Александр Горбачев

Модели информации и данных

Атом и универсум информации

Александр Горбачев

Модели информации и данных. Атом и универсум информации

«Издательские решения»

Горбачев А. М.

Модели информации и данных. Атом и универсум информации / А. М. Горбачев — «Издательские решения»,

ISBN 978-5-44-931338-6

Развитие информационных технологий происходит очень высокими темпами. Вместе с тем анализ фундаментальных моделей данных и информационных моделей позволит переосмыслить подходы к проектированию информационных систем и построению интеллектуальных систем. В книге освещается большое количество аспектов по управлению информацией и данными, исследуется универсальная информационная модель и ее сопоставление с моделями данных, переход от данных к информации и обратно.

ISBN 978-5-44-931338-6

Содержание

Модели информации и данных
Атом и универсум информации

Александр Михайлович Горбачев

ISBN 978-5-4493-1338-6
Создано в интеллектуальной издательской системе Ridero

Введение

Информационные компьютерные технологии пришли в этот мир в конце 1940-х годов. До начала 1980-х годов они оставались уделом небольшого числа профессионалов, но с возникновением персональных компьютеров они широко проникли в нашу жизнь, и в том числе в бытовую сферу. Увеличение уровня интеграции микропроцессоров и развитие телекоммуникационных технологий привели к тому, что современные мобильные телефоны-смартфоны стали компьютерами, несравненно более мощными как по производительности, так и по ресурсам даже по сравнению с самыми развитыми компьютерами 80-х годов минувшего века. Каждый год изобретаются новые материалы, технологии, всё увеличиваются масштабы производительности, объем и разнообразие ресурсов в электронных устройствах, а через это – увеличиваются объемы хранимых и передаваемых данных, число человеко-машинных интерфейсов и, как следствие, глубина вхождения и широта использования устройств и гаджетов в жизни современного человека и бизнеса.

В октябре 2003 года по обвинению в хищениях и неуплате налогов был арестован самый богатый человек России Михаил Ходорковский. Михаил провел в тюрьме более 10 лет. Пробыв эти годы в заключении, он пропустил бум сенсорных смартфонов и начало эпохи планшетов. После освобождения из заключения одним из его самых первых желаний было получить iPhone и iPad компании Apple. Это яркий пример современности, показывающий, что означают технологии и как много места занимают устройства в жизни современного человека. На сегодняшний день самыми дорогими компаниями, судя по их биржевой стоимости, являются Apple и Google, отодвинув на третью строчку нефтяной ExxonMobil.

И, тем не менее, не потребовалось никакого внешнего вмешательства в то, чтобы разрушить Вавилонскую башню компьютерных технологий, ибо её не существовало испокон веков. Аппаратные и программные платформы, языки программирования и форматы данных настолько несовместимы между собой, что самая невесомая материя, на которую не распространяются даже таможенные пошлины – информация, остается жестко связанной с конкретными платформами, в то время как межплатформенные коммуникационные технологии слишком отстали от всех остальных технологий в IT-сфере. До сих пор определение правил и последовательности интеграции как средства коммуникации между различными программными приложениями является результатом полу-ручного труда. Технологии построения автоматической трансформации данных между различными платформами отсутствуют, вместо этого существуют эмуляторы сред, технологии виртуализации. В многообразии решений люди уже отказываются что-либо понимать в технологической подложке, а потому IT-корпорации им настойчиво предлагают SaaS (программный продукт как услуга), IaaS (инфраструктура как услуга) и технологии виртуализации. Впрочем, и эти решения не являются панацеей, поскольку они лишь завертывают всё те же программные продукты в новую упаковку и позволяют закрыть глаза и не замечать технологии, лежащие в основе информационных систем. Эволюционное развитие информационных компьютерных технологий привело к тому, что на сегодняшний день они уже созрели в достаточной степени. Это выражается в замедлении развития операционных систем, сред программирования и систем управления базами данных. В отдельных областях технологии даже перезрели, что отражается на сложности программирования систем, использующих сервисно-ориентированную архитектуру (SOA), базирующихся на web-сервисах и т. д. Как результат – лавинообразное увеличение числа framework'ов (программных сред, для создания пользовательских интерфейсов) и существенное увеличение времени тестирования систем при их разработке на основе этих технологий.

Не кажется ли Вам, что закон Мура и неостанавливающийся лавинообразный технический прогресс в виде непрерывного наращивания аппаратных компьютерных ресурсов с одной

стороны, и несовершенство программных IT-технологий с другой стороны, выглядят как странная дисгармония? В этой книге я постараюсь сделать серьезный шаг к её разрушению. За прошедший десяток лет я сформировал некоторый набор, с моей точки зрения, интересных мыслей – теоретических знаний и практических наблюдений, которых я не встречал в законченном или в достаточной степени оформленном виде ни в программных реализациях, ни в теоретических выкладках. В своих измышлениях я не претендую на роль пионера и даже на оригинальность, поскольку, как сказал в 1899 году специальный уполномоченный американского бюро патентов Чарльз Дюэлл: «Все, что могло быть изобретено, уже изобрели» – похоже, примерно так и есть, за исключением деталей. Я постарался собрать мысли воедино, и предоставить вам квинтэссенцию своих записей, наблюдений и исследований в концентрированном виде в этой монографии.

Более 15 лет моя работа связана с ERP-системами. В последние несколько лет, сталкиваясь с вопросами и претензиями пользователей к несовершенству программных продуктов, а также при общении с коллегами, я не устаю повторять идею о том, что, скорее всего, современные информационные технологии находятся еще на столь ранней стадии и в настолько несовершенном виде, что множество вещей, простых и понятных в естественном мире, являются слишком технологически сложными и недостижимыми в программных продуктах. Прежде всего, это касается обмена данными между различными программными продуктами, трансформации данных, платформозависимости программных продуктов, существенной ограниченности в модификации структур данных и процессов, и пр. Также это касается и реализации довольно простых процессов в понимании людей, далёких от программирования. Подобных примеров можно привести воз и маленькую тележку. В результате за понятными вещами и процессами в естественном понимании, на бытовом уровне и даже в базовом профессиональном уровне могут стоять очень сложные и ресурсоёмкие технологии. Анализ, проектирование и изменение программных систем стоят конечным потребителям сотни долларов за час работы консультантов и программистов. Добавление колонки в отчет может потребовать 8 часового дня работы программиста. Все это регулярно вызывает закономерные вопросы пользователей «Почему так дорого?» и «Почему так медленно делаются изменения?». Альтернативой для пользователей является только добровольное залезание в прокрустово ложе коробочных решений вместо настройки системы под конкретные требования компаний, что является жестоким компромиссом, и подавляет волю, разум и веру в информационные технологии людей, привыкших к свободе мысли.

Эта книга посвящена обзору существующих технологий, и затем в ней сделана попытка воспроизвести информационный универсум – объемлющую информационную модель, которая сможет объединить и примирить разные технологии, сделать их прозрачными и взаимо-трансформируемыми.

Программные технологии становятся все сложнее с каждым днём (хотя и дают большие возможности), а программисты прошлого века, столкнись они только с одним пластом современного программирования от web-систем до баз данных, несомненно, испытали бы шок и впали бы в депрессию от обилия требуемых знаний о системах программирования. Признак сложности не является признаком совершенства, а требование наличия определенного объема знаний не означает, что система будет работать лучше по сравнению с системами на компьютерах прошлого столетия. И, напротив, складывается тенденция того, что при создании распределенных, веб-ориентированных, модульных, трехзвенных программных систем, тестирование становится все сложнее, обслуживание накладных процессов коммуникации между составляющими системы, пользовательским интерфейсом и пр. отнимает все больше сил разработчика. В этом смысле я придерживаюсь идей демокритовского атомизма, пытаясь разложить составляющие информационных технологий на элементарные частицы – атомы, отбросив, таким образом, чисто технологические детали, не добавляющие ценности в конечный

результат или же имеющие утилитарные и побочные функции. И через эту «технику» я стремлюсь разобраться и переосмыслить начала информационных процессов вновь. Я попрошу вас не сравнивать атомизм, как подход по отбрасыванию незначащих деталей, и разбор предметов «по винтикам», с наивным искусством, которое рассматривает мир через примитивное представление.

Я предполагаю, что читатели обладают базовыми знаниями в области информатики, управления базами данных, а также в области логических методов искусственного интеллекта. Несмотря на то, что я пытаюсь достаточно просто и подробно описать суть вещей и привести достаточное количество примеров, тем не менее, неподготовленного читателя могут смутить ссылки на некоторые технологии, которые никак не возможно было бы описать в этой книге. Впрочем, при наличии огромного количества информации в Интернете, я думаю, что отсылка к терминам, системам и технологиям не станет препятствием для неосведомленного в чем-то читателя, если в нем присутствует пытливость ума.

Я вполне понимаю, что можно найти множество произведений подобной направленности с поиском рецепта для всеобъемлющего решения любых задач – в том числе записи в блогах и на интернет сайтах, статьи, научные работы и книги. Также я не хочу навязывать вам какую-то определенную программную реализацию со словами «вот видите, это настоящий интеллект, настоящая реализация такой-то графовой модели». Вместе с тем, мы с вами уже видели не один десяток таких «конкретных» программ и книг, каждый раз непременно разочаровываясь. Поэтому, задумывая эту книгу, я старался обратиться к основам основ и дать детальную аргументацию, почему меня волнуют одни технологии и менее симпатичны другие, а также какие взгляды существуют на одну и ту же проблематику. И всё-таки в основном я хочу высказать свою собственную точку зрения, а не описывать среднестатистический взгляд, за долгие годы ставший нормой.

Дело в том, что, несмотря на обилие интеллектуальных технологий, в большинстве из них отсутствует такая важная составляющая, как полнота. Полнота требует видения того, как система будет работать вообще, в целом. Является ли она жизнеспособной, не решает ли она какую-то маленькую частичку общей задачи. Дает ли она возможность быть не только «вещью в себе», но и позволяет ли через себя развиваться другим технологиям. Я постарался дать полную картину. Что получилось – судить Вам.

Для чего эта книга

Постановка задачи – один из самых важных разделов любого описания, любой теории, любой практической задачи. Откройте какую-нибудь книгу, посвященную бизнес-процессам, внедрению программного обеспечения или управлению проектами, и Вы увидите одно и то же правило: «максимально точно определяйте цель». Самыми дорогими являются ошибки и упущения, сделанные на этапе проектирования системы. Непродуманность результата сказывается на качестве и на эффективности теории. Поэтому начнем с цели.

Целью этой книги является описание модели данных, которая будет способствовать созданию интеллектуальных информационных систем. Я бы написал об «универсальной информационной системе», но это было бы слишком похоже на «сферического коня в вакууме», то есть на выхолощенную систему, якобы рассчитанную на любую сферу применения, но не реализующую ничего конкретного. Так что информационная система может представлять собой платформу или набор для создания систем, например, библиотеку или программный интерфейс. Такая платформа призвана сделать шаг вперед для улучшения существующих прикладных систем. Тем не менее, далее я буду применять термин «интеллектуальная или универсальная информационная система».

На уровне современных информационных систем модель данных, описываемая в этой книге, позволит системам гораздо гибче отражать их информационное наполнение, развить собственный словарь данных, снять жесткие ограничения на структуру данных, заложенную в современных моделях данных, при этом, не отрицая принципы их построения. Предлагаемая модель данных позволит модифицировать собственные структуры данных к индивидуальным требованиям пользователей. То есть, позволит не стоять им на месте, а развиваться органически и непрерывно в противоположность «методу водопада», который в настоящее время применяется в 95% систем для изменения структур данных.

К интеллектуальной информационной системе мы предъявляем ряд требований. Оговорюсь, что «интеллектуальная информационная система» на сегодняшний день выглядит как нечто несбыточное. Но, пожалуй, пройдёт 7—15 лет и то, что сегодня мы подразумеваем под «интеллектуальной» системой, станет самой ординарной и естественной информационной системой. При этом часть следующих требований относится к самому сегодняшнему определению «интеллектуальности» системы. Итак, интеллектуальная система должна обладать следующими свойствами:

– способность решать задачи различного уровня и свойства, то есть универсальные задачи. Система, обладающая знаниями о том, как решается задача, должна решать её на основе исходных данных универсальной структуры и универсальные интерфейсы взаимодействия с пользователем,

– способность найти решение задачи. Если система не может решить определенную задачу, но система обладает знаниями о решении более абстрактной задачи, части задачи или похожей задачи, она должна сформировать алгоритм решения задачи,

– способность запросить и получить решение задачи. Если система не представляет, каким образом она может решить задачу, либо ей не хватает данных для решения задачи, то система должна обладать возможностями узнать у другой похожей системы, как решать задачу,

– способность обучаться. Система с учетом уже полученных знаний должна их накапливать, и при необходимости пользоваться ими, не запрашивая вновь,

– способность планировать действия. Система должна иметь возможность декомпозировать задачи и вырабатывать план действий,

– способность анализировать ситуацию. Система должна анализировать, интерпретировать внешнюю ситуацию и понимать, каким образом внешняя ситуация измениться при влиянии на неё.

В общем и целом, эти цели системы сводятся к ряду задач, связанных с логическими операциями, операциями интерпретации данных, операциями по структурированию данных.

Основной вопрос в контексте данной книги: а для чего нужна такая система? Есть тысячи специализированных систем. Но день за днём продолжают появляться позаказные системы, а это означает, что специализированные системы не в состоянии решать задачи с должной мерой универсальности, гибкости и масштабируемости. Одним из ярких примеров являются системы управления проектами. В каждой из них можно найти «червоточину», несмотря на то, что управление проектами – чрезвычайно широко распространенная область, требующая автоматизации.

Также есть гораздо более эффективный механизм, чем какая-то универсальная система – это человек, которые отлично справляется с этими задачами. Однако по сравнению с человеком компьютер обладает свойствами, которые человеку неподвластны:

– точность запоминания данных и долгосрочность памяти. Со временем человек способен забывать информацию, в том числе и важную информацию. Компьютер хранит данные так, как они были сохранены.

– объем хранения данных. Компьютер может хранить множество данных, которые невозможно запомнить человеку. Например, в системах управления персоналом компьютерные системы могут хранить данные по десяткам тысяч сотрудников, когда-либо работавших в компании. Ни один человек не может сделать подобного.

– собственность. Компьютер является собственностью, то есть определенный человек и компания может быть уверены, что данные будут принадлежать человеку и компании.

– доступность. Информация, хранящаяся на компьютере доступна 24 часа в сутки 365 дней в неделю.

– надежность. Как говорит один из законов Мэрфи «Компьютеры ненадежны, но люди еще ненадежнее». Это свойство является следствием из принципов доступности и определенности структур данных.

– определенность структур данных. Компьютерная система должна позволять проанализировать себя пользователю. Внутренние механизмы компьютера должны быть понятны и прозрачны, то есть анализируемы. Напротив, форма хранения знаний человеком неопределенна. Механизмы хранения знаний человеком непрозрачны и недоступны для получения извне.

Разумеется, невозможно сравнивать компьютер и человека. Такое сравнение неизменно напоминает анекдоты вроде «чем девушка отличается от телевизора…» или же «чем отличаются орехи от мужа…». Тем не менее, использование универсальной информационной системы может дать следующие возможности:

– создание гибких и динамических систем принятия решения/управления некоторыми процессами и данными,

– выявление неочевидных связей между данными,

– существенное снижение затрат на программирование систем,

– создание баз знаний, модифицирующихся структур данных и самоорганизующихся процессов в информационных системах,

– существенное упрощение получения, интерпретации и формирования данных при взаимодействии системы с другими системами и с окружающим миром.

Реализация этих обширных планов невозможна без участия двух составляющих системы – данных и процессов, реализующих обработку данных. Эта пара реализует дуализм в информационных системах, выражающих статику и движение, соответственно. Модели данных, используемые на сегодняшний момент, не позволяют в достаточной степени удовлетво-

рить требованиям, которые предъявят интеллектуальные информационные системы к этим моделям в силу множества ограничений по гибкости, заструктурированности и пр. Эти ограничения будут рассмотрены ниже. Получается, что жесткая структура данных, распространяет свое статическое влияние на процессы. Системам непременно потребуется новая модель данных, не отрицающая устоев существующих структур данных, но предлагающая принципиально новые подходы. Такая модель и является целью настоящей книги.

К сожалению, в теориях, описывающих глобальное взаимодействие множества агентов (независимых участников какой-либо системы, среды или социума) часто прослеживается идея «интеллектуального супа» или самозарождения жизни. Например, к таковым относятся макроэкономические теории, а в особенности теории, связанные с интеллектуальными технологиями. Некоторые теории предполагают, что достаточно создать некоторые начальные условия и структуры, и по прошествии некоторого времени мы сможем получить саморазвивающуюся интеллектуальную систему без дополнительных усилий по созданию такой системы. Такие теории считают, что для создания интеллектуальной системы достаточно создать «первичный бульон», а готовая система будет самостоятельным результатом эволюционных факторов и внешних влияний.

Анализ результатов работы подобных подходов говорит, что на практике в результате работы таких моделей никаких реальных результатов не появляется. «Интеллектуальный бульон» остаётся лишь набором ингредиентов, если только его не поражает гниль и плесень. Жизнь или не рождается сама, или ситуация с самостоятельным зарождением жизни похожа на **теорему о бесконечных обезьянах** («абстрактная обезьяна, ударяя случайным образом по клавишам пишущей машинки в течение неограниченно долгого времени, рано или поздно напечатает любой заданный текст»). Зато чаще всего такие декларации говорят о том, что теория недостаточно проработана и там, где теоретически должна зародиться самостоятельная жизнь, представления самих авторов теории о том, что же конкретно должно произойти в том месте, заканчиваются.

В этой книге я постараюсь ограничить себя от подобных недоработок.

Ограничения

Область действия интеллектуальных систем чрезвычайно широка – от распознавания текстов и речи до роботизированных систем.

Большая часть систем ориентирована на коммуникацию с человеческими системами – с письменностью, устной речью, лексическим анализом и пр. Обычно они производят наиболее яркое впечатление при демонстрации. Робот, моделирующий поведение человека вызывает ощущение, что перед тобой человек. В большинстве же случаев такая реализация – это не более чем моделирование отдельных составляющих человека, которые теряют смысл без всех прочих неотъемлемых частей, таких как коммуникация, хранение информации и пр.

В этой книге я сосредоточусь в большей степени на низкоуровневых механизмах, которые как раз позволят связать подходы естественного анализа, такие как логика, механизмы обучения, формирование потребностей, с техническими основами, такими как управление операциями и системы управления базами данных.

Вследствие этого все остальные части общей системы, будут отодвинуты на второй план как менее значимые. Например, всё взаимодействие с человеком будет рассматриваться на уровне традиционных экранных интерфейсов, а вопросы синтаксического анализа, например, анализ падежей и других конструкций языка, будут опущены как производные.

Картина мира

Агенты

Прежде чем переходить к частным вещам, таким как идентификация данных или связь лексических единиц, рассмотрим системы с точки зрения макро-элементов. То есть, рассмотрим среду, в которой существует информация, данные и информационные процессы. Это необходимо для определения этимологии информации, определения общих правил её распространения и для получения обобщенного взгляда на информационные процессы.

Так же, как и в окружающем нас мире, в интеллектуальных системах можно выделить несколько принципов, которые влияют на взаимодействие систем на макро-уровне. Разумеется, можно не брать во внимание это общее представление, абстрагироваться от него. В таком случае мы упростим систему, и тогда она будет ограничена только поставленными перед ней задачами обработки данных, и не будет ориентироваться на коммуникацию с внешними системами. В таком случае внешние системы, и как частный случай этих систем – человек, должны будут подстраиваться под эту систему, которая в свою очередь дистанцируется от взаимодействия с другими системами. Такая система, очевидно, будет похожа на образец систем 60-х годов прошлого века. В частности, Норберт Винер в своей «Кибернетике» писал: «В идеальную вычислительную машину все данные надо вводить сразу же в начале работы, и затем до самого конца она должна по возможности быть свободна от человеческого вмешательства. Это значит, что машина должна получить в начале работы не только все числовые данные, но и все правила их соединения, в виде инструкций на любую ситуацию, которая может возникнуть в ходе вычислений.» [1]. Для любого современного пользователя программных систем этот подход выглядит более чем анахронично в силу своей чрезвычайной закрытости и отсутствия интерактивности.

Мы же рассматриваем систему, которая должна иметь возможность постоянно развиваться, и которая будет способна к универсальности в выполнении задач. Под универсальностью понимается их возможность встраиваться в общую экологическую систему своего существования (общество).

Именно универсальность в интеллектуальных системах является наиболее ценным свойством, она означает наибольшую гибкость системы, её способность, с одной стороны, улавливать, воспринимать, распознавать, усваивать внешние данные и, с другой стороны, формировать ответы, реакции, генерировать информацию и действия, которые вписываются в общепринятую структуру знаний. Общепринятая структура знаний – это система понятий, терминов, методов и теорий, принятых в обществе.

Как видно из определения, основными свойствами системы являются коммуникативные способности в социуме. Другими словами, рассматривается не какая-то расчетная функция, не интеллектуальный механизм, а способность коммуницировать в общей среде. В то же время, отдельный элемент системы, обладающий только коммуникативными свойствами, является вырожденным, поскольку не несет в себе никакой ценности, связанной с выборкой, хранением, структурированием и анализом информации. Однако «разум в себе», не имеющий достаточных коммуникативных качеств, также является вырожденной системой. Но и так же можно сказать, что невозможно поддерживать приемлемый уровень коммуникации без обработки информации. В частности, система не сможет ответить на вопросы к ней, если у неё не будет хранилища информации, системы выборок информации и т. п.

В качестве примера, газеты с объявлениями или доски объявлений можно рассматривать не более чем как обособленные коммуникативные системы. Однако и газеты рекламных

объявлений, и другие способы коммуникации структурируют и формализуют информацию – через формат газеты, рубрики и разделы, формат объявлений.

Для рассмотрения «общества» интеллектуальных систем наилучшим образом подходит теория **многоагентных систем (MAC, multi-agent system)**. Это система, которая образована несколькими взаимодействующими интеллектуальными агентами. Интеллектуальный агент – это некоторая сущность, наблюдающая за окружающей средой или действующая в ней. Такой агент может быть роботом, программной системой, человеком и пр. Коммуникативная часть является определяющей, однако раз мы исследуем компьютерные системы, большее внимание мы будем обращать на агентов как на программные системы.

Агенты разделяются на агентов с простым поведением, агенты с модельным поведением, целенаправленные агенты, практичные агенты, обучающиеся агенты и т. д.

В теории мультиагентных систем отдельно выделяются субагенты. Субагент – это часть агента, которая может быть выделена в специализированную подсистему. Так, существуют:

– временные субагенты для принятия оперативных решений,

– пространственные агенты для взаимодействия с реальным миром,

– обучающие агенты и т. д.

Субагенты могут быть различного назначения, и в большей степени они разделяются исходя из процессов и архитектуры самого агента.

Агенты в многоагентной системе должны иметь несколько важных характеристик:

– автономность,

– ограниченность представления. То есть, ни у одного агента нет представления обо всей системе,

– децентрализованность, то есть, в системе нет агентов, управляющей всей системой.

В многоагентной среде отдельные агенты имеют возможность получать и формировать информационные поля в виде сообщений между собой. Не обязательно, что все агенты являются равными по ролям, по правам, по возможностям, по зависимостям друг от друга, по доступности или открытости взаимодействия с другими агентами. Например, в многоагентной среде наравне с интеллектуальными агентами могут существовать агенты для обмена данными, такие как поисковые сервера, публичные хранилища для обмена данными и для получения данных.

Агенты как черный ящик

В многоагентной среде на первый план выходит взаимодействие между агентами. И на второй план отходит реализация этих агентов. Главное, чтобы агенты поддерживали общепринятый протокол обмена информацией, а как устроены эти агенты внутри и из чего они состоят, по большому счету, не имеет значения.

Таким образом, агенты предстают перед нами в виде черного ящика. Мы знаем, что они общаются с нами, но не знаем, кто они такие, какие механизмы обработки данных лежат в их основе, какую информацию они хранят в себе.

Большое количество современных программ и отдельных обработок в программах представляется нам аналогичным образом. Они являются закрытыми: на их входе существует некоторое количество исходных данных, на выходе – некоторый результат. С развитием интерактивных программ, некоторые обработки в рамках программных систем становятся еще менее прозрачными для пользователя, поскольку ему не всегда понятно, какие данные системы являются исходными для обработки, куда сохраняется результат и почему получился именно такой результат. Например, при расчете остатка дней отпуска сотрудника в системе управления персоналом система может брать (или не брать) в расчет дату приема сотрудника, признак ненормированного рабочего дня, отпуска, взятые за свой счёт более 7 дней, отпуска по уходу за ребенком. Количество параметров столь велико, что нельзя быть уверенным, что процедура расчета полностью возьмет все из них, и корректно рассчитает количество дней права на отпуск. Результатом работы такой процедуры является остаток дней. Но система может их хранить в нескольких таблицах (например, по рабочим годам сотрудника и общее количество дней в целом). И нет никакой гарантии, что процедура расчета верно запишет результаты во все таблицы, и что данные в этих таблицах будут непротиворечивы. Поскольку данные хранятся во внутренних таблицах системы, проверить их простому пользователю практически невозможно (лишь используя специальные отчеты) так же, как и невозможно проверить правильность работы процедуры – от параметров до логики.

Тем не менее, принцип «черного ящика» хорош, если мы хотим абстрагироваться от существа обработок, и сосредоточиться на вопросах коммуникации между системами, либо на вопросах предоставления и получения некоторой информации. Этот принцип может быть полезен при отделении части процессов на уровень субагентов. При проектировании систем содержимое «черного ящика» обычно заменяется элементарным (простейшим) процессом или заглушкой.

Тест Тьюринга

Поскольку агенты определяются именно своими коммуникационными способностями, в этом контексте невозможно не упомянуть тест Тьюринга. Кроме того, в этой главе я хочу определиться со своим отношением к общему понятию искусственного интеллекта.

Основную идею общего понятия «искусственный интеллект» в 1950 году сформировал Алан Тьюринг, автоматически став его основоположником. В журнале Mind Тьюринг описал тест на интеллект. Тест основан на взаимодействии человека (следователя) и компьютера. Общение происходит в изолированных комнатах посредством компьютерного терминала. Следователь задает вопросы и получает ответы от своего собеседника. Следователь не знает, общается он с человеком или с компьютером. Смысл теста Тьюринга заключается в том, чтобы признать, что компьютер обладает интеллектом, если следователь не смог раскрыть компьютер в качестве собеседника.

Конечно, по прошествии почти 60 лет будет неправильно говорить о корректности замещения определения «разумности» определенным тестом. До сих пор ни одна машина не в состоянии пройти тест Тьюринга. Но нужно ли проходить этот тест?

Тест Тьюринга преследует цель выявления внутреннего разума компьютера. В то же время для смазывания разницы в человеко-машинной коммуникации был выбран метод общения посредством компьютерного терминала.

Между тем, проблема во взгляде на «машинный интеллект» всё-таки существует. Она заключается в том, что под влиянием этого теста «искусственный интеллект» становится идентичным понятию «эмуляция человека». Причём от машинного интеллекта требуется точного воспроизведения всех черт, присущих человеку, включая ошибки, неточности и неспособности, например, неспособность человека к быстрым и точным вычислениям.

Но ждем ли от искусственного интеллекта полной идентичности человеческим возможностям? Очевидно, нет. Но такая задача по построению полной аналогии машины и человека проходит лейтмотивом во множестве технологий искусственного интеллекта. Например, идея воспроизведения мозга дала старт разработке нейронных сетей. Желание «вырастить» идеальную саморазвивающуюся структуру породило эволюционные алгоритмы и игру «жизнь».

В то же время копирование внутренних механизмов, присущих человеку, и воспроизведение человеческих возможностей – вещи разные. Перцептрон, реакционный процессинг и другие механизмы помогают нам наилучшим образом реализовать определенные идеи и задачи, такие как распознавание образов. И напротив, полное повторение человека – бесперспективно, поскольку слепое копирование результата не несет ключевой ценности: человек лишь носитель, одна из реализаций интеллекта, к которому мы стремимся в программных системах.

Компьютер обладает собственными важными свойствами, которые делают его ценными. Это быстрота и точность расчетов, быстрота в поиске данных, надежность и точность (дискретность) логики в выполнении программных инструкций, возможность хранения большого числа хорошо структурированных данных.

Мышление человека существенно отличается от правил обработки данных компьютером. Он состоит в возможности принятия решений в условиях ограниченности информации, отсутствии четкой структурированности информации, динамическом построении операций. Да и, вообще говоря, можно ли сравнивать компьютер и человека? К компьютеру не могут быть применимы такие сложные понятия как чувственность, интуиция, мечты, фантазия, экспрессия и одухотворенность. Но от этого компьютер не делает хуже вычисления и выполнение программ. От этого мы не работаем с ним меньше. Так давайте рассматривать искусственный интеллект в компьютерной реализации не как конкурента или брата, а как средство.

Ставя задачу «искусственного интеллекта» перед компьютером, мы определяем те результаты, которые будут полезны для человека, поскольку на современной стадии развития программных технологий глупо стремиться воспринимать компьютер как брата по разуму. И, более того, это совершенно не нужно, поскольку компьютер остается средством для принятия решений, средством для хранения и быстрого поиска данных, средством для решения прикладных задач.

Периодически в области искусственного интеллекта мелькает еще одна крамольная идея. Звучит она примерно следующим образом: «Если воспроизвести человеческий мозг, то мы получим интеллект». К этому измышлению подстегивает Норберт Винер: «…кибернетика полагает, что строение машины или организма является показателем их способности выполнить задачу. … механическая ригидность насекомого ограничивает его интеллект, в то время как механическая гибкость человеческого существа обеспечивает его почти безграничное интеллектуальное развитие… Теоретически, если бы мы могли создать машину, механическая структура которой воспроизводила бы человеческую физиологию, то мы могли бы иметь машину, „интеллектуальные способности“ которой воспроизводили бы умственные способности людей.» [2]

Как будто интеллект – это результат химической реакции или суп, сваренный из точно выверенных ингредиентов по какому-то мифическому рецепту. К сожалению, немногого можно добиться способом слепого копирования и ожидания последующей синергии, а также перехода количества в качество. Нельзя надеяться на то, что созданная структура сама восполнит отсутствие представлений по целому пласту знаний и видения процесса, и сама начнет воспроизводство «разума». Чудес не будет. Одна из причин этого в том, что помимо воспроизведения механизмов, как минимум, эти механизмы необходимо наделить знаниями (врожденными или априорными рефлексами, инстинктами и пр.). Думаю, если рукотворно можно создать такой организм, этот организм будет схож с клоном самого человека. А нужен ли нам клон, к этому ли мы стремились, создавая такой механизм?

Позиционирование и правильное определение задач и подходов в каждой области – вот от чего зависит успех науки в отрасли знаний. До сих пор «искусственный интеллект» является чрезмерно размытой областью знаний. Сюда входит и распознавание образов, и интеллектуальные алгоритмы, к чему бы они ни прилагались, и рекурсивный поиск, и задачи распознавания языка.

В поиске методов, подходов и идей искусственного интеллекта, объектом исследований выступает человек: его форма мышления, его механизмы. Желание повторить человека – не самое лучшее желание, поскольку человек – это сложное, согласованное и гармонизированное создание. Каждая его составляющая является важной – слух, коммуникация, зрение, мышление, речь, органы движения. Ущемление одной небольшой функции может вести к потере полноценной деятельности человека. Потому идея создания подобия человека – это, пожалуй, профанация.

Другое дело, человек – прекрасное создание для анализа и воспроизведения его отдельных составляющих. В первую очередь, того, что отличает человека от другого рода организмов, а именно наличия разума.

Единый обобщающий тест на разум, который придумал Тьюринг, сомнителен как по своему назначению, так и по своей методике. При помощи этого теста Тьюринг хотел отойти от критерия «разумности», но что же измеряет этот тест? Он сличает человека и машину, или точнее – подтягивает машину к человеческим стандартам. Насколько такое сравнение корректно, и насколько такое сравнение полезно? Какой результат и какие выводы даст такое сравнение? Ясных ответов на эти вопросы попросту нет.

Кроме того, я не уверен, что все люди могут пройти тест Тьюринга. Эта неуверенность может посетить каждого, кто достаточно времени провел в чатах. Будь у человека на том

«конце провода» сложности с вербализацией (выражением мыслей), со знанием языка общения, перцепцией (восприятием) – и мы, тестируя его по Тьюрингу, то есть, сомневаясь в том, что на другой стороне «провода» человек, можем поставить отрицательный результат. Тест пройден не будет. Можно сделать вывод, что тест Тьюринга – это не тест на разумность, это система распознания «свой – чужой». То есть, тест на то, соответствует ли собеседник понятиям и стандартам тестирующего.

Единственным выводом из этой непростой ситуации является отказ от понятия «разума» как цели, к которой должен стремиться искусственный интеллект. Не надо измерять разум как единое и неделимое целое, так же как не надо стремиться определить критерии разума, поскольку это эмпирическое понятие, состоящее из сотен характеристик.

Что же надо делать в действительности, если мы всё же желаем использовать машину как интеллектуальный механизм? Очевидно, выявить составляющие разума, которые имеют ценность, как для человека, так и для их отчуждения в сторону машинной реализации.

Таким образом, с точки зрения агентов нам вообще не важно, кто находится на «том конце провода». Основная задача реализации агентов заключается в том, чтобы агент, находящийся с нами в контакте, был адекватен относительно исполняемых им задач и коммуникации. Но критерий адекватности означает не тест на соответствие полноте тестирующего человека (как тест Тьюринга), а тест на необходимость и достаточность в коммуникации. Это подобно тому, как мы можем объясниться с иностранцами, порой языком жестов и междометий, но объясняя суть вещей, несмотря на то, что он, очевидно, не знает таких слов как «шумовка» или «противень» на нашем языке общения. А, возможно, таких слов попросту нет в его родном языке.

Поскольку агенты могут существовать в гетерогенной среде, основное требование предъявляется к потенциальной возможности коммуникации.

Агенты-молчуны и вещатели

Многоагентная среда, прежде всего, предъявляет требования к коммуникативному уровню агентов. Однако часть агентов может никак не проявлять себя и заниматься исключительно сбором данных. Такой вид агентов является частным случаем. Например, это агенты, собирающие широковещательные сообщения – публичную информацию (сообщения, размещенные в публичных хранилищах), а также агенты-перехватчики частных сообщений. Примером агентов-молчунов являются индексирующие подсистемы поисковых систем.

Противоположностью «молчунам» являются «вещатели», которые не запрашивают, не принимают и не хранят внешнюю информацию, зато постоянно сообщают вовне некоторую информацию. Самым распространенным, элементарным и нужным агентом такого типа является агент-часы. Человек не может самостоятельно объективно и точно определить время, он может основываться только на своих ощущениях. Для получения данных о времени человек обращается к этому агенту. Компьютеры также синхронизируют собственное, не совсем точное время, с глобальными часами.

Структура агентов

Определение агента

Агент существует в определенной среде. По отношению к самому агенту, окружение, в котором он существует, является внешней средой.

Например, компьютерные агенты могут существовать в среде глобальной сети Интернет. И хотя мы можем на время отказаться от рассмотрения лишних и громоздких механизмов взаимодействия между агентами и сосредоточиться на внутренней структуре агента, в то же время мы не должны отказываться от его существования во внешней среде в целом при определении структуры агента.

Процессы взаимодействия с внешней средой не менее важны, чем процессы, протекающие внутри самой системы.

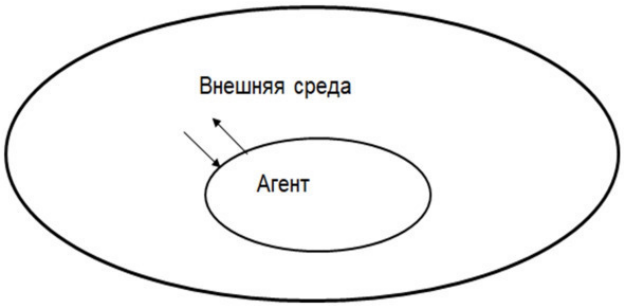

Рис.1 Взаимодействие агента с внешней средой

Несмотря на то, что в данном случае используется понятие «агент», в общем, любая программная система в той или иной степени может рассматриваться как агент. В то же время, большинство программных систем нацелены на взаимодействие с человеком либо со строго определенными информационными системами, поэтому они в чистом виде не являются агентами многоагентных систем. Если рассматривать такое взаимодействие сквозь призму агентных систем, то большинство программных систем можно определить как частный вид агента, реализующих коммуникацию через человеко-машинный интерфейс (HMI, Human machine interface). При этом построение жестко структурированных экранных и электронных интерфейсов характеризует системы как **жестко коммуницирующие**. Это результат того, что системы являются слабо адаптивными, имеющими статичную структуру данных, а их функциональность может изменяться только под воздействием ручных корректировок программистом.

Поскольку мы рассматриваем человеко-машинные системы, мы предполагаем взаимодействие машинных систем между собой, взаимодействие человека с машинными системами и, разумеется, взаимодействие людей между собой. Все они, таким образом, могут рассматриваться как единая коммуникативная среда. А, соответственно, мы можем рассматривать человека как агента. Разумеется, с точки зрения современных машинных систем, человек – это очень умный агент.

Агент отличается от жестко коммуницирующих систем тем, что агент самостоятельно может инициировать контакт с другим агентом либо, как частный случай, выбирать источник данных для получения данных. Например, хорошим примером агента являются индексирую-

щие системы поисковых интернет-систем. Они обращаются к серверам с данными на основе собственных предпочтений и алгоритмов индексации данных.

Большинство программ можно рассматривать как частный случай агента в человеко-машинной среде. Одновременно с этим существует большое количество программных систем, взаимодействующих между собой автономно. Большинство из них являются агентами, хотя их нельзя назвать интеллектуальными агентами. В частности, агентами в машинной среде являются любые программные продукты, которые взаимодействуют между собой, используя любой возможный интерфейс. Будь это FTP-протокол для обмена файлами по сети, HTTP для передачи запросов и данных через web-среду, SMTP для обмена почтовыми сообщениями или UDP для обмена различными данными на низком уровне.

Для обмена данными между программными системами существует масса протоколов обмена данных, средств упаковки данных и структур данных. Например, XML как язык разметки, позволяющий упаковать в себя произвольную структуру данных; SOAP как транспортный протокол передачи упакованных данных (сообщений).

Также сформировался специализированный класс программ, называемых middleware (ПО промежуточного уровня) как промежуточный уровень в коммуникации между различными программами. Программное обеспечение этого класса позволяет получать, отправлять, преобразовывать и буферизировать данные между различными программами-источниками и программами-получателями. Вместе с middleware существует и класс программ, обозначающих ETL (extract, transform, load – выделение, трансформация, загрузка), которые делают подготовку данных при их передаче для некоторой программной системы.

Агент – автономная сущность, обладающая памятью, совершающая процессы обработки данных и позволяющая обмениваться информацией через каналы коммуникации.

Автономность является одной из важнейших характеристик агента. Помимо прочего, эта характеристика связана с понятием собственности, в том числе материальной собственности, как например, наличие собственных носителей данных. Собственность агента может быть связана с материальной сущностью агента, то есть все, что заключено в аппаратном обеспечении агента рассматривается как собственность. Кроме того, собственность может быть связана с хранимыми данными, специфическими правилами обработки данных, специфическими правилами и средствами коммуникации. Таким примером являются авторские права и ноу-хау.

Необходимо сказать несколько слов о целостности агента. Агент – это система с определенным набором составляющих компонентов. Без части составляющих, например, носителей данных, агент не может существовать или рассматриваться как агент.

Примерами агентов могут служить компьютеры, отдельные приложения, люди, группы людей. Для каждого из этих агентов характерны единый коммуникационный канал, а для группы людей – единая позиция для внешних коммуникаций, единая структура хранения данных и система обработки данных.

Составные части агента

Переместимся от макро-объектов, где агент представлялся как один из участников общего взаимодействия с другими агентами, к рассмотрению агента как основного объекта исследований. Вопросы коммуникации, взаимодействия в общей среде будут затронуты позже.

Агент как объект является самостоятельной и самодостаточной структурой. Для обеспечения своей деятельности агент должен включать в себя ряд составляющих элементов, описываемых ниже.

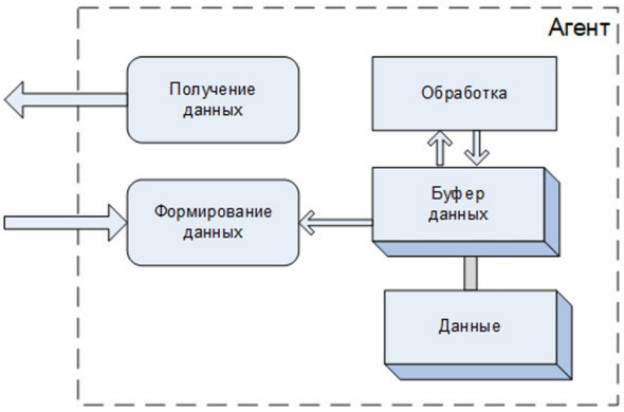

Рис.2 Составные части агента

Большинство программных продуктов включает в себя составляющую, связанную с структурированием и хранением данных, составляющую, связанную с обработкой данных, и интерфейсную часть с внешними системами или с пользователем.

Сравнивая структуру агента с аппаратными реализациями различных систем, можно сравнить его с компьютером. Так, в Intel x86-архитектуре присутствует процессор, который отвечает за выполнение команд и, соответственно, за обработку данных, память, хранящую данные и программный код. Архитектура разделяется на две подсистемы, которые обслуживаются двумя разными микросхемами окружения. Контроллер-концентратор памяти «Северный мост» (North Bridge) обеспечивает работу центрального процессора, оперативной памяти и видеоадаптера. Контроллер-концентратор ввода-вывода «Южный мост» (South Bridge) обеспечивает работу контроллеров ввода-вывода, интегрированных в материнскую плату, в том числе сети, внешние устройства и пр.

Центром агента является буфер данных. Он является аналогом быстрой памяти и используется для оперативной обработки выбранных данных из внутреннего хранилища данных, для последующего обновления хранилища данных и для целей коммуникации.

В качестве еще одной аналогии рассмотрим классическую трехзвенную архитектуру приложений: сервер баз данных – сервер приложений – клиент. Сервер баз данных хранит операционные данные, сервер приложений обрабатывает эти данные, клиент отображает данные. В сопоставлении с агентом, сервер баз данных – это хранилище данных, сервер приложений является блоком обработки, а вот клиент – это блок, связанный с коммуникацией, получением данных и формированием данных. Буфер данных агента существует на уровне сервера приложений.

Поскольку буфер данных является центром агента, рассмотрим его подробнее. А именно, почему он является центром, и зачем он нужен как таковой. Начнем с обработки данных. Данными невозможно оперировать непосредственно внутри хранилища данных (имеется ввиду

интерактивные, коммуникативные операции), поскольку давно прошли те времена, когда данные запись за записью анализировались программой в базе данных, а затем они извлекались для последующих операций. Сейчас рутинная работа по выборке данных перенесена на уровень сервера баз данных. Мы даем задание системе управления базами данных выбрать, обновить, удалить, добавить данные, а система выбирает их для нас в наш промежуточный буфер либо обновляет данные в хранилище данных. Другими словами, все фактические операции производятся в буфере данных – туда выбирается ограниченная часть данных, которые нужны для обработки. Там формируются изменения в данных, которые затем транслируются (переносятся) в хранилище данных.

Для каналов коммуникации буфер данных играет другую роль. Дело в том, что каналы коммуникации не должны напрямую обращаться к данным во внутреннем хранилище данных, так же как коммуникационные каналы не должны иметь прямого влияния на хранилище данных. Это так, поскольку одним из основных принципов построения агента является его автономность, а, следовательно, и независимость от внешних факторов. В дополнение к этому мы обсуждали гетерогенность структур агента (см. главу «Агенты как черный ящик»), где мы предполагаем, что реализация агента может быть различной. При этом язык коммуникации с другими агентами остается единым в среде взаимодействия агентов. Таким образом, при формировании данных для внешней среды и для коммуникации с другими агентами, мы должны транслировать данные из внутренней структуры агента в структуру языка для взаимодействия с другими агентами. Такая трансляция является результатом обработки. И поэтому только результаты обработки (а не данные из хранилища данных) должны отправляться в выходные каналы агента.

Аналогичная ситуация с поступающей извне информацией. С одной стороны, агент должен транслировать входящую информацию во внутреннюю структуру данных, а с другой стороны агент также должен сделать определенные предварительные обработки, такие как оценка входящей информации на непротиворечивость (валидация информации). Первый вид обработки входящей информации соответствует гармонизации с внутренней структурой данных по форме, а второй вид обработки – это гармонизация по содержанию.

Воспроизведение агентов, априорная информация

Если мы рассматриваем мультиагентную систему как среду или сообщество, то агенты в этой среде должны быть подобны организмам – возникать и исчезать из среды и присоединяться и уходить из сообществ. А потому агенты должны иметь возможность совершать действия, характеризующие живые организмы, в том числе развитие и размножение.

Интеллектуальное (не физическое) развитие агента целиком связано с процессом обучения через коммуникацию, то есть через наблюдение и обучение. Развитие агента связано с внутренними механизмами и возможностями агента.

Размножение, то есть воспроизведение себе подобных агентов, используется как основная идея программ-вирусов, центральная задача которых максимально быстро распространиться и прикрепиться к другим программам на различных компьютерах.

В мультиагентных системах могут существовать различные подходы к воспроизведению. Некоторые мультиагентные системы (например, web-среда) определяют формат протокола взаимодействия агентов, например, протокол HTTP. Создание новых видов агентов не контролируется, основное требование – соответствие требованиям протоколам обмена данными. То есть реализация агента может быть создана с нуля или может быть создана через модификацию существующей реализации агента, например, агента с открытым исходным кодом. И, разумеется, воспроизводство нового агента – это использование некоторой реализации агента.

Другие мультиагентные системы предъявляют требования к конкретной реализации агента (например, Skype и другие системы обмена сообщениями). Протокол обмена данными между агентами является недокументированным, закрытым. А создание нового агента – это инсталляция агента на некотором компьютере в определенной среде (сети Интернет).

После установки некоторой реализации агента как программного продукта вы можете использовать его с данными, которые развертываются при установке. Второй вариант – это копирование данных из существующего агента. Копирование совершенно невозможно для живых существ, но в том-то и отличие информационных систем, что мы понимаем их внутреннее наполнение, а потому можем контролировать его и управлять им.

В корпоративных системах управления (ERP-системах) практикуются подобные копирования данных и настроек. Например, если компания внедряет систему в своих нескольких филиалах (дочерних компаниях), то обычно выбирается один из филиалов для пилотного проекта внедрения, а затем внедренная система развертывается (roll out) на другие филиалы. Филиалы могут располагаться в других странах и, соответственно, предъявлять требования к системе в зависимости от государственного регулирования и местных обычаев в этих странах. Различные филиалы могут иметь собственные бизнес-процессы или применять различный объем бизнес-процессов, а, следовательно, и использовать различный функционал в системе. Тем не менее, основные бизнес-процессы остаются едиными во всей компании, они позволяют получать единообразную отчетность, как финансовую, так и по различным натуральным показателям деятельности компании.

Развертывание настроек делается следующим образом:

1) Выделяются общие настройки для всей компании и, соответственно, для всех филиалов, где внедряется система. Эти настройки образуют так называемый корпоративный шаблон системы.

2) Выделяются общие данные всех инсталляций и/или компаний системы. Например, основная часть плана счетов (детализация плана счетов может зависеть от особенностей бизнеса каждого филиала), единые коды элементов запасов, единые коды глобальных поставщиков и заказчиков и пр. Такие данные обычно контролируются и управляются централизо-

ванно, и они образуют специализированную базу данных, называемую нормативно-справочной информацией (НСИ, MDM – Master Data Management).

3) Из базы данных пилотного филиала вычищаются настройки и данные, специфичные для этого конкретного филиала. Затем эти настройки и данные развертываются через копирование в другие филиалы.

Аналогичным образом данные из одного агента могут переноситься в другие агенты. То есть платформа для обработки данных и структуры данных остаются теми же самыми, что и в исходной системе, данные же и способы обработки данных переносятся ограниченно. Эти переносимые данные и способы обработки данных далее будут описаны как априорные данные.

Остается открытым вопрос, что включают в себя априорные данные. Ответ на этот вопрос целиком зависит от возможностей системы, её гибкости и специфичности данных. Априорные данные не должны влиять на дальнейшие возможности системы к обучению, к взаимодействию с другими системами, то есть не должны формировать ненужных шаблонов поведения системы. В смысле переноса предопределенных данных принцип «чем больше информации, тем лучше» не способствует развитию обучающихся систем, поскольку у системы, которая уже имеет определенный набор данных, не возникает потребности в её восполнении. А пересмотр и замещение базовых (в данном случае априорных) данных – гораздо более затяжной процесс, чем выстраивание системы и структуры собственных данных с нуля, так как агенту не требуется сомневаться в актуальности данных, а он при потребности в них видит их отсутствие, и добывает их.

Анализ агентов

Как я упоминал выше, агенты в общей картине мира представляют собой чёрный ящик. Подобно принципу инкапсуляции в объектно-ориентированном программировании, все входящие данные должны быть обработаны внутренними механизмами агента и через них помещены во внутреннюю память, а выходящие – сформированы внутренними механизмами агента. То есть прямая запись и чтение в/из внутренней памяти недопустимы: извне не должно быть прямого доступа к хранилищу данных агента.

Тем не менее, структуры данных агента, процедуры обработки данных агента и накопленные данные агента представляют интерес для нас с точки зрения получения очищенных данных. Под очищенными данными имеется ввиду информация с отсеченной интерфейсной частью сообщений и отфильтрованная от различных конвенций и правил, связанных с общепринятыми правилами коммуникации, в том числе и языком коммуникации. То есть мы предполагаем, что внутренняя структура данных агента должна быть выстроена в соответствии с нашими требованиями, как архитекторов агента. Таким образом, мы сможем понимать, как организованы данные и, в частности, сможем получить ответ на вопросы «почему», «зачем», как и другие аналитические вопросы. Внутренняя логика агента должна быть в состоянии дать ответ на этот вопрос.

Противопоставлением логическим агентам, дающим такую возможность, выступают нейронные сети. Нейронные сети могут давать оптимальные решения, однако нейронные сети основаны на адаптивных механизмах обучения и, соответственно, они могут дать количественные характеристики выходящим данным, но не качественные. А, соответственно, мы не сможем узнать у них ответ на ключевой вопрос «почему?».

Далее рассмотрим реализацию агента. Будем подразумевать, что реализуется именно агент, однако будет рассматривать его с точки зрения интеллектуальной системы.

Вариативность платформ и аппаратных реализаций

Фактически мы считаем, что агент – это программная система, работающая на некоторой платформе. Платформа для агента является совокупностью аппаратной и программной платформ (операционная система). Аппаратная платформа реализуется некоторой архитектурой (например, x86 или SPARC). Программная платформа реализуется некоторой операционной системой и библиотеками, например, MS Windows,.NET, ADO.NET.

Платформа является некоторой оболочкой, которая может характеризовать собственность по отношению к агенту и изолированность агента по отношению к внешней среде.

С развитием технологий виртуализации аппаратная платформа стала превращаться в размытое понятие. Аппаратный сервер заменило понятие «виртуальный сервер» или «виртуальная машина». То есть агент перестает ассоциироваться с конкретной аппаратной системой, поскольку может лишь делить один и тот же физический сервер с множеством других агентов. Через современные технологии виртуализации множество различных аппаратных средств могут быть объединены в единую аппаратную платформу с точки зрения программного обеспечения. Хранилище данных может быть размещено в «облаке», то есть с физической точки зрения это хранилище никак не принадлежит агенту.

Однако при различных аппаратных платформах и платформенных реализациях вообще, логические реализации агентов (алгоритмы, реализованные в агентах) могут быть одинаковыми, в том числе структуры данных, механизмы обучения, правила обработки данных (процессинг) и пр. С точки зрения целостности система является совокупностью реализаций, которые определены проектировщиком системы. С точки зрения собственности, в этом случае к составным частям системы можно подходить аналогично финансовому лизингу (финансовой аренде) в управлении финансами: а именно, если вы берете в аренду некоторое оборудование, закупаемое специально для Вашей компании, или которое может быть использовано только Вашей компанией, оно должно быть учтено на балансе вашей компании.

При вариативности платформы для универсальной модели, нет смысла держаться некоторого эталона внутри платформы. Как мы определились ранее, агент представляет собой «черный ящик» и никого не должно волновать, что скрыто внутри него. Если мы и хотим оценить правильность построения этого агента, то это следует делать через коммуникацию с агентом, подобно тому, как это определяется тестом Тьюринга.

Построение информационной системы

Вариации архитектуры и требования к системе

Мы очень многое знаем про определенность, статичность, четкую последовательность действий и строгое определение структур данных в программных системах. И наоборот, не так много сказано про возможности вариаций – не про девиации, то есть отклонения от норм, а про наличие общей формы и вариации в реализации обработок и структур данных, которые формируют индивидуальность каждой из экземпляров программных систем. То есть, с одной стороны, мы декларируем основные принципы систем, с другой стороны, мы должны понимать, насколько гибкими являются эти системы. Гибкость достигается за счёт вариативности структур этих систем. Тремя столпами любой информационной системы являются данные, обработка и коммуникация. Коммуникацию сейчас оставим без рассмотрения как производную от обработки данных.

Архитектура и составные части интеллектуальной системы, разумеется, диктуются задачами, которые должны выполняться агентом, их масштабностью и способом реализации этих задач.

Вместе с тем, выбор языка программирования, платформы, серверов, сама реализация должна включать в себя определенный набор механизмов и методов. Их мы и рассмотрим.

Принцип универсализма системы означает, что интеллектуальная система не должна работать по заранее определенной программе. Её алгоритмы должны быть модифицируемыми и гибко изменяемыми. Процесс обучения самой системы, как необходимое условие для интеллектуального агента, должен включать в себя не столько изменение, усвоение операционных данных, а скорее формирование управляющих данных, с помощью которых агент сможет применять новые методики, новые практики, новые умения. То есть агент должен не просто воспринимать новые данные и распознавать новую информацию, но и стремиться лучше и более эффективно учиться в будущем.

Программа в современном понимании – это процедурный код либо объектно-ориентированный код, который инициируется событиями. Когда мы говорим про модификацию или изменение алгоритма, это означает, что код собирается из составляющих частей на основе поставленных целей. Соответственно, событийно-ориентированные программы если не заменяются, то дополняются программами, основанными на целях, которые к тому же собираются в зависимости от целей и налагаемых на их достижение условий.

Предъявление таких требований к возможностям интеллектуальной системы означает, что для его реализации должны применяться механизмы, которые до сих пор не реализованы в языках программирования, такие как модификация или воспроизведение программного кода или мета-программ.

Данное выше описание как нельзя лучше подходит под системы класса BPM (Business Process Management, управление бизнес-процессами) и технологии SOA (Service Oriented Architecture, сервисно-ориентированная архитектура), когда конечная система складывается из компонентов как из кубиков. Но здесь есть одно значительное «НО!», которое заключается в том, что компоненты в системах с такой архитектурой являются статичными, и они не обладают достаточной гибкостью, чтобы именно собирать их как кубики, а не переписывать и проектировать составляющие их компоненты каждый раз заново, как это и делается на практике.

Требование необходимости в реализации описываемого агента не ограничивается процедурами обработки данных – это требование относится также и к данным. Для каждой функциональной задачи требуется строить структуру данных, которая позволяет хранить исходные

данные, результаты обработки, статистику и производные данные для отчетности после получения из внешних источников и в процессе обработки.

В качестве примера: в системах управления производственным планированием структуры данных включают базовые данные, такие как центры работ (оборудование), производственные маршруты, списки материалов (спецификация готового изделия и полуфабрикатов). В систему вводятся планы по производству и выпуску готовой продукции. Система управления производством формирует план производства для каждого вида оборудования из плана выпуска готовой продукции. Затем по результатам выполнения работ в систему вводятся данные по фактически использованным материалам, времени выполнения работ и пр. После чего система формирует отчеты и накапливает статистические данные.

Для каждого вида информации во всех системах определяется структура данных, которая впоследствии наполняется единичными или массовыми данными.

При этом реализация требования универсальности для агента требует не только гибких правил обработки данных, но также и гибких структур данных, которые способны модифицировать и преобразовывать не только сами данные, но и структуры данных.

Например, в описанной выше производственной системе мы можем структурно разделить полуфабрикаты и готовую продукцию, если они имеют существенных различий в их свойствах.

Вариации структур данных и операций

Философия в качестве измерителя для понятий применяет оппозиционную модель. В ней свойство универсальности балансирует между фиксированной, статичной, неизменяемой структурой и максимально гибкими возможностями системы – универсумом. При этом, естественно, любая программная информационная система реализуется на основе некоторой структуры данных и программной составляющей, специфичной для её задач.

Например, система управления персоналом позволяет вам создавать новые поля в карточке сотрудника. Но при этом, система определяет, что должна существовать определенная таблица «карточка сотрудника», а каждый сотрудник должен быть идентифицирован табельным номером.

Таблица «карточка сотрудника», поля «табельный номер», «ФИО» и пр. – это и есть предопределенная структура данных. Возможность увеличения или уменьшения универсальности – добавление полей, удаление и изменение полей, реструктурирование таблицы и пр. и является реализацией универсальности системы, а её возможность к изменениям является свойством вариативности системы.

Вариативность касается как возможностей в обработке данных, так и возможностей в использовании структур данных. Рассмотрим эти две составляющие раздельно.

Означает ли максимальная гибкость в операциях, что система должна отойти от какой-либо структурированности программного кода? Нет, не означает. Это вопрос относится к понятию полноты вычислимости по Тьюрингу. Известно, что машина Тьюринга обладает полнотой вычислимости, то есть она может реализовать любой алгоритм. Следовательно, чтобы достичь достаточной вариативности по обрабатываемым алгоритмам, наш агент должен иметь возможность реализовать машину Тьюринга.

Формального определения критерия универсальности в структурах данных не существует. Однако основываясь на существующих практиках и теориях, можно вывести такой критерий. Основными принципами управления данными являются возможность хранения единичных и массовых данных, а также возможность хранения связанных данных. Производными требованиями к структурам данных является контроль целостности над связанными данными (контроль на разрыв связей и на согласованность данных), а также возможность поддержки сложных и упакованных (в некоторые структуры) данных.

Крайним примером статичности информации является выбитая в камне надпись. Физическая сущность носителя таких данных гарантирует его долговечность и неизменность. Максимальную гибкость в структуре данных мы стремимся получить через лёгкость изменения данных и через простоту в определении структур данных. Для надписей структурой является естественный язык, в информационных системах структура данных определяется платформой – языком программирования (типы данных), аппаратной структурой (например, размер машинного слова), реализацией языка программирования (различные виды языка Basic определяют различный размер для типа int), СУБД и пр.

Поскольку мы делаем упор на данные, мы должны обратить внимание на три важные составляющие, связанные с ними – хранение, извлечение и запись, обработка и обмен (коммуникация). Для передачи данных с их предварительной упаковкой используется языка разметки XML и JSON, который дает возможность «завернуть» всё, что угодно. Для хранения большого объема данных со сложной структурой обычно используется СУБД (системах управления базами данных) на основе реляционной модели данных либо, что реже, базы данных, основывающиеся на иных моделях данных. Для обработки данных используется несчетное множество программных технологий – языков, платформ, framework'ов, библиотек и т. п.

Предопределенная (априорная) информация агента

Любая программа предполагает определенный сценарий или несколько сценариев выполнения операций и действий. Например, в программах управления персоналом одним из сценариев является ввод данных о сотруднике, ввод данных о позициях штатного расписания, прием сотрудника на работу.

Если мы рассматриваем агент как некоторый универсум, как решение, которое может работать с различными структурами данных и реализовывать различные алгоритмы, то одним из важных вопросов является «с чего начать?».

Часто дети говорят «я умею всё». Под этим они подразумевают, что потенциально они могут научиться любому умению, впитать любое знание. Однако практически у них нет никаких жизненных навыков. Они действительно могут научиться, но еще пока ничему не научились. У них есть потенциал, но пока нет знаний.

С другой стороны, существует понятие априорных данных, когда система заведомо обладает определенным набором данных, а не только структурами данных. У людей обычно такие предопределенные знания называются генетической памятью.

Таким образом, интеллектуальная система также должна иметь некоторые предопределенные алгоритмы и данные. Прежде всего, такие предопределенные алгоритмы касаются обмена информацией с другими агентами. Кроме того, это встроенный интерфейс для получения данных, для вывода данных и для инициации выполнения заданий.

Парадокс Рассела для операций, автогенерация кода

Обсуждение автогенерации кода начнем с описания парадокса Рассела, хорошо известного в теории множеств: *пусть К – множество всех множеств, которые не содержат себя в качестве своего элемента. Содержит ли К само себя в качестве элемента?* Обычно этот парадокс иллюстрируется несколькими, насколько занимательными, настолько и абсурдными задачами:

Парадокс брадобрея: деревенскому брадобрею приказали брить всякого, кто не бреется сам и не брить того, кто бреется сам. Может ли брадобрей брить самого себя?

Парадокс мэра: в стране вышел указ о том, что мэры городов должны жить не в своем городе, а в специальном городе мэров. Может ли мэр города мэров жить в городе мэров?

Парадокс Рассела сам по себе неразрешим из-за самой постановки задачи, поскольку он ссылается на множество всех не включающих себя множеств, то есть в задачах противоречие задаётся аксиоматически, в самом определении. Чтобы свести математический взгляд на вещи к естественному, и если хотите, к бытовому понимаю, сравним этот парадокс с парадоксом всемогущества: попросите всемогущего сделать камень, который он не сможет поднять. Если получится, значит его всемогущество утратило силу, а если нет – то он и не был всемогущ.

Аналогичный парадокс, связанный с модификацией программного кода, возникает в задачах искусственного интеллекта. А именно, может ли программный код модифицировать и выстраивать сам себя? Или в другой интерпретации: может ли программный код строить другой программный код, не содержащий сам себя?

Как же соотносятся множества из парадокса Рассела и генерируемый программный код? На элементарном уровне код состоит из команд и операций. Их стройная последовательность и является кодом, который требуется получить. Но на уровне постановки задачи генерации никакого программного кода еще нет, это обобщенная информация или мета-информация о том, каким должен быть генерируемый код. Это и есть множество (задач), не содержащее другое множество (команд). Задача – это поручение процессору выполнить определенную команду. Получается, что, создавая программу, и записывая её в память мы даем поручение процессору выполнить определенный набор команд. Когда же процессор идет по этой последовательности команд, он рассматривает каждую из них в текущем контексте выполнения (регистры, состояние памяти и пр.). Понятие множества, не содержащего само себя – это фактически мета-информация (задачи) о мета-информации (командах), в которой не содержатся задачи более низкого уровня, то есть представления задач с более детализированным видом, хотя формально этот более абстрактный вид мета-информации остается мета-информацией. Подробнее представление мета-информации будет рассмотрено ниже.

Разумеется, парадокс Рассела не является краеугольным камнем для возможности генерации кода. В нашем случае, наоборот, мы понимаем, что одна программа может сформировать программный код (об этом ниже), и сама программа – это код. Так может ли идти речь о воспроизведении кода как об аналоге всемогущества?

Среди идей искусственного интеллекта существует идея самомодифицирующихся программ. На практике обычно программы не модифицируют себя сами. Есть ограниченный класс программ, которые изменяют свой код, такие как системы безопасности и их антагонисты – вирусы. С помощью механизма самомодификации программы обычно пытаются защититься от их распознавания. Идея самомодификации искусственного интеллекта состоит в самовоспроизведении кода, подобно рельсоукладчику на железной дороге, который кладет рельсы, а затем и едет по ним же.

В связи с тем, что самомодифицирующиеся программы, за исключением специфических, указанных выше программ, сейчас фактически не встречаются, поднятый мной вопрос

на основе парадокса Рассела не так уж наивен для сегодняшнего уровня развития информационных систем.

Существует вид программирования, основной задачей которого является генерация программного кода путем самомодификации либо путем создания новой программы. Такой вид программирования называется **мета-программированием**.

Мета-программирование, собственно, предлагает два варианта – это шаблоны и внеязыковые средства, такие как синтаксические и лексические анализаторы. Полиморфизм объекто-ориентированного подхода программирования также предлагается как один из инструментов мета-программирования. Частным случаем мета-программы является «квайн» (quine): программа, которая выдает на выходе точную копию своего исходного текста. Тем не менее, какие бы технологии программирования мы не перечисляли, созданием программ в мета-программировании занимаются люди, а подходы призваны лишь упрощать написание кода.

Что же формирует код или, говоря простым языком, заставляет программы исполняться? Крупным классом программ, порождающим программный код, являются компиляторы. Противоположностью генерации низкоуровневого исполняемого кода является исполнение кода высокого уровня при помощи программ-интерпретаторов, которые на ходу анализируют исходный код и исполняют инструкции программ. Существует и компромиссный вариант между компиляцией и интерпретацией программ. Это технологии, осуществляющие трансляцию исходного программного кода в байт-код или в p-код (пи-код). Байт-код является более низкоуровневым, p-код – в большей степени высокоуровневым. Внутри исполнимого модуля эти виды кода исполняются интерпретатором.

Существенным отличием интерпретаторов и p-кода от компилированного кода состоит в том, что, используя механизмы интерпретации, система «на ходу» может выполнять скрипты на том же языке программирования.

В качестве альтернативы внешние программные коды также могут встраиваться в систему. Например, Microsoft предлагает собственный SDK, основанный на Visual Basic for Applications (VBA), подобно тому, как VBA существует в офисных продуктах Microsoft – MS Word, MS Excel и пр.

Подводя итог, подчеркнем, что, учитывая разнообразие существующих методов исполнения кода, выбор той или иной программной архитектуры системы важен, поскольку интеллектуальная система должна уметь формировать и модифицировать порядок и правила выполнения задач.

Однако при автоматическом создании кода мы неизменно столкнемся с типичными проблемами, связанными с корректностью кода. Согласно статистики, в среднем 60% времени по разработке программного обеспечения тратится не на программирование, а на отладку и тестирование кода. Если человек сталкивается с подобными проблемами, то с ними столкнется и автоматическая система динамической генерации кода. То есть либо код должен быть лишен ошибок (что, очевидно, невозможно), либо ошибки при исполнении кода должны порождать исключения, которые в свою очередь должны возвращать исполнимый код на уровень регенерации кода. Коррекцию или регенерацию можно сравнить с проведением эксперимента, а последовательную отладку программы – с методом проб и ошибок.

Также как программист пишет код с точки зрения собственных представлений или теории, при отладке и тестировании этого кода и при возникновении логической ошибки или некорректных результатов программа с отладочной средой указывают, что теория программиста была не идеальной. Последовательными итерациями программист корректирует код, доводя его до совершенства.

Дилемма первичности для операций и данных, операции с операциями

В отношении информационных технологий мы можем поднять вопрос первичности: что возникло раньше – данные или операции? Эта дилемма возникает не из праздного интереса, это дилемма определения главенства и выстраивания зависимостей. Должны ли обработки подстраиваться под структуры данных либо структуры данных должны строиться, исходя из особенностей обработок, в том числе языка программирования, особенностей доступа к данным?

С точки зрения агентов, информация является первичной, поскольку в окружающем мире информация не является собственностью самого агента, а потому агент существует в среде, насыщенной информацией. Кроме того, сошлемся на понятие априорной информации, описанное выше: она должна существовать до того, как первый процесс начнет «работать».

С точки зрения современных информационных систем и, в частности интеллектуальных систем, внутреннее хранилище данных также является первичным, поскольку результаты и исходные данные обычно напрямую определяются в структуре данных системы. Особенности архитектуры, поддерживаемые алгоритмы обработки данных обычно не влияют на структуру данных, поскольку все современные системы могут работать через универсальные интерфейсы с самыми разными хранилищами данных. Другими словами, при создании информационных систем сначала проектируется структура данных, и затем алгоритмы, работающие с этими данными.

Однако, если методы обработки данных предопределены заранее (например, мы вынуждены пользоваться каким-либо внешним программным модулем), то мы вырабатываем и подстраиваем под программу собственные структуры данных. В том числе эти структуры должны отвечать условиям необходимости и полноты внешнего программного модуля. Формально мы можем трансформировать данные, но не можем влиять на полноту данных, так как мы вынуждены оперировать требуемым от нас набором данных, а процедура дополнения данных до некоторого требуемого уровня – это неординарная и не всегда решаемая задача.

Возвращаясь к вопросам, освещенным в предыдущей главе, мы можем рассмотреть программный код как данные. И в этом случае дилемма первичности приводит нас к задаче структурирования кода как данных и дальнейшему управлению кодом (хотя бы даже кодом различного уровня, не обязательно кодом низкого уровня – машинным кодом). И тогда дилемма первичности превращается в несколько иную дилемму: что первично – курица или курица из яйца? Не воспринимайте этот вопрос как игру слов с целью затуманить Ваш мозг, это лишь попытка привести рассуждение к рассмотрению кода как данных.

Для рассмотрения возможностей соединения двух полярных субстанций (данных и процессов) рассмотрим эту дилемму с точки зрения аппаратных архитектур.

Код в данном случае представляет собой управляющие данные. Данные, которые обрабатываются программным кодом, называются операционными данными. Часто возникает вопрос разделения управляющих и операционных данных. Поскольку управляющие данные, то есть программы, хранятся ровно так же, как и операционные данные, возникает вопрос, могут ли смешиваться эти виды данных.

Если проанализировать историю программных и аппаратных архитектур, программных продуктов и парадигм программирования, то можно найти различия в подходах реализации хранения данных обработки и программных кодов.

Идеи, основанные на автоматном программировании, явно разделяют инструкции и память. Самые известные примеры автоматного программирования – это машина Тьюринга и машина Поста.

Аналогичная ситуация складывается и с продукционным программированием (нормальные алгоритмы Маркова). Продукции описываются одной структурой, а операционные данные рассматриваются как другая структура. Функциональное программирование (лямбда-исчисление) также разделяет описательную и операционную часть.

Ещё в 1946 году фон Нейманом при проектировании первых процессоров были сформированы несколько принципов построения ЭВМ. Одним из его принципов был принцип однородности памяти, в котором говорилось, что программы и данные хранятся в одной и той же памяти. С командами можно выполнять такие же действия, как и с данными. Альтернативой фон Неймановской архитектуре является гарвардская архитектура, которую отличает раздельное хранение команд и данных.

В практическом применении процессоры в начальной стадии развития разделяли общую область памяти на отдельные сегменты. В одном сегменте хранились управляющие данные, то есть коды команд, другой сегмент был областью памяти для хранения операционных данных. С развитием архитектуры процессоров для защиты программы от некорректных операций и, следовательно, для поддержания целостности управляющих инструкций процессора (команд), сам процессор контролировал область памяти с программным кодом и не позволял записывать в нее какие-либо значения.

В высокоуровневых языках программирования транслятор обычно ограничивает доступ к памяти, где хранится исполняемый код, а в интерпретируемых языках и вовсе отсутствует возможность не то, что скорректировать текст программы, но и даже получить этот текст. Одним из исключений является инструкция List некоторых реализаций языка программирования Basic, выводящая на экран исходный код программы. В других языках программирования для этих целей специально придумывают специальные программы, называемые «**квайн**» (**quine**). Аналогично, в скриптовых языках типа bat-файлов в dos/windows и *sh в Unix-системах код существует в текстовых файлах, а соответственно и доступ к ним может осуществляться через операции с файлами. В части языков программирования такая возможность реализуется через возможности интерпретатора или p-кода (пи-кода).

Другим исключением является операции аналогичного вида со встраиваемыми языками и скриптами. Первые – это SDK VBA, которые позволяют вызывать интерпретируемые VBA-процедуры из программного кода и скрипты обычно относятся к собственным языкам программирования либо тому же интерпретируемому языку программирование (как, например, команда exec в T-SQL и eval () в PHP).

Но фактически эти возможности остаются невостребованными с точки зрения модификации кода.

Говоря о различии между программным кодом и данными, нельзя забывать, что и программный код содержит данные.

Строгое разделение данных и программной обработки данных влияет и на целостность самих данных. В частности, программный код содержит константы, которые ссылаются на данные в хранилище данных, в том числе в СУБД. Данные со временем могут меняться, в то время как ссылки на них остаются неизменными. Часто эти ссылки невозможно отследить, что является потенциальным источником ошибок как результат несогласованности между данными внутри программного кода и данными в хранилище данных, что называется нарушением логической целостности данных.

Например, мы настраиваем финансовый отчет на основании оборотов по продажам и затратам. В программном коде отчета мы жестко привязываемся к счету плана счетов «Затраты на командировки», чтобы по этому конкретному коду счета настроить отчет, собирающий финансовые проводки из главной книги. В результате, если в дальнейшем потребуется изменить план счетов, например, изменить код для счета «затраты на командировки» либо создать дополнительный счёт с новым кодом для этих же целей, то отчет не будет работать

так, как нужно – не будет работать вовсе либо будет работать частично по используемой ранее более узкой логике.

С одной стороны, вместо определения константы с кодом мы можем брать его из отдельного справочника или элемента данных, привязанного к плану счетов. Но с другой стороны, мы не можем это сделать, поскольку структура данных является фиксированной, а, кроме того, в таком случае нас ждёт сложная процедура администрирования этих структур, включая организацию ссылки на план счетов, массовых данных при необходимости (например, вместо одного счета мы должны указывать ряд счетов) и визуализации данных. Такой комплекс процедур влечет за собой существенные накладные расходы при настройке отчета, поэтому от этого пути отказываются в пользу отказа от контроля над целостностью данных.

В этой главе мы определили необходимые требования к реализации агента: это были требования сверху. То есть требования, которые помогут достичь критерия необходимости для достижения универсальности структур информации и данных.

Однако сам процесс структурирования и обработки информации невозможно начать, не обратившись к началам теории информации. Этому будут посвящены следующие главы.

Информатика структур

Определения

В предыдущих главах мы подходили к рассмотрению интеллектуальных систем сверху вниз. Теперь же настало время подойти к проектированию системы в обратном направлении, «снизу вверх», который обычно определяет ограничения и общепринятые стандарты к построению систем. Как и прежде, мы будем основываться на требовании универсальности при проектировании интеллектуальной информационной системы. И в этом случае нам понадобится рассмотреть базовые структуры данных и процессов, а также подходы к управлению данными и процессами, которые должны быть частью универсальной модели данных.

Информатика известна как наука о способах получения, накоплении, преобразовании и использовании информации. В упрощенном понимании эта наука рассматривает информацию как последовательность битов, а операции – как различные логические преобразования.

Нам необходимо определиться с основополагающими вопросами, связанными с определением информации. Это следует сделать, поскольку в большинстве случаев происходит смешение понятия информации и данных. Так что давайте начнем с определений.

Информация – это сведения о некоторых объектах и процессах, которые могут храниться и передаваться между агентами информации. Например, годовые кольца на спиле дерева отражают информацию о его возрасте. Сами кольца являются данными, их носителем и, следовательно, агентом – дерево. Мы расшифровали эти данные о возрасте – это наша информация, мы ее носители. Также информация может трактоваться не только как сведения, но и как знания о некоторых объектах и процессах. Информация располагается на некотором носителе и является неотделимой от носителя, то есть информация не существует без носителя, а при разрушении носителя, разрушается и информация.

При этом информация может отражать некоторые факты или быть самостоятельной. Например, в поле мы можем увидеть некоторый механизм неизвестного нам назначения. С помощью зрения мы «сняли» этот механизм, следовательно, информация о существующем объекте отразилась в нашей памяти. Теперь мы знаем, что существует такой объект определенной формы, но непонятного назначения. С помощью умозаключений мы можем интерпретировать наличие и назначение этого объекта. Например, то, что он стоит в поле, означает, что это объект сельскохозяйственного назначения. Такие выводы являются **интерпретацией**, которая позволяет восстановить смысл вещей, не определенный явным образом.

Информация не является лишь отражением некоторого физического факта, поскольку она также описывает умозрительные объекты. Например, понятие конкатенации как операции «склеивания» объектов данных линейной структуры не имеет отражения в материальном смысле. Эта операция может быть спроецирована на некоторые физические объекты в качестве примера, но в оригинальном своем значении она является умозрительной операцией.

Данные – это представление фактов и идей в формализованном виде, пригодном для передачи и обработки в некотором информационном процессе. Другими словами, данные – это формализованная или закодированная информация. При этом данные сами по себе являются информацией, поскольку они являются своего рода физическим объектом. В нашей реализации интеллектуальной системы мы стремимся к тому, чтобы закодировать любую информацию в данных, однако это не является самим собой разумеющимся процессом.

Данные являются некоторым кодом, который может храниться, транслироваться, копироваться и, в свою очередь, являться основой для хранения информации как, например, строковые данные. В таком случае информация и данные являются идентичными, при этом данные

являются информацией. Но не наоборот, поскольку информация не является формализованной, как данные.

Слово «данные» происходит от латинского «datum», буквально означающего «факт», однако в настоящее время «данные» не определяются как факт. Данные не трактуются как некоторая данность, определенные свершившиеся или заданная информация, их современная трактовка связана с определенностью – типизированностью, упорядоченностью, формализованностью.

В контексте этих двух определений мы можем понять, что предметом исследований информатики обычно являются именно данные, а не информация. То есть основным объектом изучения информатики является «очищенная», типизированная или подготовленная информация в виде данных. Однако постановка задач, их выработка и анализ производится именно на менее формализованном уровне – на уровне информации. В частности, далее мы рассмотрим семантическую модель данных, описывающую данные, но с помощью информации.

С учетом вышесказанного прямо сейчас мы находимся на очень неявном водоразделе, где перемешиваются информация и данные. С одной стороны, мы не можем поставить знак тождественности между информацией и данными, а с другой стороны, все технологии, связанные с данными, называются информационными технологиями. И ведь действительно, данные непосредственно связаны с информацией. Таким образом, пока мы не погрузились в пучины одного и другого, давайте на берегу договоримся о том, как разделять эти понятия.

Информация не является типизированной, не имеет ограничений по размерности. Всё, что касается типов (например, целочисленные значения), для информации является свойством, но не определяющим типом. Подробнее эти аспекты будут рассмотрены ниже. Данные, напротив, основываются на типах и конкретной размерности.

Процедура реализации информации в данных будем называть **кодированием**. А процедуру «декодирования» данных в информацию, то есть наделение данные смыслом (семантикой) будем называть **интерпретацией данных**.

Также существенным различием между информацией и данными является субъективность. Данные являются объективными, то есть мы воспринимаем их такими, какие они есть – все видят предметы в окружающем мире примерно одинаково, и зрение транслирует нам объективные данные. Информация же является субъективной в силу того, что каждый человек или группы людей со схожими взглядами, знаниями и понятиями интерпретируют и рассматривают один и тот же предмет по-разному. Например, маркшейдер рассматривает землю как породу, земледелец оценивает её плодородность, а строитель – плотность грунта для постройки дома.

Обратимся к вопросу об элементарном представлении информации.

Элементарная структура информации

В обычном, традиционном понимании элементарной единицей информации является бит. Однако это не единица информации, а единица данных! Давайте разберемся, в чем тут разница. Информация – это сведения или знания. Следовательно, информация представляется нам как некоторое смысловое выражение.

Например, «Лампочка горит» или «Лампочка не горит» будет элементарным значением состояния лампочки. Это состояние лампочки мы можем закодировать в единицу данных, а именно в бит. В то же время значение некоторого абстрактного бита говорит лишь о его собственном состоянии. Мы про него можем сказать, что «значение бита такой-то ячейки памяти – 0», это будет интерпретацией данных, но на абстрактном смысловом уровне. Если же мы знаем нагруженность этого бита состоянием лампочки, то посмотрев на значение бита, мы можем сделать вывод о состоянии лампочки.

Отношение информации к данным состоит в том, что информация обладает смыслом. Данные не связаны со смыслом, но могут являться основой для получения информации. То есть, информация может быть закодирована в данных. Также данные могут восстановить информацию, если они нагружены смыслом. В соответствии со смысловой нагрузкой данные могут быть интерпретированы. Кроме того, данные могут нести представление информации, когда они являются текстом некоторого сообщения, наделенного смыслом.

Например, предложение на русском языке несет смысловую нагрузку. Мы можем использовать Unicode или другую кодовую страницу для преобразования символов в языковые данные. В этом случае, используя преобразование данных в символы, мы получим информацию, закодированную в данных. Однако если мы применим другую кодовую страницу для расшифровки – например, ASCII к тексту в Unicode, мы не сможем понять содержимое данных, а, следовательно, не сможем извлечь смысл из кодов.

Ремарка: перекодированные данные в текст являются информацией только если закодированный текст имеет смысл.

Другим примером является реверс-инжиниринг программного кода, также известный как декомпиляция. При анализе чужого программного кода можно проследить логику, но невозможно понять, почему программа была написана именно таким образом, и чего хотел добиться автор определенными программными выражениями. Код, сгенерированный в языках программирования по технологии. Net корпорации Microsoft, является байт-кодом, а, соответственно, к нему может быть применен реверс-инжиниринг, который восстанавливает названия переменных, внутренних процедур, внутренних классов и пр. Всё это является хорошим материалом для понимания смысла программы, поскольку придает семантическую (смысловую) ясность. Разработчики, всячески стремящиеся противодействовать подобного рода анализу, придумали специальные программы **обфускаторы**, которые заменяют эти названия на другие, не несущие никакого смысла, а также которые противодействуют реверс-инжинирингу кода, поскольку добавляют в идентификаторы – названия переменных, методов и пр. запрещенные символы с точки зрения языков программирования (спецсимволы, пробелы и пр). Люди, всё же желающие анализировать код придумали программы **деобфускаторы**, которые позволяют сделать корректным код после декомпиляции при его реверс-инжиниринге. Разумеется, оригинальные названия переменных и других элементов кода деобфускаторы вернуть не в состоянии. Получается, что обфускаторы уничтожают смысл кода программы через искажение названий идентификаторов (переменных, названий процедур, классов и пр.), и, следовательно, восстановить смысл можно только из анализа операций с определенным методом, свойством, классом, переменной и пр.

Таким образом, само понятие битов и других структур данных, насколько являются необходимыми для хранения информации, настолько же они являются техническими структурами.

Если с единицами данных всё более-менее понятно, в таком случае что является единицей информации? Каким образом мы можем выделить смысл, и каким образом мы можем определить его элементарную составляющую?

Мышление человека основывается на образах. Образы тесно связаны с понятиями. И в целом, понимание образов происходит от чувств человека как источников восприятия мира. Например, лимон ассоциируется с желтым цветом, продолговатой формой фрукта и его кислым вкусом, который мы можем тут же представить.

Образ является единицей информации, но он слабо определяем. По большому счёту образ – это собирательный элемент информации, поскольку образы связаны с различными ощущениями от различных органов чувств, и, кроме того, часть представлений человека вообще не связывается с **чувственными** образами, например, различные абстрактные понятия, такие как, понятие «данные» или «наука». Такие понятия назовем **аналитическими**. Кроме того, образ нельзя назвать элементарным элементом информации в представлении человека, поскольку образ – это обычно целая сложная структура объектов в визуальном представлении, букет составляющих в обонятельном представлении, композиция звуков в аудиальном представлении и т. д. Образы часто сопоставляются с некоторым словом, то есть одно слово описывает сложный образ в виде набора объектов. Например, при слове «карта» Вы можете себе представить топографию какой-либо местности.

Основываясь на представлениях о логике мышления человека, а также о логических операциях, можно сделать следующее заключение. Возьмем какое-либо слово в смысле понятия или выражения. Выбранное понятие или выражение нагружено связями с другими словами, свойствами, понятиями и представлениями. Если последовательно применять к выбранному слову операцию абстрагирования, то наше понятие будет терять связи со свойствами или с другими понятиями. В конце концов, оно будет представлять собой только одно название, даже без какого-либо определения в форме словесного выражения. Например, представим лимон. Последовательно отбрасываем от понятия лимон его свойства – цвет, структуру, вкус, физическое представление, запах и т. п. Это можно делать через последовательные превращения лимона в более абстрактные понятия – «цитрусовые», фрукты, плоды, растительные продукты и так далее. В результате мы получаем понятие «нечто».

Объединяя понятия – от самого абстрактного и до самых сложных, имеющих сотни связей, можно дать им единое название – «**знак**».

Знак может быть абстрактной сущностью без каких-либо связей, и также он может иметь произвольное количество связей, в том числе репрезентации в виде чувственных образов. Таким образом, знак, с одной стороны, является элементарной единицей информации и, с другой стороны, может отражать любое сложное понятие.

Для простоты понимания знак можно воспринимать как какой-либо объект или понятие.

При этом знак является всеобъемлющей и единой структурой информации за счет возможности его связывания с другими знаками. Это очень важная особенность знака как структуры, поскольку с использованием такого представления знака в данных пропадают ограничения на структуру данных – на глубину свойств и связей. Отсутствие ограничений – очень важная характеристика, которая также имеет влияние на объединение данных и метаданных в единую структуру.

Одним из главных свойств знака является **репрезентация**. В психологии репрезентация определяется как уподобление или образ, дающий впечатление об оригинале. Репрезентации формируются у человека через различные репрезентативные системы: визуальную, аудиальную, кинестетическую и т. д. Образ, который мы обсуждали выше, является свойством знака как один из вариантов репрезентаций – мыслительной имитации или ассоциации с предметом.

Давайте будем реалистами, и не будем углубляться в такие сложные и неоднозначные задачи, как определение, каким образом следует хранить представление запаха или вкуса, поскольку они потребуют решения задач по идентификации запаха и вкуса. Выше я сделал оговорку, что мы отойдём от этих тем, поскольку они не являются основным объектом исследования.

Вместе с простыми, и со сложными визуальными, аудиальными и другими образами мы воспринимаем, распознаем и выражаем понятия через естественный язык. А это значит, что и текст на естественном языке является репрезентацией, поскольку через раскрытие существа понятий на естественном языке, будь то английский, русский или какой-либо другой язык, мы можем воспринять, проанализировать, воспроизвести, понять и транслировать информацию о некотором объекте.

Естественный язык служит универсальным средством представления информации. Тем не менее, язык не всегда является наилучшим средством выражения информации. Во-первых, естественный язык в силу своей универсальности в некоторых случаях менее эффективен, чем специализированные средства. Этот факт хорошо известен в различных областях деятельности, например, в музыке, где существует своя нотная грамота. Во-вторых, естественный язык ориентирован на вольный стиль общения и не обеспечивает необходимой точности регистрации и передачи информации. Но следуя требованиям универсальности, мы придерживаемся естественного языка как наилучшего средства выражения образов.

Знак не может быть изолированным объектом, он должен существовать в некоторой среде. Будучи связанным с другими знаками, назовём её **информационной средой**. Сюда входит язык, контекст, специфика употребления знака и пр.

Ранее я подробно описал агентов, чтобы показать, что информация в основном не является уделом закрытой архитектуры. В силу своего основного свойства она должна распространяться. Но при этом информация может преобразовываться, видоизменяться внутри и вне агентов. Поэтому информация может различаться внутри и вне агентов.

Информация может находиться внутри агента – это та информация, которой обладает агент. Назовем её **собственной информацией**.

Когда агент получает информацию извне, он должен принять и согласовать её в соответствии со своими внутренними образами, знаниями и представлениями. Этот процесс называется **восприятием** или **перцепцией**.

Если агент, напротив, хочет передать собственную информацию во вне, процесс формирования сообщений называется **вербализацией**. Вербализация и восприятие являются базовыми коммуникационными процессами. Далее в основном мы будем рассматривать собственную информацию агента.

Вербализация и восприятие образов

Уже имеющиеся у агента репрезентации могут быть им вербализованы. **Вербализация** – это словесное описание переживаний, чувств, мыслей, поведения. Вербализации всегда происходит в форме словесного описания. Описание через звуки, жесты и пр. относится к невербальной коммуникации – неформальной, нетипизированной, сложнее поддающейся формализации, несмотря на то, что роль невербальной коммуникации у людей очень высока, через нее передается и почерпывается до 40—50% информации.

Например, съев кусочек лука, вы можете сказать, что это лук, что он горький, что он щиплет слизистую оболочку. Таким образом, вы идентифицировали лук на вкус и вербализовали свои чувства. С другой стороны, существует огромное количество понятий, которые могут быть однозначно вербализованы человеком, но непонятно, каковы могут быть их исходные (чувственные) репрезентации у человека, поскольку они различаются в зависимости от мироощущения, способа запоминания знаний. Это нематериальные понятия, такие как описание характеров людей, понятие бесконечности, непрерывности и т. д.

Вербализация чувств и образов человеком является неоднозначным и неформализуемым процессом, зависящим от конкретного индивидуума, от его мировосприятия, от темперамента и множества других личностных факторов. В то же время, вербализация – это процесс, присущий именно человеку, ведь только человек использует естественный язык. При этом представление знаний внутри самого человека, если можно так выразиться, является скрытой структурой для самого человека. И сама вербализация чувств не всегда подвластна человеку, однако это развиваемое качество – четко и тонко выражать свои чувства.

Человек воспринимает образы, помнит их, обращается к ним как к первоисточникам собственных выводов, мыслит образами. Однако образ – это композитное, сборное понятие, поскольку образы являются результатом отображения реальности органами чувств в представлении и восприятии человека. Человек по-разному помнит визуальные и аудиальные образы. При этом чувственное восприятие является очень важной, хотя и слабоформализуемой частью жизни человека. В первую очередь человек получает опыт через органы чувств, через чувственные переживания и впечатления. Человек делает проверку и оценку предметов прежде всего через органы чувств: нам так важен «мягкий» пластик в автомобилях, при выборе автомобилей же так важен цвет кузова и салона. Для проверки, жесткий ли предмет, мы нажимаем на него и прислушиваемся к своим тактильным ощущениям.

С одной стороны, чувственные ощущения дают нам элементарную информацию о свойствах предметов, а с другой стороны, они дают нам то, что в обозримом будущем не получат компьютеры – а именно эстетическое восприятие чувственной информации: картины, музыкальные произведения, высокая кухня, парфюм.

И всё-таки образы являются «**сырой информацией**» (**raw information**): образы идентифицируются человеком, однако для полноценной аналитической деятельности они должны быть не только идентифицированы, но и интерпретированы. До этого процесса вся мыслительная деятельность человека будет похожа на мыслительную деятельность животных, которая в большей своей части основывается на рефлексах и инстинктах. С помощью образов не могут строиться обобщения, делаться выводы. И всё это из-за того, что образов недостаточно для аналитической деятельности, поскольку большую часть в человеческом мышлении занимают **понятия**. Понятия не имеют чувственной репрезентации – образа, поскольку являются аналитическими или искусственными понятиями, например, «интеллект», «обычай», «время», «математика» и т. п.

Задача идентификации образов в форме распознавания образов как один из этапов **восприятия образов** – это отдельная и достаточно сложная тема для информационных систем.

И в этой книге я не буду уделять достаточно внимания этой специализированной теме. Мы отделим эту задачу от задач управления знаниями, а интеллектуальная система может оперировать уже подготовленными репрезентациями на уровне естественных языков, на уровне общепринятых понятий.

Через процесс интерпретации информации образы могут быть сопоставлены с внутренними представлениями и выражены словами на естественном языке. При этом внутренние представления будут являться знаками, а слова – репрезентацией знаков.

Помимо хорошего средства выражения понятий, естественный язык является универсальным средством коммуникации между людьми (имеется ввиду вербальная коммуникация), и поэтому образы, превращенные в языковые конструкции – в слова, фразы, предложения – являются общепринятыми в обществе, а, следовательно, в разумной степени очищенными от субъективного восприятия и смысловой нагруженности агентом. Однако самое ценное от использования вербализованных образов в виде языковых конструкций – это возможность использования самого языка как такового, поскольку он:

– хорошо кодируется в данных,

– хорошо (или, как минимум, в достаточной степени) структурирован,

– является основой для коммуникации людей,

– является полным с точки зрения выражения смысла. То есть любое явление может быть выражено в языке с достаточной точностью.

Вербализации образов в форме языка, в свою очередь, сами могут рассматриваться как репрезентации. Поэтому в дальнейшем будем рассматривать репрезентацию именно как репрезентацию в естественном языке. В других случаях буду делать оговорку.

В философии мы можем встретить идеи, связанные со значением языка – в частности, понятие **логоса** как закона бытия, введенного Гераклитом в греческой философии, **онтологии** как раздела философии о принципах бытия, его формах, структурах и закономерностях.

Помимо сказанного выше, не могу не упомянуть, что язык лежит в основе христианства. Первая строка книги Нового завета в Евангелии от Иоанна – это «В начале было Слово, и Слово было у Бога, и Слово было Бог» (In the beginning was the Word, and the Word was with God, and the Word was God). Это указывает нам на то, что лишь смысловая составляющая, выраженная в естественном языке, является основой всего сущего, и веры, как его части. Слово в христианстве есть отражение Логоса. Слово Бога обретает плоть и становится «сыном» Бога, воплотившимся в Боге-Сыне Христосе: «И Слово стало плотию, и обитало с нами, полное благодати и истины; и мы видели славу Его, славу, как Единородного от Отца» (Евангелие от Иоанна, 1.14).

Организация информации

Название этой главы указанно именно как «организация информации», а не структурирование информации, поскольку структура подразумевает достаточно жесткие отношения между элементами. Организация же определяет принципы построения взаимоотношения между элементами информации.

Другими словами, структура обычно связана с отношениями элементов в некотором объекте. Но мы говорим про саму модель взаимодействия. Например, мы можем определить структуру данных в реляционной модели данных: сущность «класс» связана с сущностью «ученик» связью 1:n (один ко многим, то есть в одном классе учится много учеников). Но в данном случае мы хотим обсудить именно принципы выстраивания структур, то есть саму модель организации информации.

Принципы, по которым организуется информация, обычно определяются через информационную модель. Поскольку с точки зрения терминологии информация и данные внутри агента очень близки, а информация кодируется в данных, модели данных являются информационными моделями, хотя информационные модели напрямую и не предназначены для явного описания структур и взаимоотношений между структурами данных. Заметьте, что до сих пор мы не определяли правил, по которым информация должна быть спроецирована в данные.

Одна из задач, которые мы поставили себе для организации информации, состояла в том, что знак как элементарная частица информации должна иметь возможность быть изолированной, но также и иметь возможность включать в себя другие знаки как свойства.

Также организация информации и, следовательно, модель информации зависит от операций, которые должны проводиться с элементами информации.

Свойства знаков

Как мы рассмотрели выше, с помощью естественных языков можно выразить подавляющее большинство информации, которой владеет человек. Исключением могут являться собственные чувства, музыка и пр. – они могут относиться к классу образов, которые «нельзя выразить словами» (хотя обобщенно с помощью языка можно описать и это). Получается, знаки в своем взаимодействии должны иметь возможность представить всё, что описывает язык в том или ином виде, с той или иной подробностью.

В деталях к знаку можно предъявить следующие требования:

– Знак должен иметь возможность выразить и представлять собой как индивидуальное значение, так и комплексную, сложную структуру информации. В частности, знак должен включать в себя свойства знака.

– Знак должен уметь описывать предметы, действия и операции.

Конец ознакомительного фрагмента.