В.И. Городецкий

Теория, модели, инфраструктуры и языки спецификации командного поведения автономных агентов. Обзор (Часть 2)[1]

Аннотация. Современные исследования в области автономных агентов и многоагентных систем в настоящее время во многом стимулируются потребностями робототехники, в которой одной из центральных задач является задача координации коллективного поведения в команде автономных роботов при выполнении ими некоторой миссии. Главная особенность командной работы автономных агентов – это отсутствие внешнего управления поведением команды при выполнении миссии. Этот аспект поведения агентов является источником сложных проблем. В работе приводится краткий обзор и анализ состояния исследований в рассматриваемой области. В первой части были проанализированы известные результаты в теории командной работы. Во второй части дается обзор известных моделей и предметно–независимых программных инфраструктур, поддерживающих создание приложений, и приводится сравнительный анализ существующих языков спецификации индивидуальных и командных аспектов поведения агентов.

Ключевые слова: автономный агент, командная работа, теория командной работы, модель командной работы. инструментальное средство, язык спецификации.

3. Модели и инструментальные средства разработки

Обе теории, теория общих намерений и теория общих планов (см. Часть 1), послужили основой для разработки ряда конкретных *моделей* командной работы автономных агентов. При этом авторы таких моделей использовали комбинацию идей обеих теорий, хотя и несколько отличными способами. Далее рассматриваются две наиболее известные и популярные модели, Teamcore и RETSINA. Они наиболее глубоко проработаны, поддерживаются инструментальными средствами, предназначенными для поддержки разработки предметно–независимых компонент программного продукта, реализующего приложения в области командной работы. Существуют и другие модели командной работы и варианты их реализаций. Они описаны в работах [21] (модель COLLAGEN), [13] (модель GRATE), [14] (модель ADEPT), [4] (модель COOL). Однако они разработаны в меньшей мере и, в основном, не имеют инструментальной поддержки, а в литературе нет сведений об опыте их практического использования.

3.1. Модель STEM и программная инфраструктура Teamcore

Модель командной работы, предложенная в [26], описывает общие рамки рассуждений и взаимодействия агентов команды на основе комбинирования теории общих намерений и теории общих планов. На основе теории общих намерений в этой модели формируются базовые блоки плана командной работы. Модель общих намерений и обязательств дает возможность явного представления плана совместной работы агентов команды, который задает отношения на множестве действий агентов и их групп, а также индивидуальный и групповой

[1] Данная работа является инициативной и выполнена при частичной поддержке гранта № 10–07–00026 Российского фонда фундаментальных исследований.

вклад агентов в его выполнение. Этот план используется как руководство для рассуждений о координации и коммуникации, а также позволяет на основе мониторинга командной работы обнаруживать и своевременно предотвращать ошибки исполнения общего плана.

Для того чтобы обеспечить согласованную работу агентов команды на основе рассуждений агентов о координации и коммуникации в процессе выполнения плана, а также чтобы реализовать мониторинг, модель STEM использует идеи теории общих планов. При этом решаются четыре задачи, которые описываются далее.

Первая из них состоит в декомпозиции общих намерений команды на общие намерения подгрупп агентов и индивидуальные намерения агентов, а также их представление в виде иерархии. Эта декомпозиция выполняется без указания конкретных исполнителей действий, реализующих намерения. В результате строится иерархия намерений (от общего намерения команды до намерений отдельных агентов) и действий общего плана, а также индивидуальных действий агентов и их групп, ведущих к достижению общей цели. На этом этапе конкретные исполнители действий не указываются. Общий план доводится до агентов, что формирует у них взаимные убеждения о плане выполнения командной работы. Таким образом, на этом этапе модель общих намерений представляется в виде множества базовых блоков, которые структурируются иерархически на основе теории общих планов. Компоненты этой иерархии отражают ментальные состояния индивидуальных агентов команды и отношения на множестве действий, следование которым гарантирует достижение командой (общей) цели.

Вторая задача, решаемая в модели STEM, имеет целью обеспечить агентов команды информацией, минимально необходимой им в процессе исполнения общего плана. В соответствии с теорией общих планов, каждый агент должен знать, какое именно намерение сейчас реализуется тем или иным агентом или группой агентов, но не как именно это намерение реализуется. Это минимально необходимая информация, которая определяет, каким образом именно агенту следует выполнять мониторинг текущих действий команды.

Третья задача решается всякий раз тогда, когда на некотором шаге общее намерение оказывается недостижимым, и STEM в этом случае формирует общее намерение перепланировать работу команды. Перепланирование состоит в том, что сначала команда пытается анализировать причину недостижимости. Например, причина может состоять в том, что текущая подзадача не назначена агенту (группе), или она не может быть выполнена назначенным агентом или группой. В обоих случаях каждый агент команды действует так, чтобы найти агента (группу), который в состоянии выполнить задачу. Агент может также предлагать для этих целей себя или другого агента. Могут быть и другие причины (они обычно выявляются экспериментально при компьютерном моделировании). Естественно, что для реализации таких функций убеждения агентов должны содержать, по крайней мере, информацию о способностях различных членов команды и о том, что именно они делают в текущий момент.

Четвертая задача состоит в следующем. Например, некоторый агент может получить частным образом информацию о том, что намерение β_i достигнуто. В теории общих намерений он будет пытаться сделать эту информацию доступной всем агентам. В теории общих планов любой агент знает всегда следующий оператор β_j для исполнения, и команда не может приступать к выполнению β_j, пока все члены команды, ответственные за выполнение β_j, не будут осведомлены о достижении β_i. Для этого необходимо с помощью рассуждений на основе отношений на множестве действий вывести необходимую схему коммуникаций. В STEM используется явное декларирование отношений зависимостей информации для пары действий, требующей дополнительной коммуникации. Таким образом, сообщая об окончании действия β_i, STEM определяет для каждого выведенного или декларированного отношения следующее действие β_j, и при этом информация, имеющая отношение к выполнению действия β_j, также передается адресатам, чем достигается взаимное убеждение в том, что β_i достигнуто. Таким образом, коммуникации оказываются различными в зависимости от того, имеются или нет зависимости между информацией, имеющей отношение к действию β_i и к действию β_j.

Позднее автором модели STEM была предложена архитектура и инструментальная поддержка для разработки предметно–независимых компонент приложений в области командной работы

автономных агентов. Эта архитектура известна под именем Teamcore [27, 20]. Она интегрирует в себе базовую модель командной работы STEM. Инструментарий Teamcore позволяет быстро объединять прикладных агентов, возможно, гетерогенных, в единую команду.

Архитектура агента команды, сформированной с помощью инструментария Teamcore, разделяется на две части. Одна из них является предметно–зависимой, а другая – предметно–независимой. Предметно–независимая часть агента называется teamcore–агент. Фактически она играет роль "обертки" (прокси) для предметно–зависимой части агента, наделяя агента в целом способностью к командной работе. Именно она построена на основе модели STEM.

Архитектура Teamcore наделяет агентов способностью к переговорам. Предметно–независимая часть Teamcore включает в себя также компоненту, которая наделяет агента способностью к обучению командной работе. Эта способность основана на использовании понятия роли. При этом основная функция обучающей компоненты teamcore–агента состоит в том, чтобы управлять распределением и перераспределением ролей, меняя тем самым организационную структуру команды в процессе исполнения командного плана. Под ролью понимается функциональность, которую агент реализует в интересах выполнения общего плана, причем функциональность роли относится к предметно–зависимой компоненте агента.

Для решения задачи обучения авторы разработали модель организационной структуры команды. В ней каждый агент поддерживает часть общей модели организационной структуры команды, которая в целом представляется ациклическим графом. В этом графе роли соответствуют узлам (они также помечаются именем агента, которому эта роль назначена), а дуги отвечают отношениям между ролями.

Стиль разработки приложений в области командной работы, реализованный в инструментарии Teamcore, авторы назвали "программирование, ориентированное на команду" (team–oriented programming) [19]. Детальное описание архитектуры и инструмента можно найти в [28].

3.2. Архитектура RETSINA

Модель RETSINA и соответствующая ей архитектура [24, 25] построена, главным образом, на основе теории общих планов, которая реализуется в форме *кооперативного интерфейсного агента*, инкапсулирующего способность рассуждать о намерениях. Механизм рассуждений этого агента используется в нем для того, чтобы:

1. в нужные моменты времени определять релевантного получателя критической информации и передавать ему эту информацию;
2. отслеживать зависимости между различными задачами, исполняемыми членами команды;
3. своевременно распознавать конфликты и нарушения ограничений;
4. предлагать решения для разрешения конфликтов;
5. выполнять мониторинг процесса выполнения плана и работы членов команды.

На практике все эти действия должны выполняться в распределенном варианте, и командная работа должна быть организован таким образом (в соответствии с теорией общих намерений), чтобы всегда находился какой-то член команды, ответственный за то, чтобы определить необходимость выполнения соответствующего действия и выполнить его.

В модели RETSINA полагается, что любой агент, кроме решения своих запланированных задач, должен быть в состоянии получать дополнительные задачи и подцели от других агентов, передавать другим агентам информацию о ходе выполнения своих задач, выполнять мониторинг действий команды в целом и уметь делегировать свои или чужие задачи другим членам команды.

Чтобы успешно решать последнюю задачу, агент должен знать, какие задачи он способен решать сам, какую из целей обслуживает та или иная задача и какой агент может эту задачу решить. Кооперативный интерфейсный агент реализует эти функции с помощью коммуникационного модуля. Он также имеет декларативное представление целей агентов и механизм планирования действий по их достижению. Кроме того, механизм планирования позволяет агентам рассуждать о действиях, которые он должен предпринять сам, и какие действия он должен делегировать другим агентам, а также – каким именно.

Для обеспечения эффективной координации распределенного поведения членов команды в динамической среде, архитектура RETSINA построена с использованием следующих принципов:

1. Полагается, что среда является открытой и формирование команды может происходить динамически путем добавления или удаления членов команды.

2. Члены команды – это гетерогенные агенты с различными способностями, причем множества способностей агентов могут иметь непустое пересечение.

3. Члены команды имеют общую модель командной работы, которая не зависит от предметной области.

4. Перепланирование, как индивидуальное, так и командное, необходимо в интересах общей цели и выполнения обязательств.

5. Каждый агент может инициировать командную работу по достижению некоторой поставленной им цели.

6. В задачах выполнения некоторой миссии при формировании команды могут потребоваться агенты с разными способностями. Начальное положение членов команды и возможности их использования при выполнении командной задачи могут быть неизвестны. Более того, команда может быть реконфигурирована ввиду потери некоторых членов команды или потери ими некоторых своих способностей (например, когда у робота отказал некоторый сенсор). Реконфигурация может состоять также в добавлении нового члена команды.

Все эти особенности, а также необходимость добавления в команду нового агента, замены его или удаления могут потребовать изменения текущей модели командной работы путем

1. поиска агентом других агентов с требуемыми способностями (agent discovery);

2. быстрого ввода новых агентов в "курс дела", т.е. быстрого информирования их о миссии и текущем плане командной работы, который может быть уже частично выполненным;

3. быстрого переназначения ролей для выполнения оставшейся части плана при появлении в команде новых агентов (заново введенных или пришедших на замену отказавшим).

Все эти три задачи называются координацией способностей членов команды агентов.

После того, как команда агентов сформирована, все члены команды должны выполнять мониторинг деятельности команды для того, чтобы поддерживать синхронную работу и порядок выполнения ими своих подзадач, соблюдение и выполнение своих обязательств, а также выполнение работы по достижению командной цели в соответствии с планом. Такой план должен учитывать временные ограничения и сроки выполнения тех или иных действий, а также поддерживать выполнение ограничений по ресурсам.

3.3. Обсуждение

Описанные модели и инструментальные средства являются наиболее известными и наиболее проработанными. В принципе, судя по имеющейся литературе, они активно используются при разработке приложений. Однако, поскольку они, в основном, используются для разработок в интересах военного ведомства США, то подробная информация об этих приложениях в доступной литературе отсутствует. Однако информация о принципах, заложенных в модели, а также доступная информация об архитектуре инструментальных средств и их возможностях, позволяет сделать вывод об их работоспособности для создания приложений. Представляется, что эта информация позволяет понять, каким образом аналог каждого из инструментариев может быть создан.

4. Языки спецификации командной работы

Модели и инструментальные средства, описанные в предыдущем разделе, позволяют создавать предметно–независимую часть программного обеспечения, которое реализует командный стиль работы группы автономных агентов. Авторы полагают, что такие средства позволяют реализовать стиль разработки программного обеспечения, который они называют *программированием, ориентированным на команду* [19]. Однако представляется, что моделей и инструментальных средств типа STEM+Teamcore или RETSINA еще недостаточно для того, чтобы полностью поддержать такой стиль программирования. Инструментарии типа Teamcore, RETSINA оставляют все же значительный объем работ для ручного программирования всего, что относится к конкретному приложению. Обычно агент является только управляющей компонентой робота, физического или программного (виртуального), а потому всегда требуется разработка предметно–зависимого программного обеспечения агента, и его интерфейсов к исполнительным органам робота. Все это требует спе-

цифического программирования, которое должно поддерживаться выразительными языками программирования.

Применительно к отдельным автономным роботам первые разработки подобных языков относятся еще к началу 1990-х годов. В последующем они были положены в основу разработок языков спецификации роботов, объединяемых в команду. Рассмотрим некоторые из наиболее известных языков такого назначения.

4.1. Язык Colbert

Язык описан в [15]. Основные операторы языка позволяют:

– выстраивать последовательности действий, из которой робот должен выбирать очередное действие для исполнения;

– специфицировать мониторинг исполнения базовых действий и других операций (в частности, для определения условия окончания действия, например, по достижению роботом заданных координат; для детектирования ошибок и непредвиденных ситуаций; для проверки условий типа таймаутов и др.);

– запускать программы, которые соответствуют действиям;

– проверять значения внутренних переменных и задавать их значения.

Язык Colbert достаточно прагматичен. Он построен на основе парадигмы машины перехода состояний, которая задает формально операционную семантику, и обладает возможностями описания и реализации параллельных процессов. Для ускорения работы программы и обеспечения ее переносимости на большинство ОС авторы использовали в качестве базового языка программирования стандартный язык ANSI–C.

Напомним, что, машина перехода состояний (конечный автомат) представляется множеством состояний ***S,*** множеством входов ***I,*** множеством выходов ***O,*** функцией перехода состояний автомата $f(S,I) \rightarrow S$ и функцией выхода $g(S) \rightarrow O$. Автомат представляется графом, в котором узлам поставлены в соответствие состояния, в которых *выполняются* некоторые *действия (*иначе – узлам графа соответствуют *действия*), а дуги – это (мгновенные) переходы между ними. Они метятся условиями, которые должны удовлетворяться для перехода в новое состояние (для перехода к выполнению нового действия). При выполнении этих условий генерируется событие, запускающее новое действие. Состояния метятся именами (функциями) тех действий, которые в узле (в состоянии) выполняются. В традиционном конечном автомате вся информация о состоянии кодируется в самом узле. В языке Colbert узлы представляют только состояние исполнения действия. Манипулирование другой информацией выполняется иными средствами.

В схеме функционирования (в графе конечного автомата) явно не задается информация о том, когда следует начинать очередное примитивное действие. Эта информация задается внешними по отношению к автомату средствами в виде условий перехода к следующему состоянию (следующему действию). Иначе говоря, после выдачи команды на очередное действие оно выполняется, а машина состояний ожидает (в текущем состоянии), когда будет выдана команда на переход к следующему состоянию, которая выдается по окончании выполнения действия в текущем состоянии. В языке Colbert эти условия перехода выполняются "по умолчанию", однако они могут дополняться параметрами *noblock* и *timeout*. Первая команда указывает, что нужно переходить к выполнению действия, соответствующего очередному состоянию, а вторая явно указывает время ожидания.

Достоинство языка Colbert состоит в его способности выполнить интуитивное преобразование из конструкций операторов языка С в конструкции машины состояний, которая управляет роботом. Заметим, что машина перехода состояний может оказаться достаточно громоздкой для прямого программирования, а язык Colbert помогает эти трудности преодолеть.

Рассмотрим основные элементы, с помощью которых записываются операторы языка Colbert (Табл.1).

1. Утверждения. Утверждения языка могут быть четырех категорий:

а. Управляющие действия (*Control actions*);

б. Действия по тестированию состояния исполнения сценария (*Activity state tests*) в интересах мониторинга;

в. Действия внутреннего состояния (*Internal state actions*);

г. Упорядочивающие действия (*Sequencing actions*).

Варианты таких утверждений, примеры и минимальные пояснения к ним даны в Табл. 1.

2. Под–сценарии. Язык Colbert поддерживает иерархию вызова сценариев, т.е. некоторый под–сценарий может быть вызван из исполняемого сценария, если под–сценарий является дочерним узлом исполняемого сценария. Тем самым язык реализует иерархическую декомпозицию задачи.

3. Параллельное исполнение и синхронизация. Задача управления роботом обычно декомпозируется на множество подзадач, которые могут требовать координации. Семантика Colbert в состоянии поддерживать множество *одновременно выполняемых действий* (сценариев), которые взаимодействуют либо косвенно (через базу данных), либо напрямую путем посылки сигналов (сообщений – при агентской технологии программной реализации). В первом случае сценарий может запросить информацию о состоянии исполнения некоторого действия, что реализуется просто благодаря семантике машины состояний. Во втором случае это может быть сигнал (событие) о начале и/или об окончании некоторого действия, которое может быть успешным или нет, когда оно завершается по таймауту. Сценарий может также временно приостановить исполнение некоторого действия, и тогда оно не выполняется до тех пор, пока не поступит внешний сигнал на его возобновление. Заметим, что прерывание и возобновление действия (сценария) – это обычные управляющие сигналы для запущенных процессов. Синхронизация действий реализуется *Программой исполнения* (*Colbert executive*), которая рассматривается далее.

4. Программа исполнения (*Colbert executive*). Эта программа реализует операционную семантику языка, т.е. семантику действий, предусмотренных сценарием, на основе семантики машины переходов состояний. Сначала она запускает команду `start`, которая помещает экземпляр схемы поведения робота в структурированный список действий (сценариев). Имея иерархический список упорядоченных действий (потенциальный список нитей исполнения в операционной системе), программа выполняет их поочередно или параллельно, двигаясь по списку, и реализует операционную семантику соответствующих действий списка, которая интерпретируется с помощью семантики иерархии конечных автоматов. Базовый цикл *Программы исполнения* состоит в том, что в нем выполняется анализ каждого действия

Табл. 1. Утверждения языка Colbert
(таблица заимствована из [15] и переработана)

	Примеры	Описания
Управляющие действия		
Примитивное действие	`Move 500 timeout 20`	Запустить примитивное действие
Запуск сценария поведения робота	`Start patrol noblock`	Начать выполнение сценария
<Сигнал> действие	`Suspend patrol`	Сигнал, посылаемый в сценарий
Тестирование состояния исполнения сценария		
<состояние> (сценарий)	`stGetTaskState ("patrol")`	Тестировать исполнение сценария
Действия внутреннего состояния		
Присваивание значений, функции	`X=ObjInFront() +20`	Изменить значение атрибута в БД
Упорядочивающие действия		
goto	`goto start;`	Переход к состоянию
while if	`if (a= =0) goto start;`	Итеративный условный переход
waitfor	`waitfor(timeout (act)|| a<0);`	Остановка по условию
wait(int)	`wait 30`	Ожидание в течение *n* циклов

и сценария из списка, проверяется, можно ли изменить его состояние, вызывается соответствующее действие и обновляется состояние. За счет короткого цикла (100 мс) оказывается возможным учесть изменения всех условий в режиме реального времени.

Очевидно, что описанный язык может быть использован только для спецификации поведения одиночного робота, т.к. в нем нет необходимых средств поддержки командной работы. Например, в этом языке невозможно представить аспекты командной работ, которые предусматриваются моделями типа Teamcore или RETSINA. Его достоинство состоит только в простоте описания поведения, представления операционной семантики[2], а также в переносимости на различные операционные платформы благодаря использованию стандартного языка программирования ANSI–C, в который транслируется код, написанный на языке Colbert. Тем не менее, он может быть расширен для

[2] Простота представления и реализации операционной семантики является свойством всех языков, которые в основе своей используют модель машины перехода состояний (конечного автомата).

спецификации прикладных автономных агентов команды, управляющих роботами. Язык, описываемый в следующем подразделе, может рассматриваться как одно из таких расширений, хотя он создавался с самого начала именно как язык спецификации прикладных агентов, управляющих роботами команды.

4.2. Язык XABSL

Язык XABSL (Extensible Agent Behavior Specification Language), предложенный в [16] и развитый в [21], разработан для описания коллективного поведения роботов в условиях, когда внешняя среда частично наблюдаема и обладает непредсказуемой динамикой. Его формальную основу, как и в языке Colbert, составляет модель иерархии машин состояний. Он включает [21]:

– Модулярную архитектуру, представленную множеством параллельных машин состояний, и язык спецификации для ее описания;

– Компилятор, генерирующий документацию и промежуточный код, который затем запускается на исполнение;

– C++ библиотеку, используемую для исполнения программы.

Каждая машина состояний описывает некоторую простую последовательность действий, каждое из которых авторами называется *опцией*. Опция исполняется тогда, когда состояния соответствующей ей машины состояний активны. Некоторые состояния могут соответствовать вложенным опциям, т.е. машине состояний, которая запускается при активации этих состояний. Иначе говоря, машины состояний в языке XABSL могут быть организованы иерархически. Опции могут иметь атрибуты, значения которых задаются вызывающей опцией. Заметим, что одновременно могут вызываться несколько опций, что реализует идею параллельного исполнения действий различными агентами. При запуске иерархически организованных опций они структурируются в виде дерева, которое называется деревом активации опций, в корне которого находится *корневая опция*. Все опции описываются ациклическим направленным графом (рекурсивные операции в этом языке запрещены), который называется *графом опций*. Заметим также, что иерархическая организация машин состояний позволяет использовать одну и ту же опцию в различных контекстах, т.к. любую опцию можно вызвать из разных опций, но, однако, одна и та же опция не может вызываться из разных опций одновременно.

Операционная семантика языка XABSL задается дискретным циклом исполнения, причем на каждом таком цикле обновляется состояние каждой опции дерева опций, включая корневую опцию, а каждая машина состояний дерева опций стартует со своего выделенного состояния, называемого начальным состоянием. В опции выделяется также состояние, которое называется конечным ее состоянием. Дискретный цикл исполнения задается либо постоянным по длительности, либо инициируется некоторым событием. Опция, запустившая некоторую опцию, запрашивает также, не достигла ли последняя своего конечного состояния, и если это произошло, вызывает следующую опцию. Важно отметить, что машины состояний, описываемые в языке XABSL, ответственны только за выбор действия агента, но модели самих действий являются компонентами общего программного обеспечения. Поэтому необходимо еще иметь интерфейс от языка к общему программному обеспечению. Этот интерфейс в языке организован как вызов процедур, соответствующих действиям. Иначе говоря, язык описывает иерархическую ссылочную структуру, а конкретные программы, вызываемые по ссылкам, разрабатываются вручную. Эти программы пишутся на специальном символическом языке описания, по которому далее автоматически генерируется XML код.

Язык XABSL имеет специальные средства для описания коллективного поведения агентов, которые могут обмениваться сообщениями. Однако эти средства помогают решать лишь частные задачи координации коллективного поведения [21]. К таким средствам относится, например, специальный атрибут машины состояний, который указывает, какое количество агентов может одновременно исполнять соответствующую опцию. Например, если речь идет о проходе через дверь, то такой атрибут указывает, максимальное число роботов, которые могут одновременно выполнять это действие, чтобы не вызвать пробку. Еще одно средство предназначено для указания на то, что действия некоторого множества агентов должны быть синхронизированы, т.е. они в результате действия должны придти в некоторое со-

стояние опции одновременно. Например, некоторые агенты перед выполнением такого перехода должны ожидать, когда к нему будут готовы все агенты. Число агентов, чьи действия должны быть синхронизированы, указывается в качестве атрибута.

Другие, более сложные акты координации в этом языке не могут быть специфицированы машинами состояний напрямую. Они могут реализовываться через использование коммуникаций. Например [21], взаимодействие может быть представлено в каждом роботе в виде иерархической машины состояний, в которой на верхнем уровне имеется машина состояний, и сложное действие выполняется с использованием обмена некоторыми символами. Кроме того, обмен сообщениями используется для того, чтобы скоординировать поведение агентов, если они намерены использовать одно и то же действие с заданным атрибутом объема (количества участвующих в нем роботов).

На практике нельзя предположить, что сообщение о намерении робота и ответ на него реализуются мгновенно. Обычно в этом процессе имеется некоторая задержка. Поэтому в период между этими сообщениями может возникнуть конфликт по максимальному объему опции, если два агента пытаются использовать ее приблизительно в одно и то же время. Для предотвращения такого конфликта язык XABSL имеет средство, которое требует, чтобы любой агент, который пытается использовать опцию с заданным атрибутом объема, предварительно рассылал сообщение всем агентам об этом намерении и делал некоторую задержку в принятии собственного решения и его исполнении. За время этой задержки агент получает ответы от других агентов, оценивает возможность превышения объема опции и принимает решение в соответствии с процедурой вычисления приоритетов, задаваемой пользователем. Время задержки в принятии решения есть средство обеспечения некоторой гарантии отсутствия конфликта. Но это снижает общее время реакции системы на ситуацию, и потому здесь нужно искать некоторый баланс, который зависит от приложения и конкретной опции. В предметной области футбола роботов Robocup [23] примерами являются условия типа "*только один робот может завладеть мячом*", "*только два робота могут участвовать в действии "пас от одного робота другому*" в сценарии "игра в треугольнике" и другие. Однако реализация более сложных вариантов коллективной работы не может быть выполнена, если опираться на модель конечной машины состояний.

Основное отличие реализации языка XABSL от реализации языка Colbert состоит в том, каким образом все средства языка интегрируются, и как программа исполняется с помощью соответствующей программы (*XabslEngine run-time system*–в случае языка XABSL и *Colbert executive* – в языке Colbert). В XABSL это делается проще, поскольку он не накладывает ограничений на архитектуру агента и не управляет работой сенсоров или исполнительных устройств [21]. Это дает возможность его использования не только для спецификации реактивного поведения агента, но и для представления поведения, использующего сложную модель внешнего мира. Авторы также полагают, что для сложных случаев можно использовать машину состояний, которая предназначена специально для распределения задач на множестве агентов (роботов), в особенности, когда агенты гетерогенны и обладают различными способностями. Однако примеров подобного использования средств языка авторы не приводят. Неизвестны также случаи использования языка XABSL для спецификации коллективного поведения с количеством взаимодействующих автономных агентов, например, более двух.

Следует заметить, что язык XABSL разрабатывался для описания командной работы с хорошо определенными правилами типа командной работы роботов–футболистов [23]. И это определяет и ограничивает целесообразную прикладную область для его использования.

4.3. Язык PNP

Язык PNP (Petri Net Plan) был предложен в работе [31] и далее развит в [17] и [31]. В его основе лежит широко известная модель сети Петри. Известно, что эта модель обладает достаточно выразительной графической формой представления сложных процессов. Для нее разработаны средства формальной верификации различных свойств, что привлекает к ней (не всегда заслуженно) внимание разработчиков моделей поведения, в особенности, в задачах управления потоками взаимодействующих процессов (работ). Например, существует несколько расширений языка сетей Петри для спецификации бизнес–процессов [29]. Примерами являются язык WN

(Workflow Net) [2], и язык YAWL (Yet Another Workflow Language) [1].

Авторы данного языка позиционируют его как достаточно выразительное средство для спецификации планов поведения агентов команды в частично наблюдаемой динамической среде. Он рассматривается ими как графический язык для описания динамических систем, способный выразить ряд специфических особенностей коллективного поведения. Язык PNP имеет возможности по представлению циклов, параллельности, прерываний, возможности описания реакции системы на ошибки. Он имеет также специальные средства координации работы агентов, к которым относятся операторы жесткой (*hard–synchronization*) и мягкой (*soft–synchronization*) синхронизации, которые позволяют средствами языка представить некоторые аспекты командного плана и взаимодействия агентов. Они описаны в [31]. Модульность языка обеспечивается использованием в нем понятия суб–плана, который ссылается не на конкретное действие, а на некоторую под–сеть Петри. Понятие суб–плана реализует то же самое, что и иерархия машин состояний в языке XABSL. Далее рассматриваются средства языка PNP и анализируются его реальные возможности.

Язык PNP описывает сложное поведение агентов с помощью трех элементарных конструкций, а именно, *no action* (отсутствие действия), *ordinary action* (простое действие) и *sensing action* (зависимое действие) [30], которые расширяют язык сетей Петри. Их графическое представление в нотации, принятой для сетей Петри, дано на Рис. 1.

Отсутствие действий представляется одним узлом–местом, которое полагается начальным и конечным одновременно.

Простое действие, имеющее ненулевую длительность во времени, представляется тремя узлами–местами p_i, p_e и p_o, которые отвечают стартовому узлу действия, действию в состоянии исполнения и конечному узлу соответственно, и двумя узлами–переходами t_s и t_e, которые отвечают переходу, запускающему исполнение действия, и переходу, отвечающему окончанию действия соответственно.

Зависимое действие, имеющее ненулевую длительность, представляется четырьмя узлами–местами p_i, p_e, p_{o_t} и p_{o_f}, которые отвечают стартовому узлу действия, действию в состоянии исполнения и двум конечным узлам–действиям. Из них конкретно исполняемое действие определяется контекстом исполнения, и тремя переходами t_s, t_{e_t} и t_{e_f}, которые отвечают переходу, запускающему исполнение действия, и переходам, отвечающим окончанию действия в том или ином контексте, соответственно.

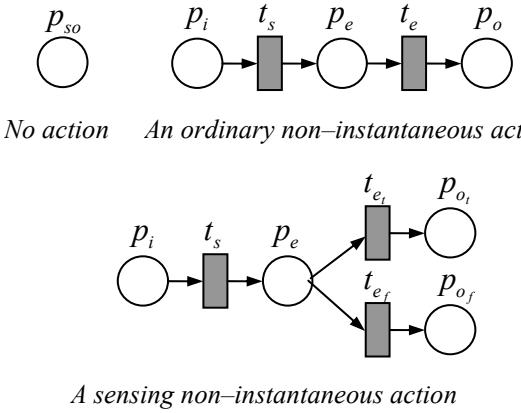

Рис. 1. Элементарные конструкции языка PNP (рисунок заимствован из [30])

В формальной нотации, представляющей сеть Петри в виде тройки *<P, T, F>*, где *P*–множество мест, *T*–множество переходов, *F*–множество бинарных отношений[3], эти элементарные конструкции представляются таким образом [30]:

No action: $<\{p_{so}\}, \{\varnothing\}, \{\varnothing\}>$;

An ordinary non–instantaneous action:
$<\{p_i, p_e, p_o\}, \{t_s, t_e\}, \{(p_i, t_s), (t_s, p_e), (p_e, t_e), (t_e, p_o)\}>$;

A sensing non–instantaneous action:
$<\{p_i, p_e, p_{o_t}, p_{o_f}\}, \{t_s, t_{e_t}, t_{e_f}\}, \{(p_i, t_s), (t_s, p_e), (p_e, t_{e_t}), (p_e, t_{e_f}), (t_{e_t}, p_{o_t}), (t_{e_f}, p_{o_f})\}>$.

В языке PNP допускается также использование мгновенных простых и зависимых действий описанного типа, которые не содержат промежуточного места p_e и переходов t_e, t_{e_t} и t_{e_f}.

[3] В сети Петри задаются еще две компоненты, а именно, функция весов *W*, которая определяет возможность запуска переходов, и начальная разметка мест M_0, которая определяет начальное состояние сети и дальнейшие переходы в ней. О них речь пойдет позже.

Введенные элементарные конструкции языка используются в языке PNP для построения операторов, в частности, операторов последовательность, условный оператор, цикл, параллельное исполнение и прерывание. Первый оператор строится как объединение двух последовательно исполняемых простых действий с ненулевой продолжительностью, при этом конечное место первого действия объединяется со стартовым местом последующего. Второй оператор строится на основании зависимого элементарного оператора и двух операторов простых действий, соединяемых с двумя выходными ветвями зависимого оператора путем слияния конечных узлов последнего с соответствующими начальными узлами простых действий. На Рис. 2 приведен условный оператор с неопределенным числом итераций, заданный с помощью элементарных операторов. Представление других операторов (условного с заданным числом операций, оператора, задающего параллельное исполнение, оператора прерывания и др.) может быть найдено в работе [31].

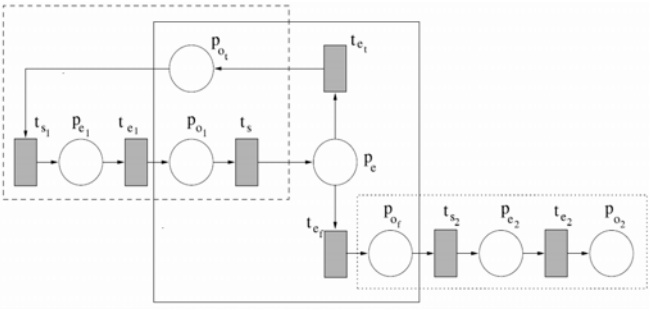

Рис. 2. Оператор итерации с неопределенным числом повторений, выполняемых до тех пор, пока воспринимаемое свойство истинно (рисунок заимствован из [31])

Как уже отмечалось, язык PNP использует иерархию в описании плана. Это обеспечивается тем, что действие может ссылаться не только на элементарное действие, заданное процедурой, но также и на некоторый план, описанный в терминах языка PNP. Для этого используется оператор прерывания (описанный в терминах примитивов сети Петри) аналогично тому, как это делается в языках Colbert и XABSL.

Для того, чтобы реализовать исполнение плана в динамической среде, план должен содержать средства реакции на события внешнего мира. Для этого каждому переходу приписывается некоторое условие, аргументами которого являются такие события. Такие условия для перехода t обозначаются парой символов t,ϕ. Если такое условие для перехода отсутствует, то по умолчанию полагается, что оно всегда имеет значение *истина*. Но иногда в таком случае его удобнее полагать равным значению *ложь*.

Рассмотрим задание операционной семантики языка PNP. Как известно, текущее состояние исполнения процессов, описанных сетью Петри, задается текущим распределением маркеров в узлах–местах, а сам процесс исполнения плана задается переходами маркеров из одних узлов–мест в другие. Поэтому одной из компонент операционной семантики является разметка сети. Элементом операционной семантики является также функция целочисленных весов, которая ставится в соответствие каждой стрелке графа сети Петри (формально – каждому отношению (p, t) и (t, p) множества отношений F, задающих сеть).

В операционной семантике языка PNP авторы различают понятия *возможный* переход и *исполняемый* переход. Для двух разметок PNP M_i и M_{i+1} на шагах i и $i+1$ переход $M_i \to M_{i+1}$ является *возможным*, если и только если существует переход $t \in T$, такой, что

(1) для любого $p' \in P$, для которого пара $(p', t) \in F$, величина $M_i(p')$ в разметке на шаге i положительна, а в разметке на следующем шаге $M_{i+1}(p') = M_i(p') - 1$;

(2) в разметке M_{i+1} на шаге $i+1$ $M_{i+1}(p'') = 1$ для любого $p'' \in P$, такого, что пара $(t, p'') \in F$.

Для двух разметок PNP M_i и M_{i+1} в ситуации на момент времени τ на шагах i и $i+1$ переход $M_i \to M_{i+1}$ является *исполняемым* в этой ситуации, если и только если существует переход $t \in T$, такой, что переход $M_i \to M_{i+1}$ является *возможным* и событийное условие ϕ, заданное на переходе t (условие t,ϕ) истинно в этот момент времени τ. Проще говоря, переход выполняется только тогда, когда в соответствии с топологией графа плана, задаваемого сетью Петри, в соответствии с весовой функцией, заданной на переходах, а также в соответствии с текущей разметкой переход является возможным, но при этом выполняется еще

условие t,ϕ, определенное на множестве элементов потока событий (потока управления).

Таким образом, операционная семантика языка PNP определяется топологией графа сети Петри, соответствующего плану, разметкой узлов–мест и потоком событий, отражающих состояния процессов исполнения плана, т.е. она аналогична операционной семантике других языков, имеющих нотацию в терминах графов и управляемых событиями [31].

Как обычно, возможность достижения требуемого целевого состояния с помощью построенного плана определяется начальной разметкой сети Петри, задающей план, а также внешними событиями, отражающими состояние и динамику внешнего мира, и событиями, которые генерируются агентами системы в процессе функционирования. Очевидно, что начальная разметка может гарантировать достижение целевого состояния в контексте только некоторого множества событий, а потому план не всегда может приводить к цели, даже если формальная верификация плана (для сетей Петри это возможно) приводит к положительному заключению.

Программа исполнения плана на каждом шаге контролирует возможность того или иного перехода и выполняет то множество переходов, которое выполнимо. Псевдокод алгоритма, реализуемого программой исполнения, приведен в работе [31].

Однако компоненты языка PNP, описанные выше, рассчитаны на централизованное исполнение плана, при котором задачи между агентами распределяются внешней программой, в частности, человеком. Эта особенность плана характерна и для его расширения, предложенного в работе [31], хотя авторы претендуют на большее. Оно обеспечивает возможность описания простой кооперации агентов (роботов), используя идеи кооперации теории общих намерений, правда, в несколько упрощенной форме.

В этом расширении языка PNP предполагается, что план создается централизованно и заранее, и при этом каждому действию построенного плана ставится в соответствие уникальный идентификатор агента (робота), который будет его исполнять. План доводится до всех роботов, и они автоматически определяют те задачи, которые им назначены для индивидуального исполнения. Кооперация агентов, предусмотренная планом, при его исполнении сводится к

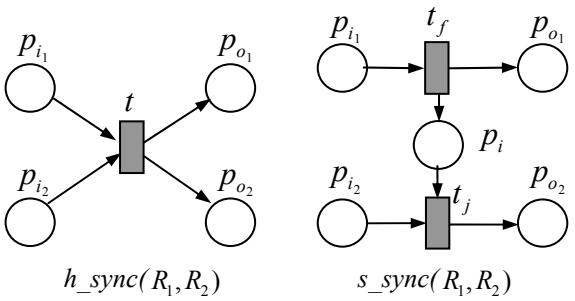

Рис.3. Операторы жесткой и мягкой синхронизации (рисунок заимствован из [32])

синхронизации действий кооперирующихся агентов. В предложенном расширении языка авторы ввели два оператора синхронизации, а именно, оператор *softsync* (оператор *мягкой синхронизации*) и оператор *hardsync* (оператор *жесткой синхронизации*). Структуры этих операторов в терминах примитивов сетей Петри представлены на Рис. 3.

Оператор *жесткой синхронизации* используется для того, чтобы пара роботов могла одновременно начинать некоторое совместное действие. Например, если два робота должны приблизиться к заданному объекту, например, к столу, а затем одновременно начать действие по его переносу, то такое общее действие должно начинаться строго одновременно. В плане, который разработан для пары роботов, эти действия синхронизируются с помощью оператора *hardsync*. На Рис. 4а дано графическое представление соответствующей компоненты общего плана для двух роботов. В нем указаны идентификаторы роботов, которые должны выполнять это действие, а также оператор жесткой синхронизации, аргументами которого являются имена этих роботов. На Рис. 4б тот же план представлен в виде индивидуальных планов, которые формируются роботами автоматически после получения общего плана. В них необходимость синхронизации указана с помощью действия *sync*(*id1,id2*). Однако если жесткая синхронизация выполняется в соответствии с представлением на Рис.3, то это потребует наличия в схеме одного общего агента, реализующего централизованное управление, что, однако, нежелательно, поскольку противоречит самой идее автономной работы команды агентов. Поэтому авторы рассматривают также вариант операции синхронизации с помощью обмена сообщениями. В таком вари-

анте оператор $h\_sync(R_1, R_2)$ декомпозируется на два коммуникационных примитива. Этот вариант демонстрируется на Рис. 5. на примере, в котором используется оператор мягкой синхронизации.

Оператор *мягкой синхронизации* задает предшествование на множестве действий пары роботов, но не требует их строго одновременного исполнения от момента старта до момента окончания действия, исполняемого совместно. На Рис. 5 этот оператор используется для того, чтобы указать, что действие робота R1, а именно, его действие *R1.act1* должно выполняться раньше действия робота R2, обозначенного идентификатором *R2.act4*. На этом рисунке переход t_{er1} соответствует окончанию действия *R1.act1*, а переход t_{sr2} соответствует началу действия *R2.act4.*

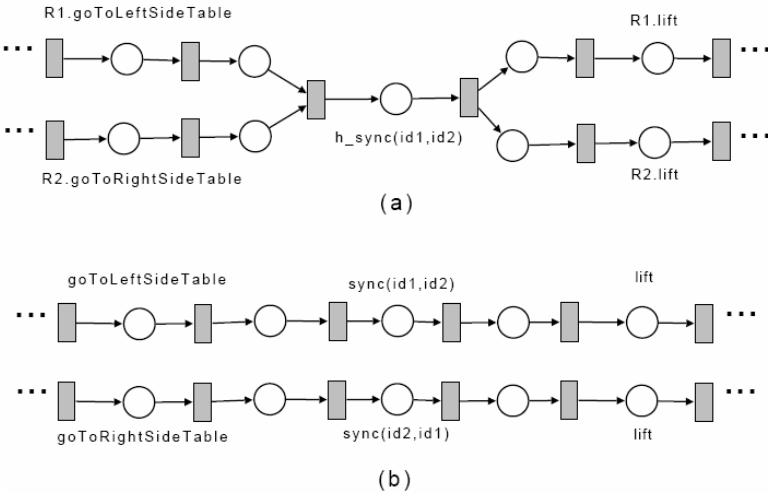

Рис. 4. Представление совместного действия двух роботов в общем плане и в индивидуальных планах роботов с использованием оператора жесткой синхронизации (рисунок заимствован из [17]).

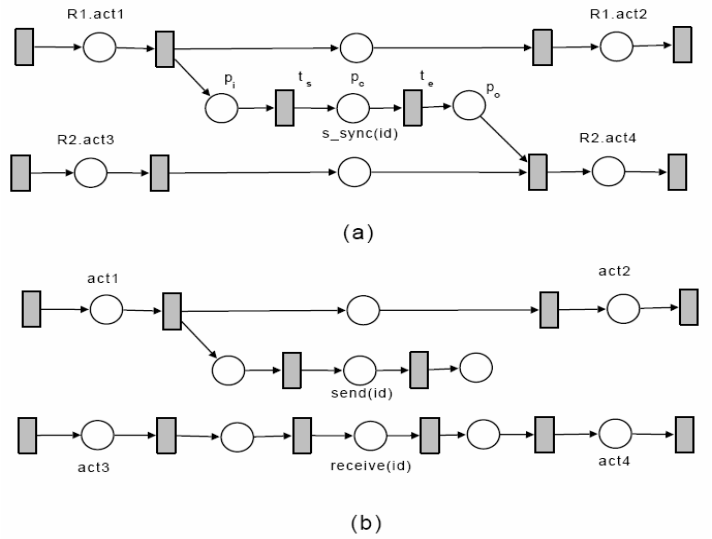

Рис. 5. Представление порядка выполнения действий двух роботов в общем плане и в индивидуальных планах роботов с использованием оператора мягкой синхронизации (рисунок заимствован из [32])

4.4. Сравнение языков и обсуждение их возможностей

Языки XABSL и PNP обладают практически равными возможностями. Использование в PNP формальной модели сети Петри не дает ему каких-либо преимуществ перед XABSL. Более того, операционная семантика в PNP реализуется сложнее, чем это свойственно модели машины перехода состояний. Почти все возможности, требуемые для описания индивидуальных действий роботов и их простых взаимодействий (иерархическое описание плана действий, синхронизация, прерывание, параллельное исполнение действий, представление зависимых действий и др.), имеются в обоих языках. Оба языка имеют одинаковые возможности по использованию протокола общих намерений для координации поведения в стиле командной работы, хотя эта возможность не заложена в средства этих языков. Использование же стандартных средств коммуникации на базе языка ACL [3] и распределенной программной инфраструктуры типа JADE для поддержки взаимодействия агентов [11] потребует одинаковых дополнительных усилий и ручного программирования интерфейсов. Имеется некоторое преимущество языка PNP перед языком XABSL и другими языками в том, что план, описанный сетью Петри, можно верифицировать формальными методами, однако это рассматривается авторами как теоретическая возможность, о которой они PNP упоминают, но практически не используют.

Оба языка имеют, хотя и различные, но достаточно ограниченные возможности по спецификации командной работы *автономных* агентов и не

в состоянии описать командную работу с той степенью полноты и гибкости, которая предусмотрена в моделях Teamcore и RETSINA.

Самый главный недостаток обоих языков состоит в том, что в них детальный план поведения агентов разрабатывается заранее. Он предполагает априорное описание всех ситуаций, в которых возможен неоднозначный выбор продолжения сценария в зависимости от текущего контекста. Оба языка не имеют средств распределения и перераспределения функций между агентами в динамике исполнения командной работы. В работах [6, 7] авторы языка PNP предлагают алгоритм динамического распределения задач между роботами, используя модель сети Петри. Однако в настоящее время нет сведений о том, что эти возможности каким-то образом нашли отражение в языке PNP.

Заключение

Агентские технологии получают все большее признание и распространение среди интеллектуальных информационных технологий. Одной из областей активного использования этих технологий является современная робототехника. Однако взгляды на важность агентских технологий для различных задач робототехники за рубежом и в России долгое время, к сожалению, были различными.

Первые применения агентской архитектуры в робототехнике за рубежом относятся еще к началу 1990-х годов. В России такой интерес только что наметился, и пока реально разработанные отечественные робототехнические системы, использующие многоагентную архитектуру, отсутствуют[4]. Однако современные мировые тенденции ушли гораздо дальше и в настоящее время речь идет о создании команд автономных виртуальных и физических роботов для выполнения достаточно сложных миссий. Предполагается, что команде автономных роботов только ставится цель миссии, а построение плана, его выполнение, а также динамическое перепланирование командной работы в зависимости от воздействия внешней среды, состояния членов команды и противодействия оппонирующей команды – это уже должно решаться командой автономно. И такой взгляд на возможности агентских технологий для обеспечения стиля командного поведения роботов не является фантазией или делом далекого будущего. Именно такое поведение уже демонстрируют команды роботов, играющих в футбол, на ежегодно проводимом уже в течение 14 лет чемпионате мира.

Именно этим, т.е. реально большим различием во взглядах зарубежной и российской науки на роль агентских технологий в командной работе роботов, мотивирована настоящая работа. В ней дается краткий обзор, своего рода введение в современную теорию, модели, инструментальные средства и языки спецификации командного поведения агентов, управляющих поведением роботов при совместном выполнении ими автономной миссии.

[4] С "высоких трибун" часто звучат значительно более оптимистические утверждения, но пусть они останутся на совести их авторов.

Литература

1. Aalst, W., Hofstede, A.: YAWL: Yet Another Workflow Language. In: Information Systems, vol. 30(4), pp. 245–275 (2005).
2. Aalst, W.: The Application of Petri Nets to Workflow Management. In: Journal of Circuits, System and Computers, vol. 8(1), pp. 21–66 (1998).
3. Agent Communication Language Specification. http://www.fipa.org/repository/aclspecs.html
4. Barbuceanu, M., Fox, M: The Architecture for an Agent Building Shell. In: Wooldridge, M., Muller, J., Tambe M. (eds.) Intelligent Agents, vol. 2, Lecture Notes in Artificial Intelligence, 1037, Springer-Verlag, Heidelberg (1996).
5. P.Cohen and H.J.Levesque. Teamwork. Nous, 35 (1991).
6. Farinelli, A., Iocchi, L., Nardi, D., Ziparo, V.A: Assignment of Dynamically Perceived Tasks by Token Passing in Multi-Robot Systems. In: Proceedings of the IEEE, Special Issue on Multi-Robot Systems (2006).
7. Farinelli, A., Iocchi, L., Nardi, Ziparo, V.A.: Task Assignment with dynamic perception and constrained tasks in a Multi-Robot System. In: Proceedings of International Conference on Robotics and Automation (ICRA'05) (2005).
8. Gorodetsky V., and Kotenko, I. Scenarios Knowledge Base: A Framework for Proactive Coordination of Coalition Operations. (Eds. M.Pechoucek and A.Tate) Proceedings of the Third International Conference on Knowledge Systems for Coalition Operation (KSCO-04), Florida, USA. October , 2004, pp. 83-88.
9. Grosz, B.: Collaborating Systems. AI Magazine, 17(2), 1996.
10. B.Grosz and S.Kraus. Collaborative Plans for Complex Group Actions. Artificial Intelligence, 86, 1996, 269-358 (1996).
11. Java Agent Development Framework. http://jade.tilab.com/.
12. Jennings, N.R,. Varga, L.Z,.Aarnts, R.P, Fuchs, J., and.Skarek, P.: Transforming standalone expert systems into a community of cooperating agents. Int. Journal of Engineering Applications of Artificial Intelligence, 6(4), 1993, pp. 317-331 (1993).

13. Jennings, N.: Controlling Cooperative Problem Solving in Industrial Multi-Agent Systems Using Joint Intentions. In: Artificial Intelligence, vol. 75, pp. 195--240 (1995).
14. Jennings, N., Norman, T., Faratin, P.: ADEPT: An Agent-Based Approach to Business Process Management. In: ACM SIGMOD Record, vol. 27(4), pp. 32—39 (1998).
15. Konolige, K.: COLBERT: A Language for Reactive Control in Saphira. In: KI '97 Proceedings of the 21st Annual German Conference on Artificial Intelligence: Advances in Artificial Intelligence, Lecture Notes in Computer Science, vol. 1303, pp. 31—52. Springer-Verlag, London, (1997).
16. Lőtzsch, M.: XABSL - A Behavior Engineering System for Autonomous Agents. Diploma thesis. Humboldt–Universitat zu Berlin (2004) Available online: http://www.martinloetzsch.de/papers/diploma-thesis.pdf.
17. Palamara, P.F., Ziparo, V.A., Iocchi, L., Nardi, D., Lima, P.: Teamwork Design Based on Petri Net Plans. RoboCup International Symposium (2008).
18. E.Oliveira, J.Fonseca, A.Steiger-Garcao.Resource selection and cost estimation on a MAS for civil construction companies. In Proceedings of the International Workshop "Distributed Artificial Intelligence and Multi-agent-Systems", St. Petersburg, Russia, pp. 162-173 (1997).
19. Pynadath, D., Tambe, M., Chauvat, N., Cavedon L.: Toward Team-Oriented Programming. In: Jennings, N., Lesperance, Y. (eds.) Proceeding ATAL '99 6th International Workshop on Intelligent Agents VI, Agent Theories, Architectures, and Languages, pp. 233–247. Springer, London, (2000).
20. Pynadath, Tambe, M.: An automated teamwork infrastructure for heterogeneous software agents and humans. In: Journal of Autonomous Agents and Multi-Agent Systems (JAAMAS): Special Issue on Infrastructure and Requirements for Building Research Grade Multi-Agent Systems (2002).
21. Rich, C., Sidner, C.: COLLAGEN: When Agents Collaborate with People. In: Proceedings of the International Conference on Autonomous Agents (Agents'97) (1997).
22. Risler , M., von Stryk, O.: Formal Behavior Specification of Multi-Robot Systems Using Hierarchical State Machines in XABSL. AAMAS08 Workshop on Formal Models and Methods for Multi-Robot Systems. Estoril (2008), see also http://www.xabsl.de.
23. RoboCup initiative, http://www.robocup.org/.
24. Sycara, K., Paolucci, M., Giampapa, J., van Velsen, M.: The RETSINA Multiagent Infrastructure. Autonomous Agents and Multi-agent Systems, vol. 7(1) (2003).
25. Sycara, K., Sukthankar, G.: Literature Review of Teamwork Models. CMU-RI-TR-06-50, November 2006, Robotics Institute Carnegie Mellon University. (2006) http://www.ri.cmu.edu/pub_files/pub4/sycara_katia_2006_1/sycara_katia_2006_1.pdf.
26. M.Tambe. Towards Flexible Teamwork. Journal of Artificial Intelligence Research, 7, 1997, pp. 83-124 (1997).
27. Tambe, M., Shen, W-M., Mataric,M., Pynadath, D. Goldberg, D., Modi, P.J, Qiu, Z., Salemi, B.: Proceedings of AAAI Spring Symposium on Agents in Cyberspace, pp 136- 141 (1999).
28. Teamcore research group. http://teamcore.usc.edu.
29. Weske, M.: Business Process Management: Concepts, Languages, Architectures. Springer (2007).
30. Ziparo, V.A., Iocchi, L.: Petri Net Plans. In: Fourth International Workshop on Modeling of Objects, Components, and Agents, pp. 267-289. Turku (2006).
31. Ziparo, V., Iocchi, L., Nardi, D., Palamara, P.F., Costelha. H.: Petri Net Plans: a Formal Model for Representation and Execution of Multi-Robot Plans. In: AAMAS, Estoril, (1), pp. 79–86. (2008).

Городецкий Владимир Иванович. Доктор технических наук, профессор, главный научный сотрудник лаборатории интеллектуальных систем Санкт-Петербургского института информатики и автоматизации РАН. Окончил Ленинградскую военно-воздушную инженерную академию им. А.Ф. Можайского в 1960 году и Ленинградский государственный университет в 1970 году. Автор более 350 публикаций. Область научных интересов: многоагентные системы, извлечение знаний из данных, самоорганизация и др. E-mail: gor@iias.spb.su , http://space.iias.spb.su/ai/gorodetsky.