(i) = \sum_{j=1}^{n} W(j) \frac{w(i,j)}{\sum_{k=1}^{m} w(k,j)},$$

де $Sim(i)$ — ступінь подібності сторінки та тематичної категорії (i), що відповідає контексту сторінки; $W(j)$ — значення ваги пошукового слова (j); n — кількість слів на сторінці; $w(i,j)$ — вага слова (i) тематичної категорії (i). Сторінка може відповідати кільком тематичним категоріям, які упорядковуються за ваговими коефіцієнтами.

Пошуковий агент перевіряє результати пошукової машини та впорядковує їх за даними, що формуються агентом контексту. З цією метою для кожної сторінки визначається вага кожної теми, що розраховується за формулою

$$Sc(i) = \sum_{k=1}^{n} W(k),$$

де $Sc(i)$ — вага (i) теми; $W(k)$ — вага відповідної словоформи. Визначені теми упорядковуються за отриманим значенням.

Зміст взаємодії контекстного та пошукового агентів полягає в наступному. Для визначення контексту користувач надає первинний документ або сукупність документів (файли, Web-сторінки тощо), зміст яких відповідає його інтересам. Для цих документів визначається значення $\{Sim(i)\}$.

За результатами роботи пошукового агента обчислюється значення $\{Sc(i)\}$. Шляхом переговорів агентів визначається відповідність знайденого матеріалу контексту та здійснюється його упорядкування за ваговими коефіцієнтами контексту.

Взаємодія агентів відбувається на кількох рівнях.

Перший рівень стосується інформаційного змісту спілкування агентів. Порція інформації, переданої на цьому рівні, являє собою висловлення (факт), наприклад “(*search* $T(i)$)” (знайти документи по темі (i)).

Другий рівень визначає наміри агентів. Той же самий інформаційний

зміст може бути повідомлений з різними намірами. Наприклад:

- (*ask* (*search T*(*i*))) — відправник запитує одержувача, чи згаданий факт істиний;
- (*tell* (*search T*(*i*))) — відправник повідомляє про свій намір одержувачу;
- (*achieve* (*search T*(*i*))) — відправник пропонує одержувачу зробити даний факт одним з його намірів;
- (*deny* (*search T*(*i*))) — відправник повідомляє, що цей факт більше не є наміром.

Третій рівень забезпечує угоди, які агенти спільно використовують при обміні повідомленнями. Існування загальнодоступних угод уможливлює координацію агентів, наприклад, при здійсненні переговорів про їхні цілі та дії.

Нарешті, четвертий рівень взаємодії має відношення до внутрішньої архітектури агентів.

Співробітництво і координація інтелектуальних агентів відбуваються завдяки передачі повідомлень між агентами засобами KQML (Knowledge Query and Manipulation Language — мова запитів і керування знаннями), розробленої як універсальна мова для вираження таких намірів, причому всі агенти інтерпретують їх однаково. KQML підтримує зв'язок через явні лінгвістичні конструкції (performatives). KQML ґрунтується на структурі мовного акта (speech act framework), розробленого філософами і лінгвістами для пояснення людського спілкування [3].

COOL допускає існування стандартного набору мовних актів, що визначають комунікативні дії. Ці мовні акти представляються як performatives — припустимі дії, які агенти можуть виконати при передачі повідомлень один одному. До стандартних мовних актів у COOL був доданий ряд мовних актів більш високого рівня:

- Propose (запропонувати) — використовується, щоб запропонувати агенту досягти підціль. Наприклад: (*propose*: *content* (*search T*(*i*))) (*time* "19-*feb*-2004")).
- Counter-Propose (запропонувати назустріч) — зустрічна пропозиція (контрпропозиція) — це інша підціль, що частково задовольняє початкову мету. Використання цього мовного акта може привести до послідовності контрпропозицій. Наприклад: (*counter-propose*: *content* (*search T*(*i*) (*time* "20-*sep*-2003"))).
- Accept and Reject (прийняти і відхилити) — використовуються, щоб повідомити про прийняття чи відхилення пропозиції або контрпропозиції. Відхилення ініціює нову стадію переговорів.
- Cancel (скасування) — скасовує попередньо прийняту пропозицію чи контрпропозицію.
- Satisfy (задовольнити) — агент повідомляє, що необхідна ціль була досягнута. Наприклад: (*satisfy*: *content* (*counter-propose*: *content* (*search T*(*i*))) (*time* "20-*sep*-2003")).
- Fail (зазнати невдачі) — агент повідомляє, що виконання заданої цілі зазнало невдачі.

Для побудови MAC COOL надає ряд конструкцій, у ролі яких можуть виступати:

- агенти;
- диспетчери розмови;
- класи розмови;
- правила розмови;
- правила відновлення розмови;
- правила продовження розмови;
- асоціації.

Коротко розглянемо призначення і семантику кожної конструкції.

Агенти. Як згадувалося вище, агент розглядається як об'єкт, що діє автономно, цілеспрямовано і виконує задані функції. Він функціонує, ґрунтуючись на індивідуальних і загальнодоступних знаннях, переконаннях і намірах. Агент у COOL — це об'єкт, котрого можна запрограмувати і який може обмінюватися повідомленнями в межах структурованих планів з іншими агентами. Ці плани задаються в класах розмови і виконуються екземплярами розмов. Класи розмови визначаються разом з агентом.

Агент визначається шляхом призначення йому імені, установки змінних, які утворюють його локальну базу

даних, і встановлення початкової розмови (initial conversation). Після визначення агента виконується його початкова розмова і впродовж цього він знаходиться в активному стані. Якщо початкова розмова не була визначена, то агент залишається в стані очікування доти, поки не одержить повідомлення від іншого агента.

Диспетчери розмови. Агенти здійснюють розмови з іншими агентами чи виконують дії усередині свого середовища. Співробітництво агентів існує в локальних або віддалених середовищах виконання. Диспетчер розмови керує виконанням агента всередині середовища. Агенти в COOL повинні бути виконані під контролем керуючої програми. Диспетчер розмови визначає набір агентів, якими він керує, і виконує різні опції трасування. У COOL можна визначити декілька диспетчерів розмови, кожного зі своїм власним набором агентів. Призначення середовища виконання полягає в тому, щоб "запустити" агентів шляхом передачі повідомлень і виконати планування агентів. Ці середовища виконання існують на різних машинах, проте повідомлення передаються між ними в такий же спосіб, як і на одній машині. Застосування, що складається з агентів мови COOL, може виконуватися як на одній машині, так і на декількох машинах у мережі. Кожне середовище виконання визначається своїм диспетчером розмови, що координує виконання агентів, зв'язаних з цим середовищем [5].

Клас розмови — це правило, засноване на описі того, що повинен робити агент у певних ситуаціях (наприклад, при отриманні повідомлення). COOL забезпечує зв'язок класів розмови з агентами, окреслюючи таким чином, які види розмов кожен агент може обробляти. Клас розмови визначає доступні правила розмови, механізм роботи і локальну базу знань, що містить стан розмови. Правила розмови індексовані на кінцевій множині значень спеціальної змінної *current-state* (поточний стан). Необхідно підкреслити, що клас розмови визначає тільки початкові і кінцеві стани розмови. Усі проміжні стани визначаються в правилах розмови як параметр у слоті *:next-state.*

Нижче наведено приклад класу розмов (можлива діаграма переходів (Conversation Plan)) для керування розмовою в застосуванні пошуку релевантної контексту інформації між агентом контексту (Context agent) і агентом пошуку (Seacher agent) мовою COOL.

(*def-conversation-plan* '*serch-conversation*
:*content-language* '*list*
:*speech-act-language* '*kqml*
:*initial-state* '*start*
:*final-states* '(*rejected failed satisfied*)
:*control* '*interactive-choice-control*
:*rules* '((*start cc*-1)
(*simi cc*-3 *cc*-2)
(*working cc*-5 *cc*-4 *cc*-3)
(*item cc*-9 *cc*-8 *cc*-7 *cc*-6)
(*asked cc*-10)
(*accepted cc*-12 *cc*-11)))

На рис. 1 діаграма переходів зображена у формі графа $G(S, T)$, де S — множина станів, а $T \subset S^2$ — множина переходів. Дуга $(v,t) \subset T$ існує, якщо існує правило розмови для переходу з поточного стану v у наступний стан t. У діаграмі переходів усі стани повинні бути досяжні з початкового стану.

Правило може бути визначене як завершене чи незавершене. Завершені виконуються без повідомлення користувача, а в незавершених правилах можуть бути відсутні необхідні дані, наприклад точний склад повідомлення, яке буде передано. Коли система зіштовхується з незавершеним правилом, вона може не знати, яке правило повинне виконатися в даному стані (відсутня умова) чи що повинно відбутися при виконанні правила (відсутня дія), і видається відповідне повідомлення користувачеві для уточнення ситуації.

Приклад правил розмови мовою COOL:

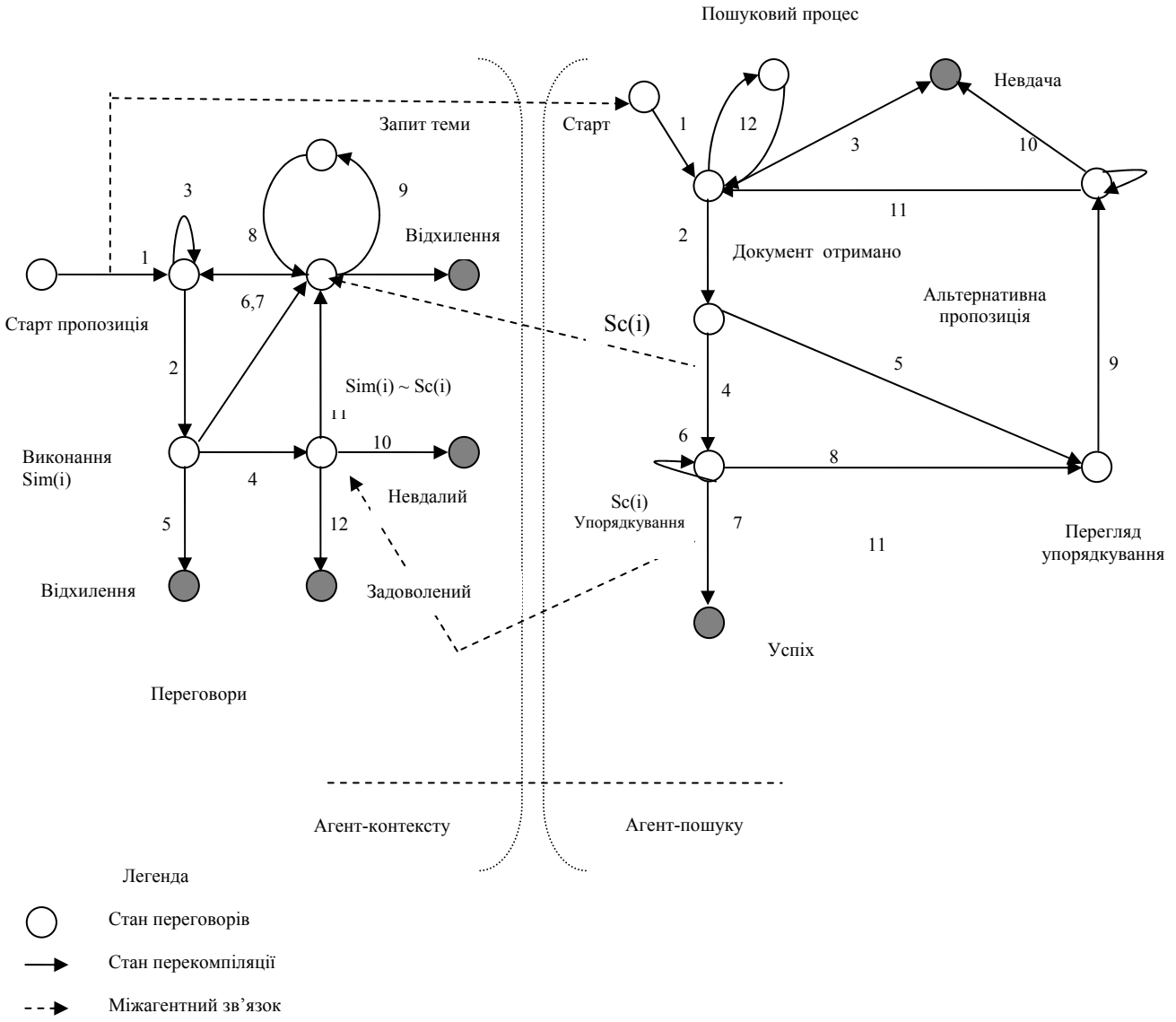

Рис. 1. Діаграма переходів агентів Context і Searcher у формі графа

(*def-conversation-rule* '*sim*-1
:*current-state* '*start*
:*received* '(*propose*: *sender*
:*content* (*item-order*
:*has-line-item* ?*li*))
:*next-state* '*order-received*
:*transmit* '(*tell*: *sender* ?*agent_search*
:*receiver agent_context*
:*content* '(*working on it*)
:*conversation* ?*convn*)
:*do* '(*update-var* ?*conv* '?*order* ?*message*))

2. Програмне середовище для побудови пошукової мультиагентної системи

Програмне середовище JADE (Java Agent Development Framework) призначене для розробки агентів, MAC і застосувань, що відповідають стандартам FIPA для інтелектуальних агентів. Середовище складається з двох основних частин: власне FIPA-сумісної агентної платформи і засобів розробки агентів Java.

Середовище JADE написане мовою Java і складається з бібліотеки Java-класів (packages), що надають розроблювачам прикладних програм готові фрагменти функціональних можливостей і абстрактних інтерфейсів [10, 11].

JADE постачається з набором інструментальних засобів, які спрощують адміністрування і розробку. Такими інструментальними засобами є (рис. 2):

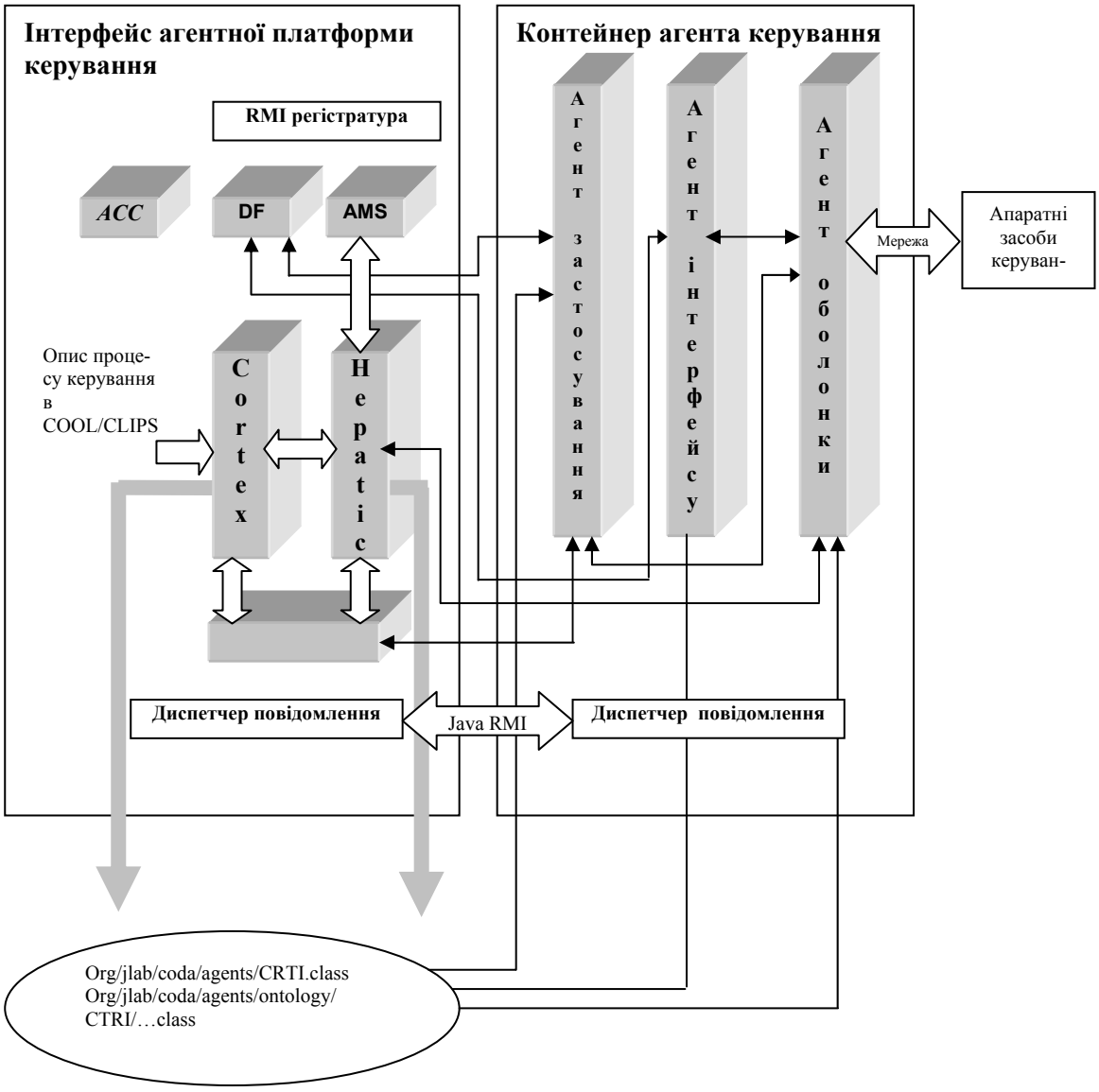

Рис. 2. Архітектура програмного середовища JADE для побудови МАС

• *Remote Management Agent (RMA)* — графічна оболонка, призначена для керування агентною платформою. JADE підтримує когерентність між декількома запущеними RMA шляхом многоканальної передачі даних між ними. Крім того, оболонка RMA здатна запустити й інші інструментальні засоби JADE;

• *Dummy Agent* — графічний інтерфейс користувача (GUI), який є агентом середовища JADE. GUI дозволяє скласти повідомлення мовою ACL, послати їх іншим агентам, відобразити список усіх відправлених і/чи отриманих повідомлень мовою ACL із вказівкою інформації про час, щоб записати і повторити розмову агента;

• *Sniffer* — агент, що може перехопити повідомлення мовою ACL в процесі їх виконання і графічно відобразити їх у вигляді послідовності прямих ліній. Це важливо для налагодження і дозволяє спостерігати, як відбувається обмін повідомленнями мовою ACL між агентами;

• *SocketProxyAgent* — простий агент, який функціонує в якості двунаправленого шлюзу між платформою JADE і звичайним TCP/IP-підключенням. ACL-повідомлення, що передаються всередині JADE у власному

внутрішньому форматі, конвертуються в ASCII-строки і відправляються через сокет-підключення. І навпаки, ACL-повідомлення можуть бути передані і конвертовані через TCP/IP-підключення в платформу JADE. Цей агент може використовуватися, наприклад, при забезпеченні взаємодії з Java-аплетами всередині мережевого браузера (засобу навігації по мережі і перегляду інформації);

- *DF GUI* — графічний інтерфейс користувача, за допомогою якого створюється складна мережа областей и підобластей "жовтих сторінок". Цей GUI дозволяє простим і інтуїтивним способом керувати базою знань DF, поєднувати DF у федерації с другими DF, дистанційно керувати базою знань DF-предків і DF-нащадків.

Агентна платформа JADE може бути розподілена між декількома комп'ютерами. Віртуальна машина Java — це основний контейнер агентів, що забезпечує середовище виконання агента і дозволяє кільком агентам одночасно виконуватися на тому ж самому хості. Агенти реалізовані як нитки (thread) Java:

1) GUI дозволяє керувати декількома агентами й агентними контейнерами з віддаленого хоста;

2) засоби налагодження полегшують розробку мультиагентних застосувань, заснованих на JADE;

3) забезпечується мобільність агента, включаючи його стан і кодування;

4) підтримується одночасне виконання множини дій завдяки використанню моделі поводження агента;

5) автоматично активізуються основні компоненти AMS, DF і ACC при запуску агентної платформи;

6) забезпечується запуск множини DF для реалізації багатодомених застосувань, де доменом є логічний набір агентів, послуги яких "рекламуються" через загальний DF. Кожен DF успадковує GUI і всі стандартні можливості;

7) забезпечується ефективна передача ACL-повідомлень усередині однієї й тієї ж агентної платформи. Повідомлення передаються в закодованому вигляді як об'єкти Java. При перетинанні границь платформи повідомлення автоматично конвертується в/з FIPA-сумісний синтаксис. Це перетворення є прозорим для розроблювача, що має справу тільки з Java-об'єктами;

8) надається бібліотека протоколів взаємодії FIPA, готових до використання;

9) забезпечується автоматична реєстрація і дереєстрація агентів у AMS;

10) ведеться FIPA-сумісна служба імен: при запуску агент одержує свій глобальний унікальний ідентифікатор *(Globally Unique Identifier, GUID);*

11) забезпечується підтримка мов змісту й онтологій для конкретних застосувань.

Для створення МАС використовуються наступні основні класи: Agent і Behavior.

Клас Agent (агент) являє собою загальний базовий клас для агентів, обумовлених користувачем. С точки зору розроблювача, агент JADE — звичайний екземпляр Java-класу, що розширює базовий клас Agent. Мається на увазі успадкування властивостей для здійснення основних взаємодій з агентною платформою (реєстрація, конфігурація, віддалене керування тощо) і основний набір методів, що можуть викликатися для реалізації заданого поводження агента.

Обчислювальна модель агента є багатозадачною і паралельною, у якій задачі (чи поводження) виконуються одночасно. Кожна функціональна можливість і/чи сервіс, наданий агентом, повинні бути реалізовані як одне чи кілька поводжень. Внутрішній планувальник, схований від розроблювача, автоматично керує плануванням поводжень.

На рис. 3 наведено життєвий цикл агента відповідно до специфікації FIPA. Агент JADE може знаходитися в одному з кількох станів, представлених у класі Agent наступними об'єктами:

– AP_INITIATED: об'єкт класу Agent створений, але поки незареєстрований у AMS, тому він не має ні

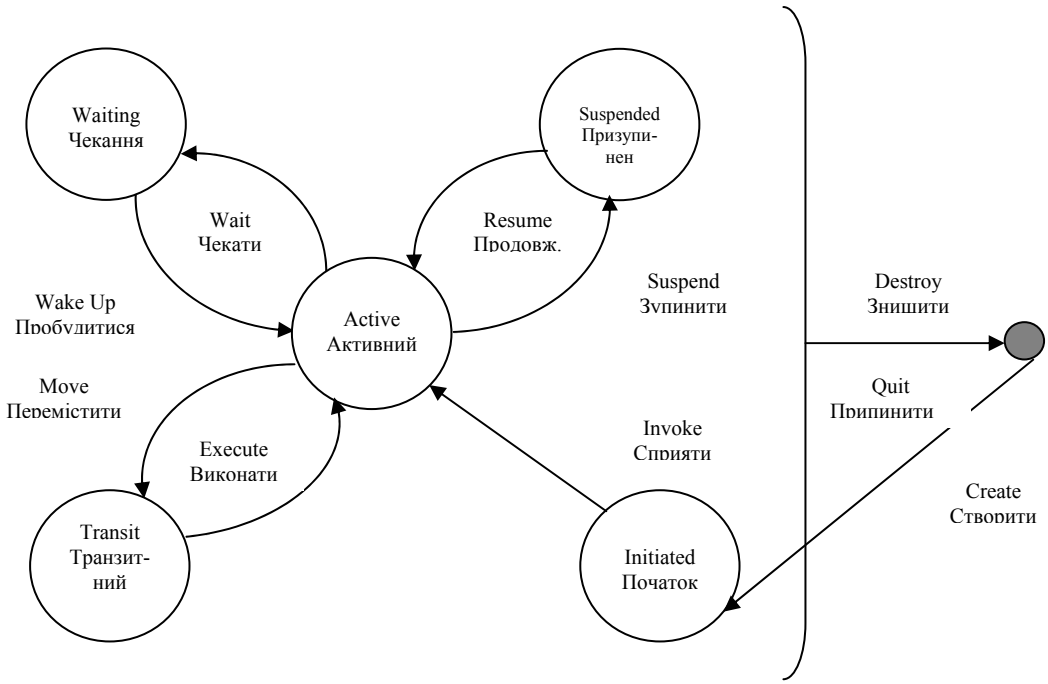

Рис. 3. Життєвий цикл агента відповідно до специфікації FIPA

імені, ні адреси і не може вести переговори з іншими агентами;

— AP_ACTIVE: об'єкт Agent зареєстрований у AMS, має постійне ім'я й адресу і може використовувати всі сервіси JADE;

— AP_SUSPENDED: об'єкт Agent у даний момент часу зупинений і не виконує ніякої дії (поводження);

— AP_WAITING: Agent блокований, його внутрішній стан — без дії і він пробудиться при виконанні деякої умови (наприклад, надходження повідомлення);

— AP_DELETED: внутрішній стан — завершений, і агент не зареєстрований у AM;

— AP_TRANSIT: мобільний агент уводить цей стан для того, щоб, поки агент переміщається до нового місця розташування, система продовжувала накопичувати і зберігати повідомлення, що будуть надіслані йому, коли він прибуде в нове місце;

— AP_COPY: внутрішній стан використовується платформою JADE для створення копії агента;

— AP_GONE: внутрішній стан використовується платформою JADE для повідомлення про те, що мобільний агент перемістився до нового місця розташування й установив стійкий стан.

Клас Behaviour (поведінка). Розроблювач визначає дії агента шляхом специфікації його поводження. Агент здатний виконувати декілька паралельних задач у відповідь на різні зовнішні події. Для того щоб керування агентом було ефективним, кожен агент JADE складається з окремого потоку виконання і всі його задачі (наміри) повинні бути реалізовані як об'єкти класу Behaviour. Розроблювач при завданні задачі агентові повинен визначити один чи кілька похідних класів від базового класу Behaviour і додати певні поводження до списку його задач. Клас Agent надає два методи: addBehaviour і removeBehaviour, що дозволяють керувати чергою задач агента, а саме додавати або видаляти поводження. Поводження агента можуть бути додані всякий раз, коли це необхідно, а не тільки в межах методу Agent.setup().

Планувальник, реалізований основним класом Agent і „схований" для програміста, виконує циклічну політику планування між усіма поводженнями, доступними в черзі на виконання. Поводження може бути заблоковано, й агент очікує одержання повідомлення.

Щоб уникнути активного чекання повідомлень (і, як наслідок, витрату процесорного часу), кожне поводження може бути заблоковано. Метод *block*() поміщає поводження в чергу заблокованих, і воно відновлюється, як тільки відбувається повернення з методу *action*(). Усі заблоковані поводження упорядковуються у разі прибуття нового повідомлення. Крім того, об'єкт класу Behaviour може блокувати себе на визначений обмежений час.

Висновки

Розглянуто новий підхід до розробки засобів взаємодії та координації програмних агентів на засадах онтологічних мов. Підхід спрямований на широке коло застосувань, зокрема в напрямку пошукових Web-сервісів опрацювання змісту текстової інформації.

1. *Jennings N.R., Sycara K., Wooldridge M.* A Roadmap of Agent Research and Development // Autonomous Agents and Multi-Agent Systems. — 1998. — **1.** — P. 275—306.
2. *Chen Ye, Peng Yun, Finin T., Labrou Ya., Cost S.* A negotiation-based multi-agent system for supply chain management // Working Notes of the Agents'99 Workshop on Agents for Electronic Commerce and Managing the Internet-Enabled Supply Chain, Seattle, WA, April 1999. — P. 36—41.
3. *Finin T., Weber J., Wiederhold G. et al.* Specification of the KQML Agent Communication Language // The DARPA Knowledge Sharing Initiative, External Interfaces Working Group, 1992. — 56 p.
4. *Gyurjyan V., Abbott D., Heyes G., Jastrzembski E., Timmer C., Wolin E.* FIPA agent based network distributed control system // TJNAF, VA 23606, Newport News, USA. March 2003. — 5 p.
5. *Singh M.P.* Towards a Formal Theory of Communication for Multiagent Systems // Conf. Artificial Intelligence, Sydney, Australia, 1991. — P. 69—74.
6. *Barbuceanu M.* COOL: A Language for Representing and Executing Coordinated Behavior in Multi-Agent Systems // Enterprise Integration Laboratory, Department of Industrial Engineering, User Manual, University of Toronto, 1996. — 56 p.
7. *Barbuceanu M.* Coordination Protocols For Multi-Agent Supply Chains // Teamwork, Enterprise Integration Laboratory, Report and Formal Specification, University of Toronto, 1996. — 78 p.
8. *Андон П.І., Дерецький В.О.* Процесори пошуку та аналізу природномовної текстової інформації в аналітичних системах // Пробл. программирования. — 2001. — N 3—4. — С. 144—165.
9. *Дерецкий В.* Об одном подходе к обработке естественно-языковых данных на основе анализа семантических сетей // Первая Всерос. науч. конф. „Электронные библиотеки: перспективные методы и технологии, электронные коллекции", 18—22 октября 1999 г., Санкт-Петербург, 1999. — С. 100—103.
10. *Haase R.* Building a Generic User Agent for Multi-Agent Integrated Enterprise // Enterprise Integration Laboratory, University of Toronto, 1997. — 37 p.
11. *Bellifemine F., Poggi A., Rimassa G.* JADE — A FIPA-compliant agent framework // Proceeding of PAAM99, London, April 1999. — P. 97—108.

Про авторів

Андон Пилип Іларіонович,
член-кор. НАН України, директор

Дерецький Валентин Олександрович,
канд. фіз.-мат. наук, провідний науковий співробітник

Місце роботи авторів:
Інститут програмних систем НАН України,
Просп. Академіка Глушкова, 40,
Київ-187, 03680, Україна
Тел. (044) 266 4342
E-mail: dva@isofts.kiev.ua