УДК 681.3.06

ДИНАМІЧНА КООРДИНАЦІЯ ПРОГРАМНИХ АГЕНТІВ НА ОСНОВІ ОНТОЛОГІЧНОЇ СТРУКТУРИ

С. Ремарович

Інститут програмних систем НАН України
03187, Київ-187, проспект Академіка Глушкова, 40,
тел.: +38 044 526 6249; rem@isofts.kiev.ua

Координація – це процес управління можливими взаємодіями між діями і процесами; механізм управління такими взаємодіями відомий як процес координації. Координація може розглядатися як процес, завдяки якому індивідуальні рішення агентів приводять до успішних спільних рішень для групи агентів. Успішний процес координації запобігає негативним взаємодіям (наприклад, запобігання одночасного доступу двох процесів до обмеженого ресурсу) і можливо покращить позитивні взаємодії (наприклад, гарантувати не дублювання дій). Ефективні механізми координації вимагають сумісного використання знань про процеси, ресурси і їх властивості в різнорідному оточенні. В роботі пропонується підхід, в якому для динамічної координації використовується онтологія, яку апріорі розроблено з метою визначення семантики мережних процесів.

Coordination is the process of managing the possible interactions between activities and processes; a mechanism to handle such interactions is known as a coordination regime. Coordination can be regarded as the process by which the individual decisions of the agents result in good joint decisions for the group. A successful coordination regime will prevent negative interactions occurring (e.g., by preventing two processes from simultaneously accessing a non-shareable resource), and wherever possible will facilitate positive interactions (e.g., by ensuring that activities are not needlessly duplicated). Effective coordination mechanisms require the sharing of knowledge about activities, resources and their properties in a heterogeneous environment. In this paper an ontological approach to coordination is appropriate.

Вступ

Координація є однією з основних проблем в системах, що компонуються з багатьох процесів, які взаємодіють між собою. Таким процесам потрібно координувати свою активність, за можливістю ці активності можуть взаємодіяти одна з одною. Наприклад, два процеси використовують ресурс, який не є спільним. Якщо обидва процеси намагаються використовувати ресурс одночасно, виникають проблеми пошкодження ресурсу. Тому потрібно координувати діяльність процесів, які використовують цей ресурс. Хоча такий сценарій зображає добре відомий вид можливої взаємодії, є багато інших менш очевидних випадків, де координація може бути взаємовигідною. Наприклад, два наукові процеси, виконуючи деяке обчислювальне завдання, потребують результати проміжного обчислення; координація має значення для них, щоб прийняти політику активного обміну інформацією, яка може принести користь іншим процесам. Тут координація не потрібна для агентів, які успішні в своїх завданнях, але є глобальна мета, яка досягається за допомогою прийняття такої політики.

Координація в обмеженому значенні синхронізації (наприклад, одночасний доступ до ресурсу) – це центральна тема дослідження в задачах розпаралелювання [1]. Проте домінуючим підходом до координації є механізм координації в структурі системи (наприклад за допомогою семафорів, моніторів або замків). У більш відкритих системах, де процеси і ресурси, які входять до системи, невідомі на етапі розробки, такий підхід неможливо використати. В таких системах бажано дозволити релевантним процесам повідомляти про свої наміри щодо майбутніх дій і використання ресурсу, і одержавши їх, обдумати координацію під час виконання з метою запобігання негативних взаємодій і сприяння позитивних взаємодій. Це динамічний підхід до координації, тому що вимога координації управляється процесом виконання, а не процесом розробки. Зазначимо, що в цьому підході комунікація вимагає погодженого загального словника для координації з точно визначеною семантикою, тому ми можемо говорити про онтологічний підхід до динамічної координації.

1. Огляд проблеми координації

Координація - це проблема управління при кооперативній роботі. Багато робіт про координацію пов'язані з мульти-агентними системами [2, 3]. Будемо називати процеси, які потрібно координувати, "агентами". Проблема координації означає управління взаєминами між діями агентів [4]. Координація вельми важлива, якщо процеси, до яких залучені агенти, можуть взаємодіяти будь-яким шляхом. Розглянемо наступні приклади.

Обмежений ресурс. Існує ресурс, який декілька агентів бажають використати. Але цей ресурс може використовуватися тільки одним агентом у момент часу. В цьому випадку дії агентів потрібно координувати.

Залежність процесів. Виконання дії агента *A* не може виконуватися, поки не завершиться дія агента *B*, тобто активність агента *A* залежить від активності агента *B*.

Збільшення ефективності. Результат виконання дії агента *A* постачається зацікавленому агенту *B*, тобто заощаджується час агента *B*. В даному випадку, дії агентів не потрібно строго координувати, тому що ту ж саму операцію може виконати агент *B*, не вступаючи в конфлікт.

Зазначимо, що координація, визначена в цьому випадку, належить до категорії добре відомого поняття синхронізації [1]. Синхронізація загалом має відношення швидше до обмеженого випадку, в якому процеси не деструктивно взаємодіють один з одним. Поняття координації є набагато ширшим. Стандартні рішення проблем синхронізації залучають режими координації в програмному коді (наприклад, Java метод *synchronized*). Проте, у великих, динамічних, відкритих системах такі режими дуже обмежені. Бажано, щоб обчислювальні процеси були здатні *міркувати* про проблеми координації у системі й вирішувати ці проблеми *автономно*.

Для того, щоб побудувати агентів для семантичних мережних застосувань, які можуть динамічно міркувати про проблеми координації, потрібно спочатку ідентифікувати можливі відношення взаємодії цих застосувань. Звідси випливає, що потрібно отримати і формально визначити ці відношення. В роботі [5] відношення між діями поділяються на *позитивні* і *негативні*. Позитивні взаємовідношення – "всі ті взаємини між двома планами, від яких може отримати деяку користь один або обидва плани агентів, за допомогою їх об'єднання" [6]. Такі відношення можуть бути визначені «запитом» (*requested*) (я прошу вас допомогти моїм діям) або визначені «не запитом» (*non-requested*) (це трапляється, коли працюючи разом, ми можемо досягти рішення, яке як мінімум краще для одного з нас, без створення іншому чогось гіршого). Розрізняють три види взаємовідношень, що визначені «не запитом».

Взаємовідношення рівності дії. Два агенти планують виконати ідентичну дію, і, усвідомлюючи це, один з них може виконати дію сам і, таким чином, збереже зусилля іншого агента.

Взаємовідношення наслідку. Побічним ефектом виконання дії в плані одного агента є досягнення однієї з цілей іншого агента, що полегшує, таким чином, необхідність явного досягнення її.

Взаємовідношення прихильності. Деяка частина плану одного агента сприяє досягненню однієї з цілей іншого агента, тобто можливо полегшує її виконання (наприклад, за допомогою досягнення вхідної умови однієї з дій).

Іншою головною частиною роботи в цій проблемі є часткове глобальне планування [7]. Основна ідея часткового глобального планування полягає в тому, що агенти розвиваються й обмінюються планами локальної активності для того, щоб ідентифікувати можливі взаємодії (позитивні або негативні) [8]. Розрізняють п'ять способів для координації активності.

Оновлення нелокальних розрізів. Агенти мають тільки локальні представлення активності, і таким чином, розділяючи інформацію, можливо допомогти їм досягти більш широких представлень.

Повідомлення результатів. Агенти можуть обмінюватися результатами трьома різними способами. Перший спосіб полягає в тому, що агенти обмінюються результатами, які вельми важливі, щоб задовольнити зобов'язання. Другий спосіб передбачає пересилку всіх результатів. Третій спосіб полягає в тому, щоб відправити результати тим, хто в них зацікавлений.

Обробка простої надмірності(redundancy). Надмірність означає дублювання роботи. Якщо надмірність виявлена у мульти-агентній системі, то вибирається випадково один агент, щоб виконати завдання. Результати потім транслюються іншим зацікавленим агентам.

Обробка негативних (hard) взаємин координації. Негативні взаємини координації – це такі взаємини, які загрожують успішному завершенню дій. Коли такі взаємини трапляються, дії агентів упорядковуються так, щоб розв'язати проблему.

Обробка позитивних (soft) взаємин координації. Це взаємини, які не є "критичними", але які можуть поліпшити загальну продуктивність. Коли вони зустрічаються, то має місце впорядкування черги, але з вищим ступенем "узгодженості": якщо не має можливості змінити графік, то система не турбується про це.

Координація – це важливий аспект сервісних обчислень, в якому приймають участь незалежні і, можливо, суперечливі агенти. Для координації використовують два підходи:

– в OWL-S [9] і WSMO [10] взаємодія і композиція процесів моделюються як послідовність виконуваних дій (workflow), яка визначена апріорі і яка виконується компонентом workflow;

– координація, яка базується на обміні повідомлень серед агентів і визначається терміном «хореографія» [11].

2. Використання онтології для координації агентів

Основна ідея використання онтології для координації полягає в тому, щоб надати можливість агентам міркувати про відношення їх дій до дій інших агентів, тобто відповісти на наступні питання:

– яка активність координується (coordinable)?

– які відношення координації мають активності одна до одної?

Коротко розглянемо ключові концепти онтології та слоти, які пов'язані з цими концептами, відношення між концептами і аксіоматику. Приклад онтології координації показано на рис. 1.

2.1. Агенти. Початковий концепт онтології пов'язаний з агентами, що визначені в системі, тобто, які виконують дії в системі і повинні координуватися. В онтології координації агенти представлені тільки одним слотом – *id*, який є рядковим представленням унікального імені для агента (наприклад, URI).

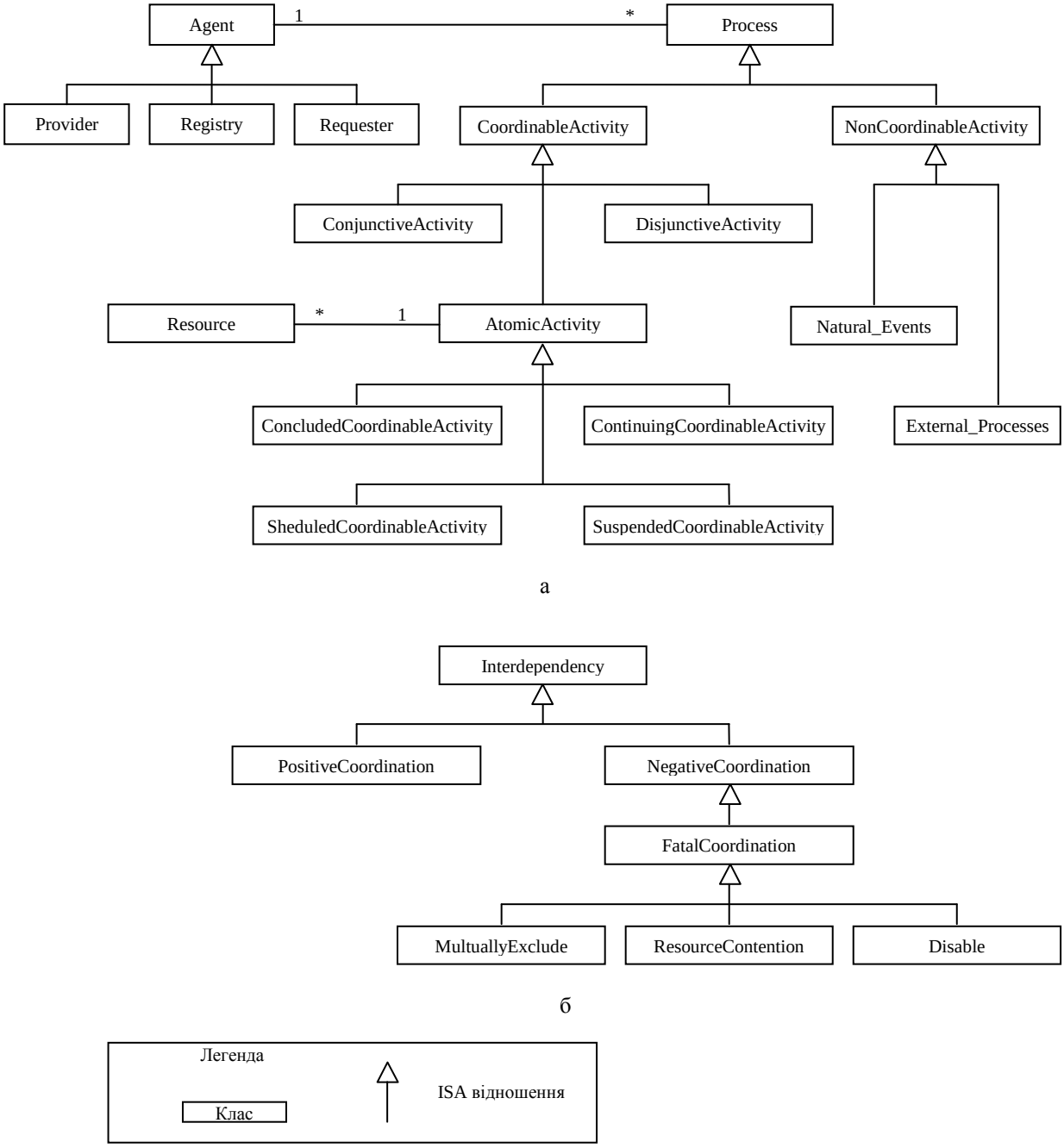

Рис. 1. Фрагмент онтології координації

2.2. Процеси і активності (дії). Наступний концепт – *Process*. Процес – це активність (дія), яка деяким способом змінює стан оточення. Це може бути процес, що закінчується або не закінчується; виконуватися людиною або іншим агентом, або бути природним (фізичним) процесом.

Концепт *Process* має два підкласи, з яких найважливішим є *CoordinableActivity*. Координована активність – це процес, який може управлятися таким способом, щоб координуватися з іншою координованою активністю. Наприклад, виконуючи виклик, Web сервіс повинен бути координованою активністю, у значенні, що виклик такого сервісу може управлятися так, щоб координуватися з іншими викликами. Припустимо, що ми маємо двох агентів, кожен з яких хоче викликати той Web сервіс з різними параметрами, тоді агенти могли б управляти своїми викликами так, щоб не заважати один одному.

Не всі процеси системи є координованими, тому необхідно ввести концепт *NonCoordinableActivity*. Мається на увазі, що цей концепт охоплює всі ті процеси, координація яких неможлива в межах системи. До них відносяться:

– природні події (*Natural_Events*). Це фізичні процеси, які мають місце безвідносно того, що будь-який агент робить в системі. Наприклад, розпад атома, викликаний по суті випадковими квантовими подіями. Очевидно, такі процеси неможливо координувати з іншими процесами: вони матимуть місце (або не матимуть місця) безвідносно того, що агенти роблять в системі;

– зовнішні процеси (*External_Processes*). Це процеси (фізичні або природні), які знаходяться за межами контролю системи, тому вони не можуть управлятись агентами в системі. Зауважимо, що такі процеси можуть бути координовані сутностями поза межами системи: суть полягає в тому, що для цілей системи, до якої звертається база знань, вони не можуть бути координовані.

Інший спосіб пояснення відмінності між координованими (*coordinable)* і не координованими (*non-coordinable)* діями полягає в тому, що завжди є агент (тобто, програмний агент в межах системи), який пов'язаний з координованою активністю, тоді як не існує агента, який пов'язаний з не координованою активністю.

Координовану активність (*CoordinableActivity)* ми уявляємо як організоване дерево ієрархії дій з атомарних активностей (*AtomicActivitys)*, які є листами дерева. Тому *CoordinableActivity* є можливо композицією (*composedOf)* багатьох інших активностей (*Activitys)*, і може бути в таких станах:

– *ConjunctiveActivity.* В даному випадку це композиція цілого ряду інших дій, які повинні бути цілком успішно завершені для того, щоб повна активність завершилась.

– *DisjunctiveActivity*. Це композиція цілого ряду інших дій, з яких що найменше одна повинна успішно завершитись для того, щоб повна активність завершилась.

– *AtomicActivity*. В цьому випадку активність не компонує ніякої подальшої активності. (Множина *CoordinableActivitys,* яку ця активність компонує, є порожньою.)

Ми можемо ідентифікувати наступні підкласи стану атомарної активності (*AtomicActivity)*:

– *ConcludedCoordinableActivity:* активність, яка мала місце в минулому часі, і нині повністю завершена;

– *ContinuingCoordinableActivity:* активність, яка знаходиться нині в процесі;

– *ScheduledCoordinableActivity:* активність, що матиме місце, тобто вона планується для виконання якимось агентом;

– *SuspendedCoordinableActivity:* активність, статус якої невирішений.

Стисло розглянемо слоти і властивості концептів. *CoordinableActivity* матиме такі слоти:

– *Actor: Agent*, тобто, агент, який має намір виконати, або виконав цю активність;

– *earliest start date* (найраніша стартова дата): дата або *null*; дата зазначає найранішу дату, з якої може початися активність; *null* зазначає, що ця інформація не відома;

– *latest start date* (найпізніша стартова дата): дата або *null*; дата зазначає найпізнішу дату початку активності; *null* зазначає, що ця інформація не відома;

– *expected duration* (тривалість очікування): натуральне число, яке зазначає число мілісекунд, впродовж яких активність чекає, або *null*, що зазначає на невідому тривалість;

– *latest end date* (найпізніша дата кінця): дата або *null;* дата зазначає найпізнішу дату закінчення активності; *null* зазначає, що ця інформація не відома;

– *actual start date* (фактична стартова дата): дата або *null*; дата зазначає, коли активність фактично почалася; *null* зазначає, що ця інформація не відома;

– *actual end date* (фактична дата кінця): дата або *null*; дата зазначає, коли фактично може закінчитися активність; *null* зазначає, що ця інформація не відома;

– *final status* (заключний статус): переліковний тип (succeeded, failed, null).

Є ряд аксіом, які можуть вводитися у цей момент. Щодо *Conjunctive* і *DisjunctiveActivitys* ми маємо:

– *ConjunctiveActivity* успішно закінчена, якщо всі компоненти успішно закінчені;

– *DisjunctiveActvity* успішно закінчена, якщо мінімум один з компонентів успішно закінчений.

Щодо відношення між запланованими активностями і їх успішним завершенням ми маємо наступне:

– якщо активність планується, то вона повинна мати *null* фактичну стартову дату і фактичну дату кінця;

– якщо активність закінчується, то заключний статус повинен бути *non-null*;

– якщо активність почалася перед найранішою стартовою датою, то вона *failed*;

– якщо активність почалася після найпізнішої стартової дати, то вона *failed*.

2.3. Ресурси. Ідея концепту *Resource* полягає у тому, що ресурс є дещо, що може бути потрібно, щоб прискорити активність. Тому, ми маємо відношення «один до багатьох» між *AtomicActivitys* і *Resources*. Концепт *Resource* має такі слоти:

– *viable* (життєздатний): логічне значення, яке зазначає, чи знаходиться ресурс в стані, щоб бути використаним; значення *false* зазначає, що ресурс не міг би використовуватися ніякою активністю (навіть якщо ця активність потребує його). Інший простий спосіб пояснення *viable* – він зазначає, чи ресурс "зупинений", або "працюючий", або ні;

– *consumable* (споживаний): логічне значення, яке зазначає, чи використання ресурсу зменшить в деякій мірі подальшу придатність ресурсу; тобто, чи повторне використання ресурсу в активностях зробить ресурс нежиттєздатним;

– *shareable* (спільний): логічне значення, яке зазначає, чи ресурс може використовуватися більш ніж одним агентом в будь-який момент часу;

– *cloneable* (здатний до імітації): логічне значення, яке зазначає, чи ресурс є *cloneable (=true*), чи унікальний і *not-cloneable* (= *false*). Прикладом *cloneable* ресурсу повинен бути набір даних або цифровий документ. Прикладом унікального ресурсу повинен бути створений фізичний артефакт, як результат специфічного експерименту, або людського існування;

– *owner* (власник): *Agent*, який володіє ресурсом, або *null* (в цьому випадку ресурс може використовуватися будь-яким агентом безкоштовно). Якщо ресурсом володіє агент і інший агент бажає використати цей ресурс, то йому необхідно вступити в переговори про експлуатацію ресурсу.

2.4. Взаємозалежності між активностями. Розглянемо взаємовідношення, які існують між активностями. Перший концепт *Interdependency* має такі слоти:

– *source* і *target*: обидва слоти є активностями, які взаємозалежні;

– *isHard*: логічне значення, яке зазначає, яким є відношення: “soft” (= *false*), чи “hard” (= *true*), з наступною семантикою:

• *hard* відношення є таким, яке істотним чином впливатиме на успіх активностей або щось інше;

• *soft* відношення є таким, яке *може* впливати на активності, позитивно або негативно, але не впливатиме на їх успіх.

Підкласи *CoordinationRelation*:

– *NegativeCoordination*: взаємодія, якщо вона відбувається, то це приведе до зниження якості рішення або користі учасників;

– *PositiveCoordination*: взаємодія, якщо вона відбувається, то це приведе до зростання користі учасників або якості рішення.

Підклас *NegativeCoordination*: *FatalCoordination* є негативним взаємовідношенням координації, яке неминуче приведе до відмови однієї або більше складових активностей. Зазначимо, що екземпляри *FatalCoordination* завжди є негативними. Підкласами *FatalCoordination* є:

– *MutuallyExclude*: екземпляр цього відношення існує між двома *Atomic-Activitys* якщо:

1) вони обидва *вимагають* ресурсу *r;*

2) фактичне або плановане використання *r* обома активностями;

3) ресурс *r* є неспільний.

Ідея полягає у тому, що ці дві активності взаємно виключені і що вони не можуть обоє мати успіх, оскільки вони вимагають доступу до ресурсу, який неможливо розділити.

– *ResourceContention*: екземпляр цього відношення існує між двома *Atomic- Activitys* якщо:

вони обидва *вимагають* ресурсу *r;*

ресурс *r* споживаний.

Ідея полягає у тому, що одна з активностей могла б запобігти успішному завершенню інших активностей за допомогою відсутності ресурсу або приведення його до нежиттєздатності.

– *Disables*: одна активність блокує іншу, якщо її присутність буде остаточно запобігати присутності іншої.

Підкласами *PositiveCoordination* є:

– *ConditionallyFeeds*: в такій координації присутність активності A_1 зробить можливою присутність активності A_2, але можливо, що A_2 могла б відбутися (тобто, наявність A_1 є достатньою, але не необхідною подією для наявності A_2);

– *Enables*: наявність активності A_1 є як необхідною, так і достатньою умовою для присутності A_2;

– *IsSubsumedBy*: активність A_1 віднесена до категорії активності A_2, якщо A_2 містить всі активності A_1;

– *Subsumed*: протилежність *IsSubsumedBy*;

– *Favors*: активність A_1 підтримує активність A_2, якщо попередня присутність згодом поліпшіть повну якість A_2. Це є позитивне (s*oft*) відношення.

2.5. Приклад координації дій агентів в системі купівлі-продажу. Розглянемо онтологію з трьох агентів: Агент-покупець, Агент-постачальник, Агент-банк (рис. 2).

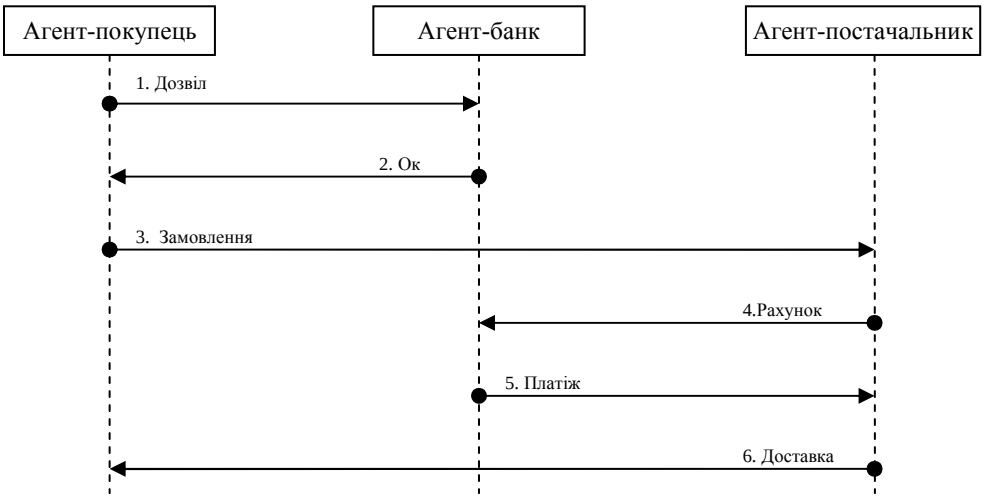

Рис. 2. Діаграма координації дій агентів

На рис. 2 зображена впорядкована послідовність дій агентів при виконанні купівлі товару.

Розглянемо сценарій координації дій агентів. Для досягнення мети – купівлі товару, агент-покупець повинен отримати позитивну відповідь на дозвіл від агента-банку, який обслуговує покупця. Активності агента-покупця залежать від активностей агента-банку. Тільки після виконання активності банку покупець має змогу відправити замовлення на товар. Оплата рахунку банком сприяє позитивному виконанню активності по доставці товару покупцеві.

Висновки

Ефективність семантичної мережі залежить від доступних технологій, які дозволяють різним компонентам – онтологіям, механізмам логічного виведення, агентам – працювати разом. Координація – це процес управління можливими взаємодіями між діями і процесами. Ефективна координація вимагає сумісного використання знань про активності (дії), ресурси і їх властивості. Сумісне використання може бути досягнуте статично, кодуванням на етапі розробки механізму координації серед агентів. Проте, у більш відкритих системах процеси і ресурси, які входять в систему, невідомі на етапі розробки. Альтернативою в цьому випадку є динамічний підхід, в якому вимоги до координації формуються під час виконання, а не на етапі розробки. Такий підхід дозволяє релевантним процесам обмінюватися своїми намірами щодо майбутніх дій і використання ресурсу, і щоб "обдумати" координацію під час виконання, з метою запобігання негативних взаємодій і сприяння позитивних взаємодій. Комунікація при динамічній координації вимагає погодженого загального словника з точною семантикою.

1. *Ben-Ari, M.*: Principles of Concurrent and Distributed Programming. Prentice Hall 1990. – p. 350.
2. *Wooldridge, M.*: An Introduction to Multiagent Systems. John Wiley & Sons 2002. – p. 366.
3. *Андон П.І., Дерецький В.О.* Засоби координації агентів у пошукових архітектурах Web//Проблемы программирования.– 2004. – № 1.– С. 60–70.
4. *Malone, T.W., Crowston, K.*: The interdisciplinary study of coordination. ACM Computing surveys 26 1994. – P. 87–119.
5. *von Martial, F.*: Coordinating Plans of Autonomous Agents (LNAI Volume 610). Springer-Verlag: Berlin, Germany: 1992. – P. 40.
6. *von Martial, F.*: Interactions among autonomous planning agents. In Demazeau, Y., Muller, J.P., eds.: Decentralized AI — Proceedings of the First European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW-89), Elsevier Science Publishers B.V.: Amsterdam, The Netherlands: 1990. – P. 105–120.
7. *Durfee, E.H.*: Coordination of Distributed Problem Solvers. Kluwer Academic Publishers: Dordrecht, The Netherlands 1988
8. *Decker, K., Lesser, V.*: Designing a family of coordination algorithms. In: Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95), San Francisco, CA – 1995. – P. 73–80.
9. OWL-S: OWL Semantic Web Services (2004): http://www.daml.org/services
10. WSMO: Web Service Modelling Ontology (2004): http://www.wsmo.org
11. http://www.w3.org/TR/2002/NOTE-wsci-20020808