**ИНФОРМАТИКА**

**2005** **июль-сентябрь** **№ 3**

*УДК 681.3.016*

**Ю.И. Воротницкий, Н.А. Зенькевич**

# КООРДИНАЦИЯ В ПРОГРАММНЫХ МНОГОАГЕНТНЫХ СИСТЕМАХ

*Предлагается подход к формализации механизмов координации и кооперации в распределенных системах, на основе которого разработаны новые алгоритмы координации для программных многоагентных систем.*

## Введение

В настоящее время технология многоагентных программных систем находит широкое применение для решения задач мониторинга и управления распределенными информационными системами [1–3]. При этом актуальной остается задача обеспечения координации программных агентов [4, 5]. В данной статье предлагается подход к формализации механизмов координации и кооперации в многоагентных системах (МАС), на основе которого разработаны новые алгоритмы координации для программных МАС.

## 1. Постановка задачи

Координация в распределенных системах с локально автономными частями является важной и сложной задачей, так как нескоординированные действия могут приводить к блокировкам и непоправимым ошибкам в процессе функционирования таких систем. Однако несмотря на важность проблемы, не существует даже общепризнанного определения координации, которое зачастую смешивается с понятием кооперации (см., например, [2–7]).

Обобщая приведенные в известной литературе определения, под координацией в МАС будем понимать процесс составления и изменения планируемых агентами действий с целью избежания конфликтных ситуаций и, при необходимости, обеспечения синхронизации этих действий.

С понятием координации тесно связано понятие кооперации, причем взаимосвязь этих понятий трактуется различным образом [2, 3]. В настоящей работе кооперация в МАС рассматривается как процесс составления и изменения планируемых действий с целью объединения усилий агентов для достижения общей цели. Таким образом, наличие кооперации между агентами наряду с координацией действий предполагает наличие совместной цели.

К основным причинам, обусловливающим необходимость координации действий агентов в МАС, можно отнести:

– ограниченность ресурсов информационной системы и совместное использование разделяемых ресурсов, что может приводить к ошибкам и блокировкам;

– необходимость выполнения распределенных синхронизируемых действий (одновременных или строго последовательных);

– необходимость повышения эффективности решений (уменьшения времени получения результата, дублирования одинаковых действий для надежности и контроля и т. п.).

Для создания формальной модели процессов координации и кооперации введем:

$A = \{A_1, A_2, \ldots, A_n\}$ – множество рациональных агентов МАС, полагая, что число агентов в системе конечно и равно $n$;

$R = \{R_1, R_2, \ldots, R_m\}$ – множество ресурсов, которые могут быть использованы агентами МАС при выполнении действий, их число также конечно и равно $m$;

$U = \{U_1, U_2, \ldots, U_l\}$ – множество $l$ параметров среды, на основе которых агенты принимают решения о выполнении или невыполнении запланированных действий или же о выборе одного из возможных действий, причем агенты обмениваются информацией об этих параметрах.

Зададим единичное возможное или планируемое действие одного агента следующим образом: $d = \{t, \Delta t, r, u, o, A_k\}$, где $t=[t_{min}, t_{max}]$ – интервал времени, в течение которого действие может быть начато; $\Delta t$ – интервал времени, в течение которого происходит действие; $r\_R$ –

подмножество ресурсов, используемых в данном действии; $u\_U$ – подмножество параметров, от которых зависит данное действие (его выполнение или отмена); $o$ – цель или ожидаемый результат действия (новые значения для некоторого подмножества параметров системы $U$); $A_k \chi A$ – агент, который планирует выполнение данного действия. Будем рассматривать систему, в которой результат всех действий строго детерминирован.

Планом действий интеллектуального агента $A_k$ с ожидаемыми результатами $O_k = o_1 4 o_2 4 \ldots 4 o_p$ является последовательность действий $D_k = \{d_1, d_2, \ldots, d_p\}$. Если для интервалов времени выполнения двух различных действий $i$ и $j$ $t_i \cap t_j \neq \varnothing$, $i, j \chi [1, \ldots, p]$, то условия $u_i$ и $u_j$ не должны выполняться одновременно. Под планом системы в целом будем понимать $D=\{D_j\}$, где $j=1,\ldots,n$.

Отметим, что данные определения подходят не только для интеллектуальных агентов, поведение полностью реактивных агентов может быть описано таким же образом. Для них план представляется в виде списка всех возможных действий при t=[0,∞) с обязательным разделением по условиям $u$, которые следует рассматривать как набор параметров на рецепторах агента.

В соответствии с приведенным выше определением наличие координации предполагает реализацию процедур обнаружения и исключения конфликтных ситуаций, отслеживания изменений значений параметров $U$ и обмен информацией об этих изменениях. Таким образом, должны быть построены скоординированные (бесконфликтные) локальные планы на какой-либо ближайший промежуток времени или на весь период работы системы.

Определим координационный конфликт как наличие в планах системы нескольких действий, выполнение которых приведет к взаимной блокировке или ошибке при одновременном использовании ресурсов или если выполнение одного из них отменится результатом предыдущих действий.

Приведем список возможных конфликтных ситуаций:

$d_i, d_j \mid t_i \cap t_j \neq \varnothing, r_i \cap r_j \neq \varnothing, i \neq j$, – конкуренция, когда подмножество требуемых обоими агентами ресурсов $r_i \cap r_j$ не может быть использовано одновременно;

$A_i$: $\{d_{i1}, d_{i2}\}$, $A_j$:$\{d_{j1}, d_{j2}\}$ и $d_{i1}$:$\{r_1\}$, $d_{i2}$:$\{r_1, r_2\}$, $d_{j1}$:$\{r_2\}$, $d_{j2}$:$\{r_2, r_1\}$ – пример блокировки, когда агенты успевают сделать первое действие, но не могут совершить второе (два агента, непрерывно выполняющие свои действия и занимающие соответствующие ресурсы, начали выполнение своих планов, и теперь первый ожидает, пока освободится ресурс, занятый другим агентом, который, в свою очередь, ждет освобождения ресурса, занятого первым агентом);

$d_i, d_j \mid o_i=o_j, i \neq j$, – повторение, агенты выполняют действие с одним и тем же результатом;

$d_i, d_j \mid t_{i\,min} < t_{j\,min}$, причем $o_i$ изменяет $u_j$ так, что $D_j$ не должно быть выполнено, – отмененное действие.

Множество конфликтных ситуаций может быть расширено за счет конфликтов, возникающих согласно бизнес-логике приложения, однако очевидно, что их можно свести к одной из вышеперечисленных путем введения дополнительных ресурсов и параметров.

Под скоординированным планом будем понимать план, в котором устранены конфликтные ситуации. Когда невозможно построить скоординированный план на весь промежуток работы МАС (например, параметры информационной системы $U$ изменяются не только агентами МАС), то необходимо строить план на максимально возможный ближайший промежуток времени или просто пытаться скоординировать ближайшие действия агентов.

Согласно предложенной модели описания поведения агентов процедуру координации можно разбить на три этапа:

– обнаружение конфликтов;

– построение и выбор решения;

– применение решения.

### 2. Алгоритмы координации

Проводя классификацию алгоритмов координации, прежде всего следует выделить два больших класса: централизованные и децентрализованные. Первые подразумевают явное по-

строение в одном из узлов МАС плана действий всех агентов $D$ или сбор в одном узле информации об изменении параметров $U$ с целью обнаружения конфликтов, их разрешения и распространения скоординированных локальных планов. Понятно, что с ростом числа агентов $n$, ресурсов $m$ и параметров $l$ время и ресурсы системы, затрачиваемые на передачу и обработку всех данных системы, будут увеличиваться [7]. К тому же использование централизованной координации может снизить преимущества применения МАС для решения конкретной задачи [2, 3].

При применении децентрализованного подхода данные обо всей системе не собираются в одном узле и агенты принимают решения только на основе локальной информации. При разработке децентрализованных алгоритмов координации МАС следует учитывать, что в распределенной системе не может быть единого времени [8]. Таким образом, действия могут быть скоординированы по времени только с определенной точностью синхронизации часов в узлах системы, определяемой скоростью обмена информацией.

Известны различные типы алгоритмов координации: организационные структуры, контрактные сети, переговоры, арбитрация, локальные самомодификации [2, 4–7]. Отметим, что многие из них могут быть использованы одновременно или же на разных стадиях координации: для построения скоординированных планов, обнаружения и устранения конфликтов.

Локальные самомодификации являются одним из наиболее простых и перспективных механизмов разрешения конфликтных ситуаций [7]. Самомодификация подразумевает изменение собственных планов с целью исключения конфликтов на основе получаемой из среды информации о параметрах системы. При этом потоки данных должны быть организованы таким образом, чтобы предоставлять всю необходимую для самомодификации информацию. Отметим, что реализация самомодификации не должна приводить к новым конфликтам и бесконечной цепочке самомодификаций.

В рамках построенной модели простейший алгоритм, реализующий метод локальных модификаций, можно описать следующим образом: каждому ресурсу $r_i$ ставится в соответствие параметр $u_i$, который может принимать значения «занят» (если ресурс в данный момент используется) и «свободен» (если ресурс в данный момент не используется). Агент, использующий ресурс $r_i$, должен иметь возможность определить значение параметра $u_i$. Перед выполнением следующего действия $d$ с использованием ресурсов $r_d$ агент получает все параметры $u_d$ и начинает выполнение действия только в случае, если все указанные ресурсы свободны. Однако данный алгоритм поведения не полностью устраняет конфликты типа конкуренции (по причине несинхронности действий) и блокировки.

Чтобы устранить конкуренцию за ресурсы, следует модифицировать алгоритм поведения агента следующим образом: перед выполнением каждого действия необходимо построить список агентов, которые будут использовать требуемые ресурсы в своем следующем действии. Далее в зависимости от заданной иерархии агентов определяется агент, который будет использовать конфликтующий ресурс (если агент находится выше по иерархии, то он имеет приоритет). Такая иерархия может быть как глобальной, неизменной на протяжении всего существования системы, так и локальной, определяющей приоритет только для данного разрешения конфликта. В программных средах проще всего в качестве глобальной иерархии использовать лексическое упорядочивание уникальных идентификаторов агентов.

### 3. Реализация алгоритмов координации в программных средах

В программных средах, как правило, сложно реализовать метод локальных самомодификаций в чистом виде, так как он предполагает нахождение агентом значений всех параметров $u_i$ для виртуальных ресурсов, от которых зависит действие.

Базируясь на рассмотренной модели описания поведения и известном методе локальных самомодификаций, предложим методику организации координированного поведения в многоагентной системе на основе обмена сообщениями. Разработанные алгоритмы полностью сохраняют все преимущества метода локальных самомодификаций, а также позволяют устранить взаимные блокировки.

Для решения конкретной задачи координации с помощью предлагаемых методов необходимо осуществить следующие действия:

– определить множество агентов и их глобальную иерархию;

– определить множество ресурсов, которые будут использовать агенты в своих действиях, а также зависимости агентов от ресурсов (построить множества агентов, использующих каждый конкретный ресурс);

– расширить список параметров за счет тех, которые описывают состояние ресурсов;

– внести в алгоритмы работы агентов обмен информацией о параметрах;

– изменить логику агента таким образом, чтобы действие выполнялось только в случае, если все задействованные ресурсы свободны (два последних шага выполняются согласно конкретному алгоритму).

В целом разработанные алгоритмы координации можно описать следующим образом: агенты должны обмениваться информацией о своих действиях со всеми «заинтересованными» агентами. Изменяя значение параметра $U_i$ (заметив, что он изменился), агент должен отправить сообщение об этом всем агентам, у которых есть зависимые от данного параметра действия. Собираясь провести действия с использованием ресурса $R_j$, агент также должен уведомить агентов, которые применяют данный ресурс в своих действиях. Таким образом, в онтологии системы должна присутствовать информация о каждом из агентов $A_i$ – множество параметров $\{U\}_{Ai}$, от которых зависят все возможные действия $A_i$. Координационные сообщения можно представить следующим образом: {«отправитель», «получатель», «параметр», «значение параметра»}.

Программные агенты представляют собой объект (или несколько объектов) с собственным потоком управления. Объект реализует два типа поведения: проактивное – действия, выполняющиеся с некоторой периодичностью, называемой сердцебиением (реализуется в виде специального метода, который периодически вызывается каркасом системы), и реактивное – обработка сообщений от других агентов и воздействий среды [1, 2]. Будем считать, что в реактивном поведении агенты не начинают действий над ресурсами и не изменяют параметров, а только изменяют свое внутреннее состояние. Действие над ресурсами выполняется в отдельном независимом потоке. Реактивное поведение реализуется в виде метода, в который передается событие, вызвавшее данную реакцию. Чтобы избежать нескоординированных действий самого программного агента, используется семафор, устраняющий возможность одновременного выполнения проактивного и реактивного действий.

Для описания алгоритма введем следующие обозначения и переменные, которые хранятся в памяти агента:

*«Агент»* – текущий агент, который выполняет данный алгоритм. Все остальные агенты выполняют тот же алгоритм, но имеют другие локальные переменные и выполняются в независимых потоках. Агентов можно сравнивать между собой согласно глобальной иерархии.

*«Сообщение»* (при реактивном поведении) – сообщение, которое возбудило данную реакцию. Обладает свойствами *«ресурс»* и *«тип»*, а также *«отправитель»* и *«получатель»*. При описании реактивного поведения используются прямые ссылки на данные свойства: *«ресурс»*, *«тип»*, *«отправитель»*. Типы сообщений: *«use»* (отправитель в настоящий момент использует указанный ресурс); *«occupy»* (отправитель собирается использовать ресурс в следующем действии); *«lock»* (отправитель хочет заблокировать ресурс); *«unlock»* (отправитель разблокирует ресурс после использования).

*«Отправить сообщение»* – отправить сообщение отправителю обрабатываемого сообщения.

*«Разослать сообщение»* – отправить сообщение всем агентам, использующим в своих действиях ресурс, указанный в сообщении.

*«NextStep»* – внутренняя булева переменная, определяющая возможность выполнения следующего действия.

$\{b\}_r$ – переменные агента, хранящие множество агентов, заблокировавших ресурс $r$.

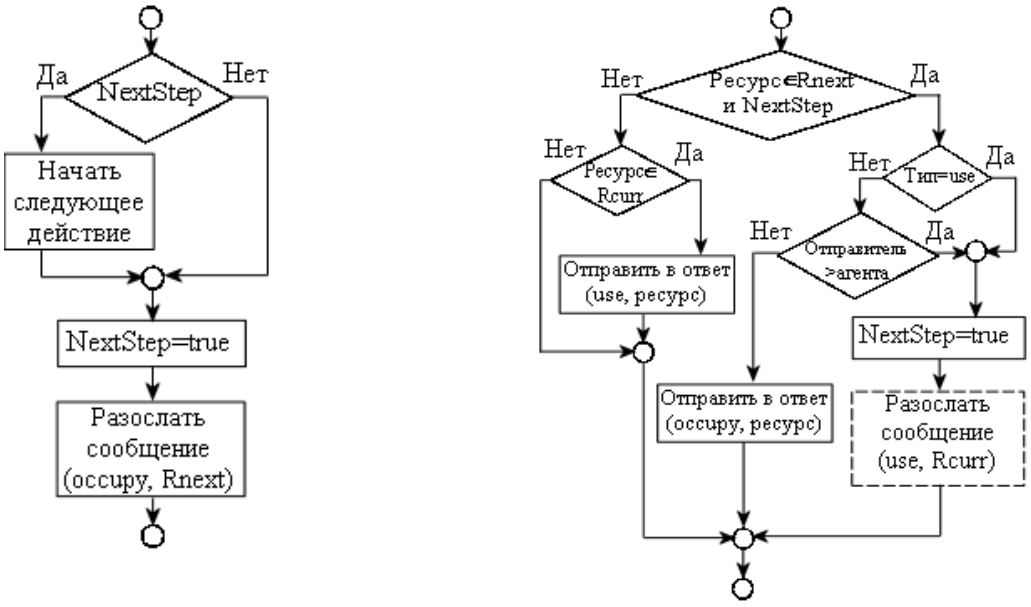
$R_{curr}$ – множество ресурсов, использующихся в текущем действии.

$R_{next}$ – множество ресурсов, использующихся в следующем действии.

$R_{prev}$ – множество ресурсов, использующихся в предыдущем действии.

В первом предлагаемом алгоритме (рис. 1) агенты не используют длительную память и не хранят состояния ресурсов, а обладают лишь одной внутренней переменной «будет ли выполнено следующее действие». Параметры, описывающие состояние ресурса, принимают два

значения: «будет использован» и «используется». Агент выполняет действие только в том случае, если значение внутренней NextStep переменной равно *true*.




*а)* *б)*

Рис. 1. Граф-схемы алгоритма поведения агента (без памяти):
*а)* проактивное поведение; *б)* реактивное

При реализации алгоритмов предполагалось, что агент не может находиться в состоянии, не использующем никаких ресурсов; в противном случае необходимо опустить действия, отмеченные пунктирной линией. Также считалось, что каждый цикл проактивного действия побуждает агента начинать следующее действие.

Второй алгоритм (рис. 2) требует, чтобы агенты хранили в памяти состояние каждого ресурса, которое они получают из сообщений. Агент выполняет действие только в случае, если все используемые ресурсы не заблокированы и не собираются быть заблокированными вышестоящими в иерархии агентами. Алгоритм позволяет сократить число координационных сообщений в системе по сравнению с первым алгоритмом.

Чтобы устранить возможность взаимной блокировки, второй алгоритм следует модернизировать следующим образом: перед каждым действием блокировать не только ресурсы, используемые на следующем шаге, а еще ряд ресурсов, вычисляемых по формуле $R_{nexttolock} = \{r\}_0 4 \{r\}_1 4 \ldots 4 \{r\}_q$, где $\{r\}_{q+1}=\varnothing$, $\{r\}_0=R_{next}$, а $r \chi \{r\}_i$ тогда и только тогда, когда $r \chi R_{next+i}$ и существуют $A_k$ (не текущий) и некоторый ресурс $r'$, такие, что $r' \chi \{r\}_{i-1}$ и $A_k$ применяет $r$ в действии, которое предшествует действию, использующему $r'$.

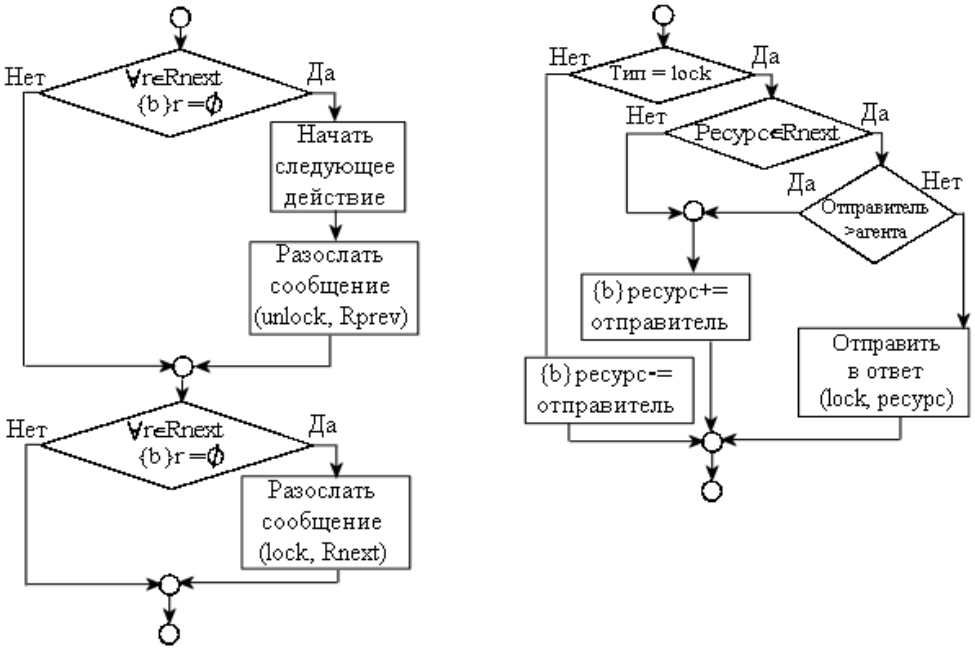
Для упрощения записи логики агента были использованы дополнительная функция $N(r)$, возвращающая для текущего агента множество агентов $\{A\}$, из-за которых данный ресурс был включен в $R_{nexttolock}$, и на её основе булева функция *NN*, которая возвращает *true*, только в том случае, если для любого агента, принадлежащего $R_{nexttolock}$, пересечение $\{b\}_r \cap N(r) \neq \varnothing$.

Заблокировав все ресурсы $R_{nexttolock}$, агент никогда не окажется в ситуации взаимной блокировки, так как другие агенты будут ждать (в случае необходимости) освобождения ресурсов. Полный модернизированный алгоритм также включает разблокирование ресурсов при неудачной попытке блокировки (рис. 3).

Для проверки работоспособности приведенных алгоритмов была решена задача передвижения независимых агентов по прямоугольной доске, разделенной на ячейки. Каждый агент обладает собственным маршрутом и имеет информацию о маршрутах всех остальных агентов. Конфликтом считается ситуация, когда два агента одновременно располагаются в одной ячейке.

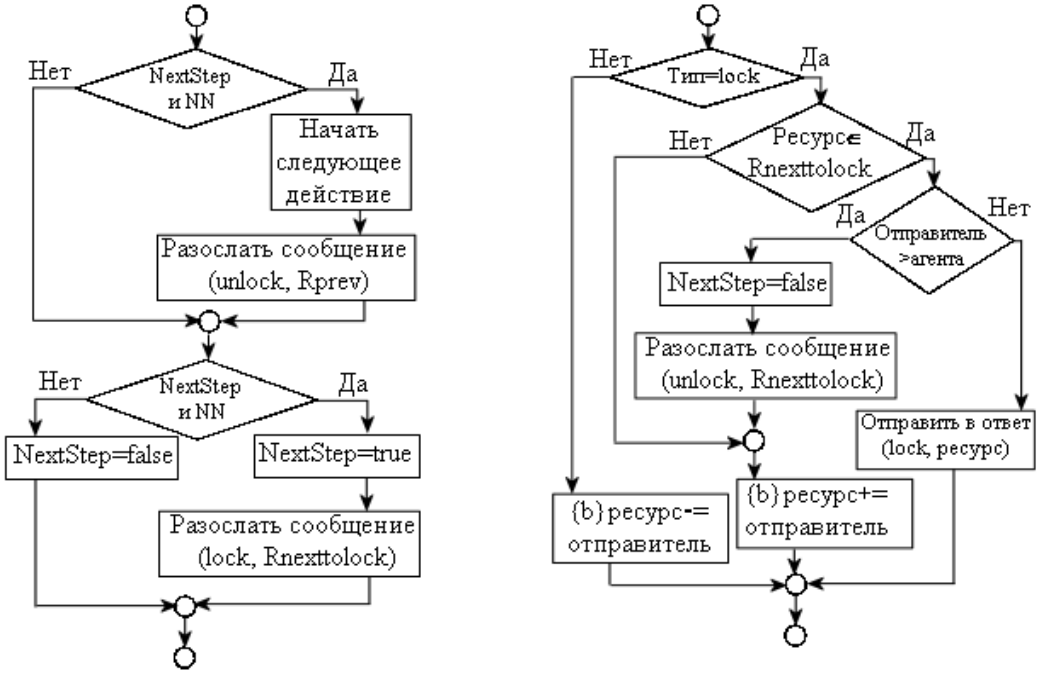
Было разработано приложение, реализующее данную логику, а именно отображение доски и агентов на ней, обмен сообщениями между агентами, обнаружение конфликтов, интерфейсы аген-

тов и маршрутов. Каждый агент выполняется в отдельном, не синхронизированном с другими потоке; период «сердцебиения» для проактивного поведения выбирается для каждого агента случайным образом. Были реализованы разработанные алгоритмы поведения агентов, а также алгоритм локальных самомодификаций без обмена сообщениями (получение информации из положения на доске), которые тестировались на маршрутах как с возможностью взаимных блокировок, так и без.




*а)* *б)*

Рис. 2. Граф-схемы алгоритма поведения агента (с памятью): *а)* проактивное поведение, *б)* реактивное




*а)* *б)*

Рис. 3. Граф-схема модернизированного алгоритма: *а)* проактивное поведение, *б)* реактивное

Как и ожидалось, все алгоритмы устраняют конфликтные ситуации типа конкуренции за ресурсы. Модернизированный алгоритм также успешно устраняет конфликты типа блокировки.

**Заключение**

Метод локальных самомодификаций был применен к программным многоагентным системам, в результате чего были разработаны три алгоритма координации, которые показали свою работоспособность, сохранив все преимущества базового метода.

Разработанные подход и алгоритмы успешно использовались для решения задачи мониторинга информационной системы с помощью МАС [9].

**Список литературы**

1. Wooldrige M. Agent-Based Computing // Interoperable Communication Network. – № 1(1). – 1998. – P. 71–95.

2. Тарасов В.Б. От многоагентных систем к интеллектуальным организациям: философия, психология, информатика. – М.: Эдиториал УРСС, 2002. – 352 с.

3. Multiagent system: a modern approach to distributed artificial intelligence / G. Weiss, M. Wooldridge et al. – London: The MIT Press, 1999. – 618 p.

4. Nwana H., Lee L., Jennings N.R. Co-ordination in software agent systems // BT Technology Journal. – V. 14. – № 4. – 1996. – P. 79–89.

5. Durfee E.H. Scaling up agent coordination strategies // IEEE Computer. – № 34(7). – 2001. – P. 39–46.

6. Ricci A., Omicini A., Denti E. Activity theory as a framework for MAS coordination // Proc. of the 3rd Int. workshop «Engineering Societies in the Agents World». – Madrid, 2002. – P. 96–110.

7. Barber S.K., Han D.C., Tse-Hsin L. Coordinating distributed decision making using reusable interaction specifications // Proc. of the 3rd Pacific Rim Int. workshop on multi-agents. – Melbourne, 2000. – P. 1–15.

8. Ван Стеен М., Таненбаум Э. Распределенные системы. Принципы и парадигмы. – СПб.: Питер, 2003. – 880 с.

9. Воротницкий Ю.И., Зенькевич Н.А. Архитектура многоагентных систем восстановления работоспособности информационных систем // Комплексная защита информации: сб. мат. VIII Междунар. конф. (23–26 марта 2004 г., Валдай, Россия). – Минск, 2004. – С. 136–138.

**Поступила 12.05.05**

*Белорусский государственный университет,*
*Минск, пр. Независимости, 4*
*e-mail: voront@bsu.by*

**Y.I. Varatnitsky, M.A. Zenkevich**

## COORDINATION IN SOFTWARE MULTIAGENT SYSTEMS

New formalization for coordination and cooperation in distributed systems and new coordination algorithms for software multiagent systems are presented.