**Введение**

*"Человек благоразумный подстраивает себя под окружающий мир, тогда как безрассудный человек упорно подстраивает этот мир под самого себя. Так что весь прогресс опирается на людей безрассудных".*

*Бернард Шоу.*

**Парадоксальное высказывание Бернарда Шоу имеет непосредственное отношение к тексту статьи. В самом деле, почему человек так стремится поработить себя машинами? На сколько велика их власть над людьми? Неужели весь прогресс человечества в том только и заключается, чтобы построить такой мир, в котором сам человек станет звеном избыточным, а потом неизбежно исчезнет?**

#### В этой статье я хотел бы затронуть некоторые из последних научных работ в области искусственной жизни и искусственного интеллекта.

Искусственный интеллект вообще и экспертные системы в частности прошли долгий и тернистый путь: первые увлечения (1960 год), лженаука (1960-65), успехи при решении головоломок и игр (1965-1975), разочарование при решении практических задач (1970-1985), первые успехи при решении ряда практических задач (1962-1992), массовое коммерческое использование при решении практических задач (1993-1995). Но основу коммерческого успеха по праву составляют экспертные системы и, в первую очередь, экспертные системы реального времени. Именно они позволили искусственному интеллекту перейти от игр и головоломок к массовому использованию при решении практически значимых задач.

### Состояние и тенденции развития искусственного интеллекта

Программные средства, базирующиеся на технологии и методах искусственного интеллекта, получили значительное распространение в мире. Их важность, и, в первую очередь, экспертных систем и нейронных сетей, состоит в том, что данные технологии существенно расширяют круг практически значимых задач, которые можно решать на компьютерах, и их решение приносит значительный экономический эффект. В то же время, технология экспертных систем является важнейшим средством в решении глобальных проблем традиционного программирования: длительность и, следовательно, высокая стоимость разработки приложений; высокая стоимость сопровождения сложных систем; повторная используемость программ и т.п. Кроме того, объединение технологий экспертных систем и нейронных сетей с технологией традиционного программирования добавляет новые качества к коммерческим продуктам за счет обеспечения динамической модификации приложений пользователем, а не программистом, большей "прозрачности" приложения (например, знания хранятся на ограниченном естественном языке, что не требует комментариев к ним, упрощает обучение и сопровождение), лучших графических средств, пользовательского интерфейса и взаимодействия.

По мнению специалистов [1], в недалекой перспективе экспертные системы будут играть ведущую роль во всех фазах проектирования, разработки, производства, распределения, продажи, поддержки и оказания услуг. Их технология, получив коммерческое распространение, обеспечит революционный прорыв в интеграции приложений из готовых интеллектуально-взаимодействующих модулей.

Коммерческий рынок продуктов искусственного интеллекта в мире в 1993 году оценивался примерно в 0,9 млрд. долларов; из них 600 млн. приходится на долю США [2]. Выделяют несколько основных направлений этого рынка:

1) экспертные системы; теперь их часто обозначают еще одним термином - "системы, основанные на знаниях";

2) нейронные сети и "размытые" (fuzzy) логики;

3) естественно-языковые системы.

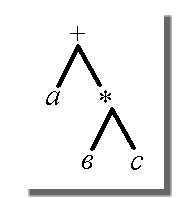
В США в 1993 году рынок между этими направлениями распределился так [2]: экспертные системы - 62%, нейронные сети - 26%, естественно-языковые системы - 12%. Рынок этот можно разделить и иначе: на системы искусственного интеллекта (приложения) и инструментальные средства, предназначенные для автоматизации всех этапов существования приложения. В 1993 году в общем объеме рынка США доля приложений составила примерно две, а доля инструментария - примерно одну треть [2].

Одно из наиболее популярных направлений последних пяти лет связано с понятием автономных агентов. Их нельзя рассматривать как "подпрограммы", - это скорее прислуга, даже компаньон, поскольку одной из важнейших их отличительных черт является автономность, независимость от пользователя. Идея агентов опирается на понятие делегирования своих функций. Другими словами, пользователь должен довериться агенту в выполнении определенной задачи или класса задач. Всегда существует риск, что агент может что-то перепутать, сделать что-то не так. Следовательно, доверие и риск должны быть сбалансированными. Автономные агенты позволяют существенно повысить производительность работы при решении тех задач, в которых на человека возлагается основная нагрузка по координации различных действий.

В том, что касается автономных (интеллектуальных) агентов, хотелось бы отметить один весьма прагматический проект, который сейчас ведется под руководством профессора Генри Либермана в Media-лаборатории MIT (MIT Media Lab). Речь идет об агентах, отвечающих за автоматическое генерирование технической документации. Для решения этой задачи немало сделал в свое время академик Андрей Петрович Ершов, сформулировавший понятие деловой прозы как четко определенного подмножества естественного языка, которое может быть использовано, в частности, для синтеза технической документации (это одно из самых узких мест в любом производстве). Группа под руководством профессора Либермана исследует возможности нового подхода к решению этой проблемы, теперь уже на основе автономных агентов.

Следующее направление в области искусственной жизни - генетическое программирование (genetic programming) - является попыткой использовать метафору генной инженерии для описания различных алгоритмов. Строки (string) искусственной "генетической" системы аналогичны хромосомам в биологических системах. Законченный набор строк называется структурой (structure). Структуры декодируются в набор параметров, альтернативы решений или точку в пространстве решений. Строки состоят из характеристик, или детекторов, которые могут принимать различные значения. Детекторы могут размещаться на разных позициях в строке. Все это сделано по аналогии с реальным миром. В природных системах полный генетический пакет называется генотипом. Организм, который образуется при взаимодействии генотипа с окружающей средой, носит название фенотипа. Хромосомы состоят из генов, которые могут принимать разные значения. (Например, ген цвета для глаза животного может иметь значение "зеленый" и позицию 10).

В генетических алгоритмах роль основных строительных блоков играют строки фиксированной длины, тогда как в генетическом программировании эти строки разворачиваются в деревья, столь знакомые специалистам в области трансляции. Например, выражение *a+b\*c* выглядит так:



Ныне одним из лидеров в области генетического программирования является группа исследователей из Стэндфордского университета (Stanford University), работающая под руководством профессора Джона Коза. Генетическое программирование вдохнуло новую жизнь в хорошенько уже подзабытый язык LISP (List Processing), который создавался группой Джона Маккарти (того самого, кто в 60-е годы ввел в наш обиход термин "искусственный интеллект") как раз для обработки списков и функционального программирования. Кстати, именно этот язык в США был и остается одним из наиболее распространенных языков программирования для задач искусственного интеллекта.

### Успехи систем искусственного интеллекта и их причины

Использование экспертных систем и нейронных сетей приносит значительный экономический эффект. Так, например:

- American Express [1] сократила свои потери на 27 млн. долларов в год благодаря экспертной системе, определяющей целесообразность выдачи или отказа в кредите той или иной фирме;

- DEC ежегодно экономит [1] 70 млн. долларов в год благодаря системе XCON/XSEL, которая по заказу покупателя составляет конфигурацию вычислительной системы VAX. Ее использование сократило число ошибок от 30% до 1%;

- Sira сократила затраты на строительство трубопровода в Австралии на 40 млн. долларов [3] за счет управляющей трубопроводом экспертной системы, реализованной на базе описываемой ниже системы G2.

Коммерческие успехи к экспертным системам и нейронным сетям пришли не сразу. На протяжении ряда лет (с 1960-х годов) успехи касались в основном исследовательских разработок, демонстрировавших пригодность систем искусственного интеллекта для практического использования. Начиная примерно с 1985 (а в массовом масштабе, вероятно, с 1988-1990 годов), в первую очередь, экспертные системы, а в последние два года и нейронные сети стали активно использоваться в реальных приложениях.

Причины, приведшие системы искусственного интеллекта к коммерческому успеху, следующие:

1. *Специализация.* Переход от разработки инструментальных средств общего назначения к проблемно/предметно специализированным средствам [4], что обеспечивает сокращение сроков разработки приложений, увеличивает эффективность использования инструментария, упрощает и ускоряет работу эксперта, позволяет повторно использовать информационное и программное обеспечение (объекты, классы, правила, процедуры).

2. *Использование языков традиционного программирования и рабочих станций.* Переход от систем, основанных на языках искусственного интеллекта (Lisp, Prolog и т.п.), к языкам традиционного программирования (С, С++ и т.п.) упростил "интегрированность" и снизил требования приложений к быстродействию и емкости памяти. Использование рабочих станций вместо ПК резко увеличило круг возможных приложений методов искусственного интеллекта.

3. *Интегрированность.* Разработаны инструментальные средства искусственного интеллекта, легко интегрирующиеся с другими информационными технологиями и средствами (с CASE, СУБД, контроллерами, концентраторами данных и т.п.).

4. *Открытость и переносимость.* Разработки ведутся с соблюдением стандартов, обеспечивающих данные характеристики [5].

5. *Архитектура клиент/сервер.* Разработка распределенной информационной системы в данной архитектуре позволяет снизить стоимость оборудования, используемого в приложении, децентрализовать приложения, повысить надежность и общую производительность, поскольку сокращается объем информации, пересылаемой между ЭВМ, и каждый модуль приложения выполняется на адекватном оборудовании.

Перечисленные причины могут рассматриваться как общие требования к инструментальным средствам создания систем искусственного интеллекта.

Из пяти факторов, обеспечивших их успех в передовых странах, в России, пожалуй, полностью не реализованы четыре с половиной (в некоторых отечественных системах осуществлен переход к языкам традиционного программирования, однако они, как правило, ориентированы среду на MS-DOS, а не ОС UNIX или Windows NT. Кроме того, в России и СНГ в ряде направлений исследования практически не ведутся, и, следовательно, в этих направлениях (нейронные сети; гибридные системы; рассуждения, основанные на прецедентах; рассуждения, основанные на ограничениях) нельзя ожидать и появления коммерческих продуктов.

Итак, в области искусственного интеллекта наибольшего коммерческого успеха достигли экспертные системы и средства для их разработки. В свою очередь, в этом направлении наибольшего успеха достигли проблемно/предметно специализированные средства. Если в 1988 году доход от них составил только 3 млн. долларов, то в 1993 году - 55 млн. долларов.

### Экспертные системы реального времени - основное направление искусственного интеллекта

Среди специализированных систем, основанных на знаниях, наиболее значимы экспертные системы реального времени, или динамические экспертные системы. На их долю приходится 70 процентов этого рынка.

Значимость инструментальных средств реального времени определяется не столько их бурным коммерческим успехом (хотя и это достойно тщательного анализа), но, в первую очередь, тем, что только с помощью подобных средств создаются стратегически значимые приложения в таких областях, как управление непрерывными производственными процессами в химии, фармакологии, производстве цемента, продуктов питания и т.п., аэрокосмические исследования, транспортировка и переработка нефти и газа, управление атомными и тепловыми электростанциями, финансовые операции, связь и многие другие.

Классы задач, решаемых экспертными системами реального времени, таковы: мониторинг в реальном масштабе времени, системы управления верхнего уровня, системы обнаружения неисправностей, диагностика, составление расписаний, планирование, оптимизация, системы-советчики оператора, системы проектирования.

Статические экспертные системы не способны решать подобные задачи, так как они не выполняют требования, предъявляемые к системам, работающим в реальном времени:

1. Представлять изменяющиеся во времени данные, поступающие от внешних источников, обеспечивать хранение и анализ изменяющихся данных.

2. Выполнять временные рассуждения о нескольких различных асинхронных процессах одновременно (т.е. планировать в соответствии с приоритетами обработку поступивших в систему процессов).

3. Обеспечивать механизм рассуждения при ограниченных ресурсах (время, память). Реализация этого механизма предъявляет требования к высокой скорости работы системы, способности одновременно решать несколько задач (т.е. операционные системы UNIX, VMS, Windows NT, но не MS-DOS).

4. Обеспечивать "предсказуемость" поведения системы, т.е. гарантию того, что каждая задача будет запущена и завершена в строгом соответствии с временными ограничениями. Например, данное требование не допускает использования в экспертной системе реального времени механизма "сборки мусора", свойственного языку Lisp.

5. Моделировать "окружающий мир", рассматриваемый в данном приложении, обеспечивать создание различных его состояний.

6. Протоколировать свои действия и действия персонала, обеспечивать восстановление после сбоя.

7. Обеспечивать наполнение базы знаний для приложений реальной степени сложности с минимальными затратами времени и труда (необходимо использование объектно-ориентированной технологии, общих правил, модульности и т.п.).

8. Обеспечивать настройку системы на решаемые задачи (проблемная/предметная ориентированность).

9. Обеспечивать создание и поддержку пользовательских интерфейсов для различных категорий пользователей.

10. Обеспечивать уровень защиты информации (по категориям пользователей) и предотвращать несанкционированный доступ.

Подчеркнем, что кроме этих десяти требований средства создания экспертных систем реального времени должны удовлетворять и перечисленным выше общим требованиям.

### Основные производители

Инструментарий для создания экспертных систем реального времени впервые выпустила фирма Lisp Machine Inc в 1985 году. Этот продукт предназначался для символьных ЭВМ Symbolics и носил название Picon. Его успех привел к тому, что группа ведущих его разработчиков образовала фирму Gensym, которая, значительно развив идеи, заложенные в Picon, выпустила в 1988 году инструментальное средство под названием G2. В настоящий момент работает его третья версия и подготовлена четвертая [1,7].

С отставанием от Gensym на два-три года ряд других фирм начал создавать (или пытаться создавать) свои инструментальные средства. Назовем ряд из них: RT Works (фирма Talarian, США), COMDALE/C (Comdale Techn., Канада), COGSYS (SC, США), ILOG Rules (ILOG, Франция). Сравнение двух наиболее продвинутых систем, G2 и RT Works, которое проводилось путем разработки одного и того же приложения двумя организациями, NASA (США) и Storm Integration (США) [10], показало значительное превосходство первой.

### Архитектура экспертной системы реального времени

Специфические требования, предъявляемые к экспертной системе реального времени, приводят к тому, что их архитектура отличается от архитектуры статических систем. Не вдаваясь в детали, отметим появление двух новых подсистем - моделирования внешнего окружения и сопряжения с внешним миром (датчиками, контроллерами, СУБД и т.п.) - и значительные изменения, которым подвергаются оставшиеся подсистемы.

Для того, чтобы понять, что представляет из себя среда для создания экспертных систем реального времени, опишем ниже жизненный цикл такой системы, а также ее основные компоненты. Описание оболочки экспертной системы реального времени приведем на примере средства G2, поскольку в нем полностью реализованы возможности, которые считаются необходимыми и уместными в подобных программных продуктах.

### Жизненный цикл приложения

Жизненный цикл приложения в G2 состоит из ряда этапов.

Разработка прототипа приложения

Разработчиком обычно является специалист в конкретной области знаний. Он в ходе обсуждений с конечным пользователем определяет функции, выполняемые прототипом. При разработке прототипа не используется традиционное программирование. Создание прототипа обычно занимает от одной до двух недель (при наличии у разработчика опыта по созданию приложений в данной среде. Прототип, как и приложение, создается на структурированном естественном языке, с использованием объектной графики, иерархии классов объектов, правил, динамических моделей внешнего мира. Многословность языка сведена к минимуму путем введения операции клонирования, позволяющей размножить любую сущность базы знаний.

Расширение прототипа до приложения

Конечный пользователь предлагает этапность проведения работ, направления развития базы знаний, указывает пропуски в ней. Разработчик может расширять и модифицировать базу знаний в присутствии пользователя даже в тот момент, когда приложение исполняется. В ходе этой работы прототип развивается до такого состояния, что начинает удовлетворять представлениям конечного пользователя. В крупных приложениях команда разработчиков может разбить приложение на отдельные модули, которые интегрируются в единую базу знаний.

Возможен и альтернативный подход к созданию приложения. При этом подходе каждый разработчик имеет доступ к базе знаний, находящейся на сервере, при помощи средства, называемого Telewindows, обычно расположенного на компьютереклиенте. В этом случае разработчики могут иметь различные авторизованные уровни доступа к приложению. Приложение может быть реализовано не только на различных ЭВМ, но и с использованием нескольких взаимодействующих оболочек G2.

Тестирование приложения на наличие ошибок

В G2 ошибки в синтаксисе показываются непосредственно при вводе конструкций (структур данных и исполняемых утверждений) в базу данных; эти конструкции анализируются инкрементно. Могут быть введены только конструкции, не содержащие синтаксических ошибок. Таким образом, отпадает целая фаза отладки приложения (свойственная традиционному программированию), что ускоряет разработку приложений.

Разработчик освобожден и от необходимости знать детальный синтаксис языка G2, так как при вводе в базу знаний некоторой конструкции ему в виде подсказки сообщается перечень всех возможных синтаксически правильных продолжений.

Для выявления ошибок и неопределенностей реализована возможность "Inspect", позволяющая просматривать различные аспекты базы знаний, например, "показать все утверждения со ссылками на неопределенные сущности (объекты, связи, атрибуты)", "показать графически иерархию заданного класса объектов", "показать все сущности, у которых значение атрибута Notes не ОК". (Данный атрибут есть у всех сущностей, представимых в языке G2; его значение - либо ОК, когда нет претензий к сущности, либо описание реальных или потенциальных проблем, например, ссылка на несуществующий объект, несколько объектов с одним именем и т.п.)

Тестирование логики приложения и ограничений (по времени и памяти)

Блок динамического моделирования позволяет при тестировании воссоздать различные ситуации, адекватные внешнему миру. Таким образом, логика приложения будет проверяться в тех условиях, для которых она создавалась. Конечный пользователь может принять непосредственное участие в тестировании благодаря управлению цветом (т.е. изменение цвета при наступлении заданного состояния или выполнения условия) и анимации (т.е. перемещение/вращение сущности при наступлении состояния/условия). Благодаря этому он сможет понять и оценить логику работы приложения, не анализируя правила и процедуры, а рассматривая графическое изображение управляемого процесса, технического сооружения и т.п.

Для проверки выполнения ограничений используется возможность "Meters", вычисляющая статистику по производительности и используемой памяти.

Полученное приложение полностью переносимо на различные платформы в среду UNIX (SUN, DEC, HP, IBM и т.д.), VMS (DEC VAX) и Windows NT (Intel, DEC Alpha). База знаний сохраняется в обычном ASCII-файле, который однозначно интерпретируется на любой из поддерживаемых платформ. Перенос приложения не требует его перекомпиляции и заключается в простом перемещении файлов. Функциональные возможности и внешний вид приложения не претерпевают при этом никаких изменений. Приложение может работать как в "полной" (т.е. предназначенной для разработки) среде, так и под runtime, которая не позволяет модифицировать базу знаний.

Сопровождение приложения

Не только сам разработчик данного приложения, но и любой пользователь может легко его понять и сопровождать, так как все объекты/классы, правила, процедуры, функции, формулы, модели хранятся в базе знаний в виде структурированного естественного языка и в виде графических объектов. Для ее просмотра используется возможность "Inspect". Сопровождение упрощается за счет того, что различным группам пользователей выдается не вся информация, а только ее часть, соответствующая их потребностям.

### Основные компоненты

Экспертная система реального времени состоит из базы знаний, машины вывода, подсистемы моделирования и планировщика.

### Базы знаний

Все знания в G2 хранятся в двух типах файлов: базы знаний и библиотеки знаний. В файлах первого типа хранятся знания о приложениях: определения всех объектов, объекты, правила, процедуры и т.п. В файлах библиотек хранятся общие знания, которые могут быть использованы более, чем в одном приложении, например, определение стандартных объектов. Файлы баз знаний могут преобразоваться в библиотеки знаний и обратно.

В целях обеспечения повторной используемости приложений реализовано средство, позволяющее объединять с текущим приложением ранее созданные базы и библиотеки знаний. При этом конфликты в объединяемых знаниях выявляются и отображаются на дисплее.

Знания структурируются: предусмотрены иерархия классов, иерархия модулей, иерархия рабочих пространств. Каждую из них можно показать на дисплее.

Сущности и иерархия классов

Класс, базовое понятие объектно-ориентированной технологии, - основа представления знаний в G2. Данный подход составляет основную тенденцию в программировании вообще, поскольку он уменьшает избыточность и упрощает определение классов (определяется не весь класс, а только его отличия от суперкласса), позволяет использовать общие правила, процедуры, формулы, уменьшает их число, да и является естественным для человека способом описания сущностей. При таком подходе структуры данных представляются в виде классов объектов (или определений объектов), имеющих определенные атрибуты. Классы наследуют атрибуты от суперклассов и передают свои атрибуты подклассам. Каждый класс (исключая корневой) может иметь конкретные экземпляры класса.

Все, что хранится в базе знаний и с чем оперирует система, является экземпляром того или иного класса. Более того, все синтаксические конструкции G2 являются классами. Для сохранения общности даже базовые типы данных - символьные, числовые, булевы и истинностные значения нечеткой логики - представлены соответствующими классами. Описание класса включает ссылку на суперкласс и содержит перечень атрибутов, специфичных для класса.

Иерархия модулей и рабочих пространств

Для структуризации G2-приложений используются "модули" и "рабочие пространства". Несмотря на то, что функции этих конструкций схожи, между ними есть существенные различия.

Приложение может быть организовано в виде одной или нескольких баз знаний, называемых модулями. В последнем случае говорят, что приложение представлено структурой (иерархией) модулей. На верхнем уровне - один модуль верхнего уровня. Модули следующего уровня состоят из тех модулей, без которых не может работать модуль предыдущего уровня. Структурирование приложений позволяет разрабатывать приложение одновременно нескольким группам разработчиков, упрощает разработку, отладку и тестирование, позволяет изменять модули независимо друг от друга, упрощает повторное использование знаний.

Рабочие пространства являются контейнерным классом, в котором размещаются другие классы и их экземпляры, например, объекты, связи, правила, процедуры и т.д. Каждый модуль (база знаний) может содержать любое число рабочих пространств. Рабочие пространства образуют одну или несколько древовидных иерархий с отношением "is-а-part-of" ("является частью"). С каждым модулем ассоциируется одно или несколько рабочих пространств верхнего (нулевого) уровня; каждое из них - корень соответствующей иерархии. В свою очередь, с каждым объектом (определением объекта или связи), расположенным в нулевом уровне, может ассоциироваться рабочее пространство первого уровня, "являющееся его частью" и т.д.

Различие между "модулями" и "рабочими пространствами" состоит в следующем. Модули разделяют приложение на отдельные базы знаний, совместно используемые в различных приложениях. Они полезны в процессе разработки приложения, а не его исполнения. Рабочие пространства, наоборот, выполняют свою роль при исполнении приложения. Они содержат в себе различные сущности и обеспечивают разбиение приложения на небольшие части, которые легче понять и обрабатывать.

Рабочие пространства можно устанавливать (вручную или действием в правиле/процедуре) в активное или неактивное состояние (при этом сущности, находящиеся в этом пространстве и в его подпространствах, становятся невидимыми для механизма вывода). Данный механизм используется, например, при наличии альтернативных групп правил, когда активной должна быть только одна из них.

Кроме того, рабочие пространства используются для определения пользовательских ограничений, определяющих разное поведение приложения для различных категорий пользователей.

Структуры данных

Сущности в базе знаний с точки зрения их использования можно разделить на структуры данных и выполняемые утверждения. Примерами первых являются объекты и их классы, связи (connection), отношения (relation), переменные, параметры, списки, массивы, рабочие пространства. Примерами вторых - правила, процедуры, формулы, функции.

Выделяют объекты (и классы), встроенные в систему и вводимые пользователем. При разработке приложения, как правило, создаются подклассы, отражающие специфику данного приложения. Среди встроенных подклассов объектов наибольший интерес представляет подкласс данных, включающий подклассы переменных, параметров, списков и массивов.

Особая роль отводится переменным. В отличие от статических систем переменные делятся на три вида: собственно переменные, параметры и простые атрибуты. Параметры получают значения в результате работы машины вывода или выполнения какой-либо процедуры. Переменные представляют измеряемые характеристики объектов реального мира и поэтому имеют специфические черты: время жизни значения и источник данных. Время жизни значения переменной определяет промежуток времени, в течение которого это значение актуально, по истечении этого промежутка переменная считается не имеющей значения.

Выполняемые утверждения

Основу выполняемых утверждений баз знаний составляют правила и процедуры. Кроме того, есть формулы, функции, действия и т.п. Правила в G2 имеют традиционный вид: левая часть (антецедент) и правая часть (консеквент). Кроме if-правила, используется еще четыре типа правил: initially, unconditionally, when и whenever. Каждое из типов правил может быть как общим, т.е. относящимся ко всему классу, так и специфическим, относящимся к конкретным экземплярам класса.

Возможность представлять знания в виде общих правил, а не только специализированных, позволяет минимизировать избыточность базы знаний, упрощает ее наполнение и сопровождение, сокращает число ошибок, способствует повторной используемости знаний (общие правила запоминаются в библиотеке и могут использоваться в сходных приложениях).

Несмотря на то что продукционные правила обеспечивают достаточную гибкость для описания реакций системы на изменения окружающего мира, в некоторых случаях, когда необходимо выполнить жесткую последовательность действий, например, запуск или остановку комплекса оборудования, более предпочтителен процедурный подход. Язык программирования, используемый в G2 для представления процедурных знаний, является достаточно близким родственником Паскаля. Кроме стандартных управляющих конструкций язык расширен элементами, учитывающими работу процедуры в реальном времени: ожидание наступления событий, разрешение другим задачам прерывать ее выполнение, директивы, задающие последовательное или параллельное выполнение операторов. Еще одна интересная особенность языка - итераторы, позволяющие организовать цикл над множеством экземпляров класса.

### Машина вывода, подсистема моделирования и планировщики

Главный недостаток прямого и обратного вывода, используемых в статических экспертных системах, - непредсказуемость затрат времени на их выполнение. С точки зрения динамических систем, полный перебор возможных к применению правил - непозволительная роскошь. В связи с тем, что G2 ориентирована на приложения, работающие в реальном времени, в машине вывода должны быть средства для сокращения перебора, для реакции на непредвиденные события и т.п. Для машины вывода G2 характерен богатый набор способов возбуждения правил. Предусмотрено девять случаев:

1. Данные, входящие в антецедент правила изменились (прямой вывод - forward chaining).

2. Правило определяет значение переменной, которое требуется другому правил или процедуре (обратный вывод - backward chaining).

3. Каждые n секунд, где n - число, определенное для данного правила (сканирование - scan).

4. Явное или неявное возбуждение другим правилом - путем применения действий фокусирования и возбуждения (focus и invoke).

5. Каждый раз при запуске приложения.

6. Входящей в антецедент переменной присвоено значение, независимо от того, изменилось оно или нет.

7. Определенный объект на экране перемещен пользователем или другим правилом.

8. Определенное отношение между объектами установлено или уничтожено.

9. Переменная не получила значения в результате обращения к своему источнику данных.

Если первые два способа достаточно распространены и в статических системах, а третий хорошо известен как механизм запуска процедур-демонов, то остальные являются уникальной особенностью системы G2. В связи с тем, что G2-приложение управляет множеством одновременно исполняемых задач, необходим планировщик. Хотя пользователь никогда не взаимодействует с ним, планировщик контролирует всю активность, видимую пользователем, и активность фоновых задач. Планировщик определяет порядок обработки задач, взаимодействует с источниками данных и пользователями, запускает процессы и осуществляет коммуникацию с другими процессами в сети.

Подсистема моделирования G2 - достаточно автономная, но важная часть системы. На различных этапах жизненного цикла прикладной системы она служит достижению различных целей. Во время разработки подсистема моделирования используется вместо объектов реального мира для имитации показаний датчиков. Очевидно, что проводить отладку на реальных объектах может оказаться слишком дорого, а иногда (например, при разработке системы управления атомной станцией) и небезопасно.

На этапе эксплуатации прикладной системы процедуры моделирования выполняются параллельно функциям мониторинга и управления процессом, что обеспечивает следующие возможности:

- верификация показаний датчиков во время исполнения приложения;

- подстановка модельных значений переменных при невозможности получения реальных значений (выход из строя датчика или длительное время реакции на запрос).

Как видим, играя роль самостоятельного агента знаний, подсистема моделирования повышает жизнеспособность и надежность приложений. Для описания внешнего мира подсистема моделирования использует уравнения трех видов: алгебраические, разностные и дифференциальные (первого порядка).

### Заключение

Рассмотренные в статье тенденции развития искусственного интеллекта позволяют утверждать, что одним из основных направлений в этой области являются экспертные системы реального времени. Рассмотрение проведено на примере оболочки экспертных систем реального времени G2, представляющей собой самодостаточную среду для разработки, внедрения и сопровождения приложений в широком диапазоне отраслей. G2 объединяет в себе как универсальные технологии построения современных информационных систем (стандарты открытых систем, архитектура клиент/сервер, объектно-ориентированное программирование, использование ОС, обеспечивающих параллельное выполнение в реальном времени многих независимых процессов), так и специализированные методы (рассуждения, основанные на правилах, рассуждения, основанные на динамических моделях, или имитационное моделирование, процедурные рассуждения, активная объектная графика, структурированный естественный язык для представления базы знаний), а также интегрирует технологии систем, основанных на знаниях с технологией традиционного программирования (с пакетами программ, с СУБД, с контроллерами и концентраторами данных и т.д.).

Все это позволяет с помощью данной оболочки создавать практически любые большие приложения значительно быстрее, чем с использованием традиционных методов программирования, и снизить трудозатраты на сопровождение готовых приложений и их перенос на другие платформы.

### Литература

*[1] F. Hayes-Roth, N. Jacobstein. The State of Enowledge-Based Systems. Communications of the АСМ, March, 1994, v.37, n.3, рр.27-39.*

*[2] Р. Harmon. The Size of the Commercial AI Market in the US. Intelligent Software Strategies. 1994, v.10, n.1, рр. 1-6.*

*[3] Expert system saves 20 million L on pipeline management. C&I July, 1994, р.31.*

*[4] Р. Harmon. The Market for Intelligent Software Products. Intelligent Sopware Strategies 1992, v.8, n.2, рр.5-12.*

*[5] D.R Perley. Migrating to Open Systems: Taming he Tiger. McGraw-Hill, 1993, р.252.*