Распределённые системы обработки информации

Крищенко В.А.

Конспект лекций. Черновик от 11 января 2012 г..

Аннотация

Конспект лекций для по курсу «Распределенные системы обработки информации». К сожалению, ещё не претендует на полный охват прочитанных лекций.

Для студентов пятого курса кафедры «Программное обеспечение ЭВМ и информационные технологии» МГТУ им. Н.Э. Баумана.

Содержание

1 Предмет курса

Прежде всего нужно определить, какое ПО является предметом данного курса.

1.1 Определение распределённой системы обработки информации

Существует несколько подходов к определению распределённой системы обработки информации (далее — РСОИ), начиная от самого общего: любая система, охватывающая несколько взаимодействующих компьютеров. Предметом данного курса является РСОИ, понимаемая как система взаимодействующих независимых автоматизированных информационных систем (АИС). Такие АИС мы будем называть далее участниками данной РСОИ. Независимость здесь понимается следующим образом: каждая из систем-участников принадлежит и администрируется различными организациям, которые преследует собственные цели (напомним, что типичная цель коммерческой организации — увеличение собственной прибыли).

При данном определении РСОИ во главу угла ставиться невозможность для одной системы получить полностью достоверную информацию о состоянии другой системы на заданный момент времени и невозможность контролировать полностью её поведение.

Любая АИС, автоматизирующая предметную область («бизнес-процессы») одной организации, не рассматривается как предмет курса, даже если логика её работы разделена на множество компьютеров (конечно, такая система может быть участником РСОИ). Также курс не рассматривает высокопроизводительные вычисления (у нас есть отдельный курс для этого), высокомасштабируемые и высоконагруженные системы, проблемы создания системы высокой доступности. Не рассматриваются распределенные БД (map-reduce), для этого есть курс БД.

Примеры распределенных систем, удовлетворяющих данному определению, вполне многочисленны. Автоматизация любого информационного процесса, охватывающий несколько независимых организаций (например, туроператор, гостиница, авиакомпания и консульство) приводит в итоге к созданию подобной системы в случае реализации автоматизированного обмена информации между ними. Если такой процесс описать в виде схемы по стандарту IDEF0, то блок A0 на ней будет соответствовать процессу, реализуемому РСОИ в целом. Участники РСОИ будут выступать на такой схеме в качестве механизмов. В типичной для нашего курса РСОИ каждый из процессов A1-AN поддерживается одним или двумя такими механизмами, а каждый механизм поддерживает один-три процесса из A1-AN.

Логично предположить, что в общем случае для создания РСОИ всем организациями, владеющих непосредственно взаимодействующими участниками РСОИ, надо заключить друг с другом договор.

1.2 Следствия из определения РСОИ

Приведём основные следствия из данного нами определения РСОИ. Из этих следствий мы затем попытаемся вывести общие требования к участнику РСОИ.

1) Несмотря на участие в едином процессе, задействованные в РСОИ информационные системы часто бывают созданы для достижения не обязательно совпадающих целей (например, прибыль одной из организации не обязательно влечёт прибыль её партнеров).

2) Каждая информационная система имеет свою логику работы, причём с точки зрения других участников эта логика полностью не просматривается, система выглядит как некоторый «чёрный ящик».

3) Обмен информации между участниками распределённой системы почти наверняка использует публичные каналы доступа и недоверенную инфраструктуру. Хотя использование, например, курьеров так же возможно, но мы будем в целом ориентироваться на использование публичных сетей с TCP/IP.

4) Каждый участник РСОИ имеет свою базу данных или несколько баз данных, в которых хранит состояние своей модели предметной области. Прямой доступ на уровне SQL к этим БД другим участникам не даётся, даже на чтение с ограниченными правами (зачем другим организациям слишком много знать?).

5) Невозможно получить достоверное состояние другой системы, поскольку невозможно отдать независимой системе приказ типа «прекрати работу, зафиксируй своё состояние и сообщи его мне; я скажу, когда продолжить». Даже если один участник сообщит другому некоторую достоверную информацию о части своего состояния на некоторый момент времени, за время передачи этой информации состояние уже изменится.

6) Системы могут выдавать упрощенную или искажённую информацию другим участникам РСОИ. Примером первого случая может являться переход к качественным величинам («этого товара много»), примером второго — выдача вместо прямого отказа ложной информации об отсутствии ресурса («мест нет! а то вы больно часто снимаете бронь»).

1.3 Некоторая терминология

Локальной системой будет далее называться система, к которой относится текущее фраза, а удалённой[1] системой — ту систему, с которой взаимодействует локальная система.

Термин «локальная», таким образом, вводится просто как противоположность термину «удалённый», и может распространятся на разные понятия. Локальной базой данных, например, будет называть база данных локальной системы (а не база данных на localhost).

1.4 Требования к участнику РСОИ

Из данного определения РСОИ и его следствий можно вывести некоторый список не зависящих от предметной области список качеств, которыми должны обладать участники любой РСОИ. Следует отметить, что все используемые термины определены в рамках данного курса, и чтобы не претендовать на общность, все они взяты в кавычки.

«Открытость». Каждый участник РСОИ должен быть открытым: его сетевой протокол, включая форматы сообщений и последовательность обмена ими, должен быть явно специфицирован. В качестве нижестоящего протокола протоколу участника РСОИ лучше всего использовать стандартные платформо-независимые протоколы, работающие на прикладного уровне стека TCP/IP или поверх него.

Это требование проистекает из того факта, что разные фирмы и организации могут иметь свои взгляды на то, что является удобной платформой для разработки их ПО. Хорошим контр-примером является использование любой бинарной сериализации структур данных, зависящей от платформы разработки ПО или даже от последовательности байтов в памяти.

С другой стороны, при приёме сообщений от других участников нельзя быть полностью уверенным, что из сообщения полностью соответствуют известной спецификации (возможно, это наше локальное ПО уже устарело). Таким образом, желательно использование «щадящего» или даже адаптивного анализатора чужих сообщений.

«Надежность». Со спецификой РСОИ связано следующее требование надёжности: функционирование одного участника не должно нарушиться, если другие участники или линии связи не работают так, как ожидалось. Соответственно, взаимодействие двух участников не должно нарушаться из-за неработоспособности третьего участника. Под «нарушением работы»

[1] От *remote*, а не *removed*.

здесь следует понимать необработанную аварийную реакцию, потерю логической целостности данных, вечное ожидание и т. д.

Это требование следует из того, что не стоит ожидать выполнения каких-либо предположений о работе других участников системы (включая предположения, что участник включён и функционирует).

Требование надёжности касается так же реакции системы на возможные проблемы с каналами связи между участниками: их выход из строя в неподходящий момент не должен приводить к неисправимым нарушениям в целостности данных (например, ресурс навсегда остаётся зарезервированным за другим участников, а единственное подтверждение о резерве теряется при передаче).

«Масштабируемость». В данном случае понимается масштабируемость всей структуры РСОИ в целом: участник РСОИ должен быть способен работать с несколькими одинаковыми системами и с разными системами близкого назначения. Добавление взаимодействия с ещё одной системой должно вызывать минимально возможные изменения в ПО. Например, добавление взаимодействие с ещё одной аналогичной системой должно затрагивать только конфигурацию ПО.

«Экономичность»[1]. Каждый участник должен достаточно разумно тратить ценные ограниченные ресурсы ЭВМ. Например, создание по потоку ОС на каждую обрабатываемую заявку для многих предметных областей будет глубоко неверным решением с точки зрения масштабируемости. Другим примером анти-экономичности будут транзакции в локальной базе данных, которые будут завершены лишь при получении некоторого сообщения от удалённой системы (данный пример также нарушает требование «надежности»).

«Эффективность». В понимании данного курса эффективность охватывает и распределение некомпьютерных ресурсов: нельзя выполнять требования других систем, которые идут вразрез с требованиями достижения цели организации. Например, оптовый дистрибьютор вряд ли забронирует на неделю большее количество высоколиквидного товара для малоизученного покупателя.

Это требование проистекает из того очевидного факта, что управляющие системами-участниками организации не должны работать во вред себе.

«Безопасность». Требования к безопасности в данном курсе следуют из того, что у каждого участника нет полного доверия не только к каналам связи, но и к партнёрам. В силу этого под безопасностью в данном курсе понимается как шифрование передаваемых сообщений, а также идентификация и авторизация систем, с которыми взаимодействует каждый участник РСОИ, так и ограничение выдаваемой информации о своём состоянии.

Из определения РСОИ следует очевидное требования авторизации: каждый из участников не должен сообщать больше необходимого о себе и желательно вообще не сообщать слишком многого. Поэтому в ответе на запрос о количестве какого-то ресурса вместо конкретных цифр могут встречаться понятия «нет», «будет», «мало», «вам хватит».

«Синхронизация». Синхронизация в классическом виде, когда в какой-то момент времени достоверное известно, что две сущности имеют одинаковое представление о чём-либо, видимо, невозможна при данном в курсе определении РСОИ.

Тем не менее, пытаться синхронизировать хотя бы часть представления о мире между участниками РСОИ нужно для работы системы, хотя, как было указано ранее, в общем случае состояние удалённой системы нельзя зафиксировать на некоторый промежуток времени. Поэтому в данном курсе каждый участник пытается согласовывать часть своих представления о мире с другими системами, обычно ведя постоянный повторяющийся обмен информации с ними.

[1] Ранее эта проблема была вторым аспектом масштабируемости.

2 Участник РСОИ

Определим протокол участника РСОИ и дадим общие предложения по его программной структуре.

2.1 Протокол участника РСОИ

Протокол обычно рассматривается как следующая пятёрка:

1) назначение протокола: предоставляемые им возможности;
2) используемый нижестоящий протокол или протоколы (например, HTTP);
3) алфавит (например, печатные символы из UTF-8);
4) словарь сообщений (например, сообщения «забронировать» и «отменить») и синтаксис сообщений;
5) возможная последовательность сообщений и их семантика;

Далее мы и будем называть используемые нашим протоколом участника РСОИ нижестоящие протоколы просто «нижестоящими».

Каждая система, участвующая в распределённой системе, имеет (в худшем случае) свой словарь внешних сообщений с собственным синтаксисом каждого сообщения, а также, возможно, свой уникальный порядок обмена сообщениями. Можно надеется, что для систем одного типа порядок часто будет совпадать, но и это необязательно. Нижестоящий протокол, используемый для передачи сообщений, так же может различаться от системе к системе. Таким образом, каждый участник имеет свой уникальный интерфейс, доступный для других систем, задаваемый один или несколькими протоколами.

В силу данного в курсе определения РСОИ нельзя гарантировать, что одна система имеет полные и корректные представления об интерфейсе другой системы. Это не касается нижестоящего протокола и словаря символов: без чёткого представления о нём вряд ли удастся передать хоть что-то. Уже формат сообщений может чуть измениться или расшириться без уведомления всех партнёров, или партнёры не успеют изменить свою систему вовремя. Предположения о возможных задержках или обязательности ответа системы на запрос, вероятно, могут нарушаться достаточно часто.

Таким образом, учитывая требования надежности, система не должна полагаться на то, что её предположения об интерфейсе других систем являются полностью верными, и это должно учитываться как при разборе полученных от других систем сообщений, так и при проектировании протокола обмена сообщениями.

2.2 Сообщения и заявки

Для дальнейшего описания определим полуформально понятия *сообщения* и *заявки*.

Сообщением нижестоящего протокола называется минимальная передаваемая им единица полезной информации, например, письмо в случае протокола SMTP.

Сеансом нижестоящего протокола называется минимальная логически завершённая операция обмена данными, передающая сообщение (возможно, передаюбщая и принимающая сообщение), например, посылка одного письма в случае протоколов SMTP или POP3, посылка запроса и получение ответа в случае протокола HTTP.

Сообщением протокола участника РСОИ (или просто сообщением) будем называть наименьшую единицу обмена информацией определенным синтаксисом и семантикой, например, «отмена заказа» или «резервирование номера в гостинице».

В рамках одного одного сеанса обмена используемого нижестоящего сетевого протокола передаётся одно или несколько (обычно с случае буферизации) сообщений протокола участника РСОИ. В теории возможен так же случай мультиплексирования передаваемой информа-

ции, когда в рамках одного сеанса нижестоящего протокола передаётся только часть сообщения вышестоящего, а в дальнейшем оно «склеивается» из данных, переданных в нескольких сеансах. В данном курсе мы будем считать последний неудачным, поскольку по сути нам придётся тогда реализовывать «свой TCP», и будем считать хорошим вариантом только связь «один-ко-многим» между сообщениями нижестоящего и вышестоящего протокола (и «один-к-одному», как частный случай)

Для удобства иногда можно считать, что ответ от транспортной подсистемы вида «не удалось доставить» и событие о завершении интервала ожидания ответа («тайм-аут») также являются сообщениями-ответами на отправляемое сообщение.

Заявкой для определенности и по отдалённой аналогии с системами массового обслуживания будем называть некоторую единицу обслуживание, например, заказ номера гостиницы. Сообщение «забронировать номер» в случае успешной обработки приводит к созданию заявки, а сообщение «снять бронь» – к завершении её жизненного цикла.

В данном курсе любое изменение состояния модели предметной области удалённой системы производится через заявку.

Сообщения, таким образом, можно разделить на две следующие группы:

— сообщения, меняющее состояния заявки, в том числе создающую новую заявку. Эти сообщения меняют состояние системы;
— сообщения, не меняющее состояния какой-либо заявки — сообщения о получении (части) состояния системы.

Сообщения из второй группы всегда являются идемпотенными.

2.3 Предложения по структуре участника

Из указанных требований можно составить предложения по части структуры автоматизированной информационной системы, входящей в распределённую систему. В ней можно выделить три части, связанных именно с её участием в обмене информацией в рамках распределённой системы:

— подсистема обмена сообщениями, использующая нижестоящие протоколы;
— подсистема обработки заявок, ответственная за их жизненный цикл;
— подсистема фильтрации сообщений, находящаяся между подсистемами обмена и обработки заявок.

Разумеется, в участвующей в РСОИ АИС также присутствует основная логика работы, которую мы будем условно называть «логикой локальной модели предметной области» или «бизнес-логикой». Вместе со своими базами данных она образует локальную модель предметной области системы участника.

В системе также могут присутствовать интерфейсы для различных видов пользователей (ИП), а так же подсистема сбора и хранения статистики, которая хранить, помимо прочего, историю заявок и сообщений, полученных от других систем. Статистика в дальнейшем используется подсистемой фильтрации.

Отметим, что с точки зрения всей РС все эти подсистемы составляют единое целое. Поэтому отделение интерфейса от логики, например, ни в коем случае не создаёт РС в понимании данного курса, хотя и крайне желательно по известным причинам, к которым добавляется ещё одна: логика может использоваться протоколом обмена при обработке внешних сообщений, при этом никакого локального пользователя у системы может и не существовать.

Основная логика работы может декомпозироваться на ряд подсистем и быть довольно сложной и разделённой на множество компьютеров. С точки зрения данного курса, это не столь важно, поскольку он сосредоточен на вопросах обмена информацией между АИС, а не на во-

просах разделения логики на компоненты, разносимые по разным компьютерам и не на вопросах интеграции различных приложений одного предприятия в единую АИС.

Опишем теперь кратко основные функции подсистем, которые должны присутствовать в АИС, участвующей в распределённой системе.

Подсистема обмена сообщениями занимается вопросами преобразования полученных от подсистемы обработки заявок сообщений в форму, понятную другим системам. Таким образом, она отвечает за лексику и синтаксис сообщений, исходя из определения протокола участника РСОИ. Эта же подсистема занимается также надежной (при необходимости) и безопасной отправкой и приёмом сообщений, часто опираясь на отвечающие этим требованиям нижестоящие протоколы. Надёжной здесь считается такая доставка, которая гарантирует доставку при заданных условиях («удалённая система включается как минимум на два часа в течение суток»). Безопасная доставка предполагает защиту от просмотра и изменения передаваемой информации третьми лицами, а также возможность идентификации получателем отправителя сообщения. Эта же система занимается буферизацией сообщений, их шифрованием и цифровой подписью, а так же идентификацией систем-партнёров.

Подсистема фильтрации сообщений фильтрует сообщения во внутреннем представлении, чтобы не загружать систему работой, которая выглядит неправдоподобной. Можно также сказать, что эта система реализует некоторый механизм авторизации («если покупатель всегда брал по тонне в месяц, то заказ на 100 тонн выглядит ошибкой его менеджера»). Она выносит «вердикт» по заявкам, полученных от других систем, исходя из которого протокол обмена предпринимает те или иные дальнейшие шаги. Подсистема фильтрации в качестве типичного случая предлагает исполнить или отклонить заявку или обратится за подсказкой к человеку. Вердикт может иметь и иной вид, например «подождать час и начать выполнять, если это будет возможно». Вердикт основывается на:

— текущем состоянии модели предметной области, получемой от бизнес-логики;
— истории взаимоотношений с партнёром, полученной из подсистемы статистики;
— параметрах заявки (например, на количестве запрашиваемых ресурсов);
— самих правилах принятия решения.

Подсистема обработки заявок является ключевой с точки зрения участия в работе распределённой системы. Она реализует основную часть высокоуровневого протокола участника РСОИ, связанную с жизненным циклом выполняемой заявки. Чтобы абстрагировать эту подсистему от конкретного вида сообщений, передаваемых «по кабелю», логично передавать ей (и подсистему фильтрации) сообщения от подсистемы обмена в некоторым удобном внутреннем виде. Разумным решением будет организация очереди таких сообщений в постоянном хранилище (например, в БД), возможно, с приоритетами. Подсистема обработки заявок будет последовательно извлекать и обрабатывать внутренние сообщения из этой очереди.

Аналогичная очередь может быть использована и для передачи внутренних сообщений к подсистеме обмена сообщениями. Преобразованный во внутреннее представление сообщения передаются к и принимаются от подсистемы обмена сообщениями. Источником внутренних сообщений, кроме неё и подсистемы обработки заявок, может быть также основная бизнес-логика.

По мере необходимости подсистема протокола обращается к подсистеме принятия решений для определения дальнейших действий, а также к бизнес-логике для запроса и изменения в состояния модели системы, и к системы статистик для ведения статистики по полученным от других систем сообщениям.

Отметим, что процессы регулярного получения информации о наличии ресурсах в удалённых системах могут быть не связаны с обработкой заявок. Тем не менее на данном этапе развития курса логично считать такой процесс также частью системы обработки заявок (в дальнейшем видимо стоит выделить его отдельно).

3 Обмен сообщениями

Опишем подробнее задачи и функции подсистемы обмена сообщениями.

3.1 Задачи подсистемы обмена сообщениями

Можно выделить следующие задачи подсистемы обмена сообщениями:

— преобразованием информации между внутреннем представлением и видом, используемом в протоколе обмена информацией между двумя системами;
— обеспечением безопасности в части шифрования сообщений и обеспечения их целостности и идентификации систем, включая подпись и проверку подписи сообщения.
— собственно обменом сообщениями между системами и обнаружением ошибок обмена;
— реализацией надежного асинхронного обмена (обычно это требуется, но возлагается на низ), реализация надежного синхронного обмена (но обычно это не требуется);
— буферизацией при обмене информацией.

Можно выделить три модели обмена информации:

1) опрос-ответ, когда инициатором выступает желающий получить информацию;
2) уведомление, когда обладающей информацией сообщает желающим (не обязательно надёжным способом — уведомления всё равно устаревают);
3) доставку сообщений (надёжную, в отличие от уведомлений), инициатором выступает желающий что-то сделать.

Обмен во всех трёх случаях происходит в рамках одного сеанса нижестоящего протокола («опрос-ответ» здесь рассматривается таким образом как синхронный).

Модель опроса и выглядит более разумно в РС (по сравнению с уведомлениями), поскольку не предполагает, что источник информации тратит свои ресурсы как на хранение списка подписчиков, так и на попытки соединиться с недоступным подписчиком. В курсе в основном подразумевается использование модели опроса, когда речь идёт о синхронном обмене и получении состояния.

С другой стороны, уведомление в ряде случаев выглядит более логично, если вспомнить о целях участниках РСОИ («давно вы что-то не интересовались нашими остатками и прайсом»), поскольку потребитель информации может являться и заказчиком с точки зрения бизнеса. Использование модели уведомления осложняется прежде всего тем, что желающий получить информацию может быть недоступен в момент её рассылки. На практике модель уведомления так же часто достаточно используется, в этом случае вместо регулярного, повторяемого в случае ошибок опроса будет либо регулярное, повторяемое до устаревания информации уведомление, либо использование нижестоящего протокола, берущего на себя эту функцию (например, рассылкой по SMTP).

3.2 Преобразование сообщений между внешним и внутренним представлением

При реализации РС встречаются три проблемы, связанных с внешними интерфейсами различных, но близких по назначению систем.

1) У них почти наверняка имеются сообщения с одинаковой семантикой, но разным синтаксисом.
2) У них есть сообщения с одинаковой семантикой, но с разными синтаксисом и и разными походами к их передаче. Например, у одной системы подтверждение заявки приходит как ответ по почте, другую надо опрашивать по SOAP.
3) У них близкие по смыслу сообщения имеют несколько разную семантику, включая разные предположениями о последовательности сообщений. Например, одна система требует по-

вторного подтверждение бронирования ресурса. Различия могут быть вызваны и различными бизнес-процессами (например, продажа по предоплате или в рассрочку).

Подсистема обмена сообщениями должна решать первую проблему, может решить (как минимум частично) вторую и не может решить третью — последняя задача требует изменения вышестоящего протокола.

Для решения первой задачи внутреннее представление сообщения, соответствующего сообщениям с одинаковой семантикой, неформально, должно быть объединением форматов этих сообщений для всех систем одинакового назначения. Подсистема обмена сообщениями должна решать задачу преобразования входящих сообщений во внутреннее представление и исходящих сообщений во внешнее.

3.3 Запросы получения и изменения состояния

Для запросов изменения состояния и ответов на них (если последние используются) крайне желательно использовать надёжный нижестоящий протокол.

Для уменьшения нагрузки на разработчика и использования готовых проверенных решений вместо самодельных велосипедов следует использовать готовое решение, в лице которого по описанным в технологическом разделе причинам может выступать почтовая система. Можно создать собственную почтовую инфраструктуру так и использовать существующую (вот почему сообщения обязательно нужно шифровать, кстати). Следует отметить, что подсистема обмена сообщениями всё равно будет представлять собой нечто большее, чем прямую отправку по SMTP, который по терминологии данного курса выступает в роли нижестоящего протокола.

Для получение состояния системы (в том числе состояния нашей заявки) в качестве нижестоящего протокола могут использоваться и простые синхронные средства обмена, не использующие какой-либо промежуточной инфраструктуры типа почтовой системы, и рассчитанных на прямое соединение двух сторон. Возможные технологии описаны в соответствующем разделе.

3.4 Шифрование и подпись сообщений

(О том, что кругом враги и «шпионы посредине» — писать лень. См. плакаты «Someone talked» или советские).

Может показаться, что необходимость в шифровании и электронной подписи сообщений зависит от того, используется ли безопасный транспортный протокол (например, HTTPS или HTTP поверх VPN) или небезопасный (например, HTTP поверх публичной сети). Безопасным транспортом может быть сам нижестоящий протокол или один из протоколов под ним.

Однако с точки выявленных требований к РС можно сформулировать следующее положение: каждая система должна, в идеале, использовать ассиметричное шифрование и электронную подпись своих сообщений, поскольку этим достигается максимальная протяжённость безопасного пути для сообщения, вместе с безопасным нижестоящим протоколом.

Таким образом безопасность на уровне сообщений не исключает требование использования безопасного транспортного протокола. Например, при использовании SMTP или HTTP мы можем использовать авторизацию по паролю (хотя курс склоняется к идентификации по сертификату открытого ключа, но конспект пока старый), что требует использования SSL для безопасной передачи пароля. Возможно, что мы не хотим показывать третьим лицам и адрес электронной почты получателя и другую служебную информацию[1].

Таким образом, в РС лучше всего использовать как безопасный протокол, защищённый от прослушки и проблемы «шпиона посредине» (читай — HTTPS или SMTP+SSL), так и шифро-

[1] Тем не менее мы должны помнить, что не бывает *security via obscurity*.

вание на уровне сообщений, которые затем будут передаваться по безопасному нижестоящему протоколу.

Отметим, что наш нижестоящий протокол может как совпадать с HTTPS (HTTPS/REST), так и работать поверх него (XML-RPC, SOAP).

3.5 Буферизация

Под буферизацией здесь понимается группировка нескольких сообщений протокола обмена сообщениями, передаваемых к одной и той же удалённой системе, в одно сообщение нижестоящего протокола. Для использования буферизации удалённая система должна быть способна принять несколько сообщений протокола РСОИ в одном сообщении нижестоящего протокола, будь то запрос HTTP или письмо, передаваемое по SMTP. Кроме того, иногда мы моем передать несколько нижестоящих сообщений в рамках одного сеанса нижестоящего протокола, что тоже даёт эффект (см. SMTP).

На практике буферизация реализуется следующим образом: обнаружив в очереди сообщение, которое необходимо отправить, подсистема обмена сообщениям не бросается сразу выполнять это указание а ждёт некоторое время (зависящее от системы и характера нагрузки, порядка секунд), и если в течении этого времени появляются другие сообщения для этой же удалённой системы, то они отсылаются вместе. Разумно считать время буферизации связанным со временем отклика на уровне IP или даже временем обмена с удалённой системой (если успешные пинги ходят за 200мс, то эта величина и является возможным временем буферизации).

Буферизация позволяет снизить накладные расходы и повысить производительность системы, особенно при частом периодическом опросе состояния удалённой системы (*«как там наши двадцать заявок?»*). Неверное использование буферизации (слишком большое время задержки при малом числе сообщений для одной удалённой системы) может привезти к падению производительности.

Буферизация может работать и в обратном направлении: вместо того, чтобы сразу отправлять сообщение (во внутренней форме) в очередь обработки подсистемой обработки заявок, можно использовать буферизацию в качестве возможного метода борьбы с противоречивыми сообщениями (недостатком этого подхода является до, что для этого при буферизации нужно понимать семантику сообщений).

4 Обработка заявок

Опишем подробнее задачи и функции подсистемы обработки заявок.

4.1 Особенности протокола РС

В основе работы распределённой системы лежит протокол взаимодействия между участниками. Его ключевая часть —семантика сообщений — реализуется, по нашему предложению, подсистемой обработки заявок.

Протокол участника РСОИ имеет следующие особенности и принципиальные отличия от стандартных сетевых протоколов.

— Протокол работы всей РСОИ в целом охватывает системы нескольких видов. Каждый вид может быть представлен множеством систем с различным интерфейсом.

— Реализация протокола в каждой из систем создаётся, в общем случае, независимыми друг от друга разработчиками и может различаться в силу разных причин (особенно касаясь предположений по нюансам семантики и корректной последовательности сообщений). При проектировании протокола не следует полагаться на полностью корректную работу других систем,

поэтому отслеживание всех тайм-аутов является обязательным, а обработка двусмысленных сообщений — желательной.

— Успешная доставка сообщения удалённой системе не означает, что это сообщение принято (или будет принято) к исполнению удалённой системой.

Задачей подсистемы обработки заявок является реализация семантики протокола, решающего с учётом выявленных факторов стоящие перед системой задачи.

Задачи, стоящие перед фирмой, владеющей АИС, могут быть отчасти противоречивы, например, шанс увеличить текущую прибыль моет противоречить цели создания хорошей репутации. Будем считать, что у фирмы существует некоторая политика отношений с партнёрами[1], которая позволяет определить как правила подсистемы фильтрации, так и логику обработки заявок.

Одна из составных частей описания протокола — возможности, предоставляемые им. В данном случае речь идёт о протоколе самого верхнего уровня, используемом в рамках какой-либо автоматизированной системы и для каждой из систем, участвующей в РС, эти функции могут быть уникальными. Общей частью в их описании, могут быть следующие моменты:

— описание целей, стоящих перед АИС, для достижения которых создан этот протокол;
— описание систем, с которыми этот протокол взаимодействует;
— описание временных интервалов, за которые протокол выполняет свою функцию или информирует систему о неудаче;
— описание интерфейса с бизнес-логикой (включая вызовы бизнес-логики, инициируемых протоколом).

4.2 Жизненный цикл и параметры заявки

Жизненный цикл начинается по сообщению от внешней или от локальной же системы. Основное состояние заявки принадлежит некоторому конечному множеству («ожидает бронирования», «забронирована», «выдана»). Жизненный цикл имеет и некоторое конечное состояние, обычно как минимум два — для успешного и безуспешного завершения. После попадания в конечное состояние заявка уже не меняет свое состояние, а находится в некотором архиве завершённых заявок[2].

Кроме основного состояния, заявка обычно характеризуется множеством параметров, которые можно разбить на две группы. К первой относятся полученные извне параметры заявки, появляющимися в момент её создания из параметров сообщения, и, может быть, меняемые в дальнейшем (например, число бронируемых мест в кинотеатре). Эту группу мы будем (условно) называть параметрами запроса.

Ко второй группе относятся добавленные в ходе выполнения заявки самим протоколом параметры, связанные с процессом обработки заявки (например, число повторов попыток бронирования, список опрошенных систем и так далее). Это множество может меняться в процессе жизни заявки. Эту группу мы будем (условно) называть параметрами обработки заявки.

Очевидно, что создание протокола, допускающего произвольное изменение параметров запроса извне может быть довольно сложным, особенно если речь идёт о запросе дополнительных ресурсов или частичном уменьшении требуемых ресурсов (представим, что в ходе обработки заявки мы закупаем недостающий товар у третьего лица). Поэтому в реальных системах мы часто видим ситуацию, когда единственным допустимым изменением со стороны является

[1] Для нас не важно, выражается она в договорах, в виде внутренго документа или существует как неформальное знание.

[2] Как мы помним из курса БД, в реальной жизни никто ничего не удаляет, если не считать случая сокрытия следов.

запрос на отмену заявки, а для добавление ресурсов нужно послать запрос на создание новой заявки.

4.3 Представление обработки заявки в виде автомата

Естественным представлением жизненного цикла заявки является детерминированный конечный автомат с выходом. Формально было бы верно считать множеством состояний этого автомата подмножество декартова произведения множества состояний заявки и всех множеств значений параметров обработки заявки. Однако практическая полезность у такого автомата может быть только при использовании формальных методов его исследования, для программиста же весьма сомнительна полезность автомата, где каждая из N попыток повтора опроса M идентичных (с точки зрения уровня протокола) систем представлена отдельным состоянием.

В силу этого, при проектировании используется сокращённый вид автомата, где явно отображаются только основные состояния заявки, в то время как состояние автомата определяется, формально, как подмножество декартово произведения состояния и параметров.

Функция перехода будет отображать состояние заявки, параметры заявки и сообщение (со всеми его параметрами) в новое состояние заявки и в новые параметры заявки (например, число неудавшихся попыток увеличилось на единицу). «Выходом» автомата является:

— посылка сообщений другим системам;
— запуск таймеров, которые пришлют автомату сообщение о таймауте.
— обращение к бизнес-логике (а также через неё к пользователю, например, за подтверждением действия).

Состояния автомата соответствуют моментам времени, в котором он ожидает получения сообщений (от внешних систем, подтверждения от человека и т.д.).

В соответствии с данным определением РС, автомат должен отвечать как минимум двум требованиям.

— При получении любого сообщения, на появление которого не удалось придумать сколь-нибудь разумную реакцию в данном состоянии, автомат остаётся в данном состоянии (и добавляет запись в журнал для дальнейшего «разбора полётов»), а не сообщает о недопустимости входной цепочки. Эту реакцию по умолчанию нет нужды показывать при конструировании автомата.
— Автомат должен попадать в одно из конечных состояний за конечный промежуток времени при любой входной цепочке внешних событий. Таким образом, при каждой смене состояния должен запускаться (или продолжать работать) таймер, который пришлёт сообщение о тайм-ауте, которое должно быть обработано автоматом.

Отметим, что успешная отправка сообщения в РС не означает, что сообщение принято к исполнению, поэтому нет оснований утверждать, что мы когда-нибудь получим ответ. Поэтому, в силу требования надежности, использование таймеров необходимо всегда.

Отдельным моментом в реализации автомата является опрос M однородных систем, где число M, конечно же, меняется. С этой точки зрения внешние события в случае параллельного опроса, кода необходимо получить $R \in [1, M]$ ответов, могут быть сформулированы примерно как:

— получен очередной ответ, число ответов меньше R;
— получено R ответов;
— тайм-аут.

4.4 Обработка двусмысленных сообщений и ответов

В РС может существовать ситуация, когда от одна система может присылать, например, в ответ на заявку о занятии ресурса несколько противоречивых ответов. Например, сообщение об успешном занятия ресурса было связано в попытками найти его у третьих фирм, но сделать этого в дальнейшем не удалось, или подсистема принятия решений отвергла заказ, но потом его вручную подтвердил менеджер продаж или наоборот.

В силу характера РС, такая внутренняя несогласованность (или просто политика «лучше обмануть клиента, чем упустить заказ») одной АИС, к сожалению, становится хорошо заметной для её партнёров. Это порождает задачу обработки несогласованных ответов.

Простейший подход к обработке двусмысленных сообщений и ответов состоит в их буферизации, что можно сделать как на уровне транспорта, так и на уровне протокола (и не ясно, что лучше; возможно, комбинированный вариант). В этом случае никаких действий по обработке сообщения не предпринимается до истечения некоторого интервала времени («периода недоверия»), после чего во внимание принимается последнее из противоречивых сообщений. Очевидным недостатком является снижение производительности в случае отсутствия двусмысленности и увеличение времени отклика, достоинством — простота реализации.

Хотя такой подход выглядит неэффективным, в условиях, когда двусмысленные ответы достаточно часты, он может быть разумен. Если обработка сообщения вызывает изменения в третьих системах, то этот подход выглядит практически единственным правильным. Если же изменения в состоянии третьих систем не входят в реакцию на сообщение, то можно реализовать протокол, отменяющий изменения в состоянии локальной системы при приходе нового ответа.

«Решение» этой проблемы по приниципу «сказанного не воротишь» не представляется разумным.

4.5 Запросы получения состояния

Основной целью автомата является обработка запросов изменения состояния данной системы и опрос внешних систем. Для обработки запросов получения состояния конечный автомат часто может иметь вырожденный вид и иметь единственное состояние, в котором следует:

— вызвать подсистему принятия решений (возможно, от нас хотят слишком много информации?);

— опросить бизнес-логику и получить от неё состояние системы;

— в случае успеха передать полученные сведения в качестве ответа, возможно, в упрощённом виде (*«данного ресурса осталось мало»*).

Таким образом, на практике при обработке запросов получения собственного состояния работа подсистемы протокола сводится к одной функции и не имеет состояний. Забегая вперёд, отметим, что такой запрос можно реализовать, например, на основе запроса-ответа HTTP в пределах одного сеанса TCP/IP (используя XML-RPC или HTTP/REST и т. д.).

Однако, запрос получения состояния в худшем случае может охватывать и получение состояния от удалённых систем. Поскольку нет гарантий, что эти системы работают в настоящий момент, то для реализации такого запроса необходим полноценный конечный автомат с тайм-аутами, и реализовать этот запрос в пределах одной сессии TCP/IP в принципе невозможно. Для решения этой проблемы можно использовать и кеширование информации о состоянии удалённых систем, что позволит избежать чрезмерно частого опроса их состояний и «разведёт» процесс опроса состояния удалённых систем и обработку запросов к локальной системе.

4.6 Идентификаторы заявок и сообщений

Заявка должна иметь идентификаторы, позволяющие:

— идентифицировать заявку в удалённой системе при получении её состояния (*«как там мой заказ №1234?»*);
— идентифицировать заявку внутри локальной системы.

Это может быть либо один и тот же идентификатор, сообщаемый системе-инициатору заявки до (предпочтительнее) или после (менее предпочтительно, ответ не должен потеряться!) создания заявки. Это могут быть и два разных идентификатора: один создаётся в системе, обрабатывающей заявку, а другой состоит из пары из идентификатора системы-инициатора и идентификатора заявки в системе-инициаторе.

Нежелателен вариант, когда обрабатывающая заявку система полностью полагается на идентификатор, который был в запросе на создание заявки (а вдруг системе-инициаторе сломался автоинкремент или ещё что?).

Наличие в пришедшем сообщение на создание заявки идентификатора заявки, назначенного удалённой системой, крайне полезно, поскольку позволяет отбросить дубликаты таких сообщений. Аналогично и наличие у всех сообщений уникальных (в пределах системы-отправителя) идентификаторов сообщений позволяет отбросить дублирующиеся сообщения.

4.7 Реализация автомата

Поскольку временные интервалы ожидания ответов в автомате могут измеряться часами, то даже для слабонагруженных систем реализация автомата в виде отдельного потока с хранением его состояния в ОЗУ выглядит неверной идей. Для высоконагруженных систем такое решение просто катастрофическое, особенно если речь идёт о потоках ОС, а не о «виртуальных» потоках исполняющей среды.

Состояние автомата в данном курсе следует хранить в БД, а вести действия по обработке заявки (преобразовывать вход автомата в выход) в идеале следует в несколько потоков. Число потоков разумно выбирать в зависимости от вычислительных мощностей, а не от числа активных заявок (см. требование экономичности). Обработка заявок в единственном потоке также возможно, как заведомо потока-безопасная, в случае небольшого потока сообщений.

Разумная реализация состоит в создании нескольких рабочих потоков, каждый из которых обрабатывает события примерно следующим образом:

— извлекает событие из очереди (например, внутреннее сообщение от подсистемы обмена);
— определяет, к какому автомату относится событие;
— считывает его состояние из постоянного хранилища (БД);
— обрабатывает событие;
— сохраняет в БД новое состояние автомата и удаляет событие из очереди событий (которая, видимо, так же должна содержаться в БД).

Хотя это и не снимает проблемы выключения системы при обработке события, но по крайней мере даёт основания считать, что выключение компьютера при сотне недообработанных заявок не приводят к их полной потере, что неплохо. Работа с локальной БД (охватывающей очередь сообщений, состояния автомата и модель предметной области) должна вестись в рамках одной транзакции.

При загрузки системы после восстановления следует достаточно корректно восстановить все таймеры, поэтому в состояние имеет смысл включить время последнего события или отдельно время запуска всех активных таймеров. В силу этого ясно, что таймауты автомата уместно реализовать не через системные таймеры, а один (или несколько) таймеров и ведение

списка будущих событий таймаутов в БД. Тоже самое касается организации очереди внешних событий.

В нашем д/з допустима, хотя и нежелательна, реализация автомата как отдельного потока с хранением его состояния в оперативной памяти[1]. В курсовом проекте это считается крупным недочётом. То же самое касается организации очереди событий — по соображениям надёжности событие следует помешать в постоянное хранилище как можно быстрее, а уже затем извлекать его оттуда для обработки. Это позволит не связывать вероятность потерять события со средним временем их обработки.

Остаётся отметить, что для упрощения реализации автомата внутри одного состояния можно запускать другой автомат и переходить в другое состояние в случае его завершения (а внутри него — ещё один и так далее). Однако, это затрудняет реализацию механизма сохранения и восстановления состояний.

4.8 Ограничения независимой обработки заявок

В данном разделе рассматривалась идея о независимой обработке заявок, когда обработка одной заявки влияет на другую только косвенно, изменяя состояния системы через бизнес-логику. Это может быть неудобным по ряду причин, поскольку в жизни регулярно обнаруживается, что стройные построения не всегда совместимы с реальностью.

Некоторые возникающие проблемы можно попытаться решить, не отказываясь от этой концепции: например, проблему генерации множества маленьких заявок смежнику можно решить их буферизацией («укрупнение входа»), другие — группируя заявки до начала их обработки («урпунение входа»), третьи — промежуточныи вариантом, используя в пределах одной системы два вида автоматов: одного для маленьких заявок, одного для укрупнённых.

4.9 Кеширование

Кеширование в РС заключается в хранении полученного состояния удалённой системы для дальнейшего использования. Получение состояние при этом часто разумно организовать заблаговременно, получая состояние системы с некоторым интервалом и выполняя каждый раз несколько попыток в случае неудачи.

О когерентности такого кеша в РС речь не идёт. Между получением состояния удалённой системы и попыток его изменить (т. е. захватить какие-то ресурсы) состояние удалённой системы всегда может поменяться произвольным образом, поскольку по данному определению РС в ней нет нет ни критических секций, ни блокировок. Поэтому отсутствие у кеша состояния удалённой системы свойства когерентности, видимо, не является нашей самой большой проблемой.

Закешированное состояние удалённой системы может использоваться пессимистически, как оценка сверху («час назад у ней не было нужного ресурса») или оптимистически: за оценку сверху берётся последнее количество ресурсов, увеличенное, к примеру, на 10 процентов. Следует также помнить, что удалённая система может приближённо сообщать о наличии ресурса («нет», «ждём», «есть») и имеют свою систему принятия решений, и только при обработке заявки станет точно известно, удастся ли нам его получить.

Основная проблема при кешировании – во ряде случаев состояние системы может быть слишком велико для того, чтобы быть переданным за разумное время. Причиной этому может быть слишком большое описание состояние, неэффективный формат передачи, медленные каналы связи, регулярная быстрая смена этого состояния (и высокие требования к «разумности» времени). В этом случае возможно, в теории, использовать репликацию, однако использование репликации (на уровне прикладного протокола) в РС находится под определённым вопросом и

[1]Кстати, что процессы Эрланга не тождественны потокам ОС и являются новой сущностью.

её, вероятно, разумно применять только к справочникам (например, ко всему каталогу продукции без указания наличия оной),

4.10 Тестирование и моделирование протокола. Представление о верификация протокола

Для тестирования протокола АИС её системы-смежники заменяются заглушками, реализующих полностью случайное, предположительное (случайное) или некоторое детерменированное (например, наихудшее) поведение. Тестирование реализации протокола РС «в лоб» может быть достаточно бессмысленно в силу наличия часовых и более таймаутов, поэтому тестируется «масштабированный» по величине тайм-аутов вариант.

Кроме различных видов тестирования, для исследования протокола может применяться имитационное и (в ряде случаев) аналитическое моделирование, как на основе создания отдельно модели (напрммер, в simpy), так и на основе использования самого кода протокола или его подмножества.

Тестирование и имитационное моделирование протокола, к сожалению, не может гарантировать обнаружения всех возможных проблем. Поскольку количество состояний самого протокола в каждой из ИС конечно, а от счётного[1] множества состояний основной БД системы можно отчасти абстрагироваться, то обычно можно создать модель, позволяющую полностью перебрать все состояния РС для обработки одной первоначальной заявки, и произвести её верификацию путём рассмотрения всего графа состояний (см. SPIN/Promela). У этого подхода есть два недостатка, небольшой: в реальности множество состояний может превосходить наши вычислительные возможности, и крупный: если верификация модели не выявила проблем, это не значит, что их нет у самой реализации (обратное проверяется).

5 Прочие подсистемы

5.1 Подсистема фильтрации

Подсистема фильтрации реализует часть политики фирмы по обработке заказов. В качестве своей главной своей задачи она пытается определить, можно ли взяться за исполнение заказа, учитывая :

— известные сведения о заказчике;
— известное состояние самой системы и предположительное состояние систем-партнёров.

В нашей работе мы будет использовать простейшую подсистему «принятия решений» в виде логики предикатов первого порядка, поскольку в этом курсе для нас важно понимать её наличие. На лекции давалась пара простейших подходов по определению вердикта (с весом и без).

Для описания правил принятия решения должен использоваться (в идеале) специализированный язык, достаточно понятный специалисту по бизнес-процессу. В его качестве на практических занятиях мы будем использовать пролог, а описывать правила мы будет в форме дизъюнктов Хорна.

Логично требовать, что принятие решений системой фильтрации должно быть достаточно быстрым, хотя и не обязательно почти моментальным: принятие решений в основном используется при обработке запросов на изменения состояния, которые обычно носят асинхронный характер. При обработке идемпотентных (и обычно синхронных) запросов на получение

[1] Игнорируя конечность памяти.

состояния логика принятия решения об обслуживании сообщения также может использоваться («а не слишком ли многое хочет знать о нас другая система?»).

Подсистема фильтрации решение принимает на основе:

— параметров заявки (*«кажется, здесь лишний ноль»*);
— состояния модели (например, базы данных с таблицами остатков товаров, поставками и платежами данного клиента);
— дополнительной статистике об истории «отношений» с обслуживаемой системой;
— представлений о состоянии смежников: можно ли получить недостающие ресурсы у третьих фирм;
— самих правил и значений констант в программке принятия решений (в нашем случае это будет простейшее сравнение пороговыми значениями).

Результатом работы подсистемы принятия решений может быть одно следующего списка:

— немедленно начать выполнять заявку;
— отклонить заявку;
— запросить подтверждение человека о выполнении заявки;
— подождать с выполнением некоторое время, по истечении которого «подумать» ещё раз;
— не выполнять, но ответить что приняли к исполнению;
— понизить приоритет заявки или сообщения;
— и так далее.

5.2 Интерфейс пользователя

В теории нам всё равно, как и на чём реализован интерфейс пользователя в РСОИ, лишь бы он не был монолитно связан с протоколом обмена сообщениями или с бизнес-логикой, а взаимодействовал с последней через чётко определенный интерфейс (например, на основе веб-сервисов).

В ходе практических занятий мы будет использовать исключительно MVC-каркасы для динамических языков, с целью познакомить слушателей с динамическими языками и MVC-каркасами.

Введение в MVC-каркасы описано в методических указаниях по первой л/р, здесь же напомним единственный момент. Моделью в нашем случае выступает не компонент доступа к базе данных (в примерах к MVC-каркасам в этой роли выступает ORM), а интерфейс доступа к бизнес-логике (которая, вместе со своею БД, и есть модель). Бизнес-логика, в свою очередь, может обращаться к логике работы протокола.

5.3 Замечания по бизнес-логике

С точки зрения данного в курсе определения, полноценные глобальные распределённые ACID-транзакции (использующие менеджер транзакций и произвольные ресурсы) в нашей РСОИ невозможны: система не может согласится на участие в транзакции, которая займёт неизвестный промежуток времени, зависящий от работы других систем.

Таким образом, подобные транзакции, охватывающие несколько ресурсов, могут охватывать только ресурсы одной отдельно взятой системы. Более того, любые ACID-транзакции начинаются и заканчивается во время обработки одного сообщения и, вероятно, даже при единственном вызове бизнес-логики. Решение, когда в одном состоянии автомата протокола транзакция начинается, а в другом — завершается, является, вероятно, полностью ошибочным.

6 Технологии удалённого взаимодействия и их применение в РС

6.1 Выбор нижестоящего протокола

В роли нижестоящего протокола для базового сетевого взаимодействия в РС в теории может использоваться и напрямую TCP или даже UDP. Однако обычно практично использовать работающий поверх TCP протокол, который позволяет передать произвольный текст с указанием его кодировки[1].

Таких протоколов у нас уже есть несколько. К данным протоколам относятся следующие:

— протокол HTTP и основанные на нём протоколы высокоуровневого удалённого вызова (SOAP, XML-RPC);
— протоколы почтовой системы SMTP, POP3 и IMAP (последний нам явно не нужен);
— протокол XMPP, используемый в Jabber.

В силу установленных требований к РС следующие протоколы не могут рассматривается как претенденты на нижестоящий протокол РС:

— что все платформенно-зависимые протоколы высокоуровневого удалённого вызова (Java RMI, Remoting.NET, PyRo и др.);
— ОС-зависимые протоколы (MSMQ, «низкоуровневый» классический RPC);
— кросс-платформенные, но слишком сложные протоколы, имеющие единственную референсную реализацию, такие как Zeroc Ice, так же сложно рекомендовать к использованию в качестве транспорта РС.

6.2 Механизмы синхронного взаимодействия

После отбрасывания всех платформо-зависимых RPC остались следующие кандидаты на роль синхронного (т.е. запроса-ответа протокола РСОИ, укладывающегося в один сеанс нижестоящего протокола) взаимодействия:

— прямое использование HTTP для передачи текстовых документов;
— использование HTTP с дополнительными соглашениями о методах HTTP (например, HTTP/REST);
— протоколы, основанные на создании поверх HTTP аналога удалённого вызова (SOAP, XML-RPC, JSON-RPC).

Представители всех этих видов рассмотрены в моих предыдущих опусах, где можно с ними и ознакомиться. Здесь же отметим следующие особенности.

Концепция HTTP/REST, возможно, и интересна как простой интерфейс для получение иерархических данных, но использование методов HTTP для указания операций с ресурсами (удаление, редактирование) с точки зрения РСОИ скорее неправильно. Вместо прямого редактирования в РС следует явно посылать сообщение о необходимости редактирования ресурса или заявки.

Концепция прозрачного удалённого вызова достаточно спорна с точки зрения РС. В частности, сообщение о невозможности установить соединение — типичный результат удалённого вызова. Кроме того, рекомендованный для РС интерфейс удалённого вызова — единственный удалённый вызов для каждой категории партнёров, принимающий и получающий документ на языке размётки.

Спецификация SOAP включает формат спецификации внешнего интерфейса (на языке WSDL), которая может использоваться для автоматической генерации кода вызова на стороне

[1] На всякий случай напомню, что нормальная кодировка в 21 веке — это UTF-8.

клиента. Хотя этот метод и позволяет легко создать простые удалённые вызовы веб-сервиса даже неквалифицированному программисту[1], его премущества для РСОИ с передаваемыми в виде сложных текстовых документов сообщений относительно невелики, поскольку XSD схема этих документов в WSDL не включается (если я не ошибаюсь; проверьте).

6.3 Механизмы передачи сообщений

Механизм передачи сообщений означает, прежде всего, саму инфаструктуру, поддерживающую надёжную передачу сообщений по маршруту, и, во-вторых, протоколы общения с сервером отправки сообщений и сервером получения сообщений. Инфраструктура эта может начинаться на localhost виде локального SMTP-сервера (ведь до localhost мы сможем достучаться почти всегда).

Поскольку все системы, реализующие стандарт Java Message System, использовать не рекомендуется как платформено-зависимые, то из живых систем выбор в настоящий момент ограничен двумя возможностями:

— почтовой системой с потоколами SMTP и POP3;
— системой обмена сообщениями jabber с протоколом XMPP.

Описывать эти протоколы здесь мне так же лень, они (как и HTTP/HTTPS) должны быть известны из курса ВКиС или английской википедии.

Хотя обмен системы были рассчитаны скорее для посылки сообщений от человека к человеку, ничто не мешает их использовать в качестве надёжного транспорта в РС. Недостатком использования POP3 является необходимость регулярно опрашивать POP3-сервер, но при необходимости это можно исправить, например использовав в конце маршрута передачи сообщения свой собственный модифицированный SMTP-сервер, непосредственно связанный с транспортной подсистемой.

Для отправки сообщения в локальной сети (или даже на локальной машине) требуется развёрнуть SMTP-smarthost, который и возьмёт на себя функции повтора попыток отправки на следующий SMTP-сервер (находящийся, возможно, на посреднике или на удалённой системе).

При использовании системы передачи сообщений внешний интерфейс системы с технологической точки зрения определяется её адресом и самим форматом сообщения на языке размётки.

Готовый механизм надёжной отправки рекомендуется использовать при посылке сообщений, изменяющих состояние удалённой системы, чтобы не реализовывать вручную повторы запросов по XML-RPC или подобному протоколу.

6.4 Использование языков размётки

Использование документов на языке размётки как единственной и независящей от транспортного протокола формы передачи сообщений позволяет решить ряд проблем. В первую очередь к ним относится хорошая переносимость системы на другой транспортный протокол. Ценой за это является первоначальное незначительное усложнение системы.

В настоящее время известны следующие языки размётки: JSON, XML, YAST. Более подробно они здесь описаны не будут, благо это уже сделано. На экзамене следует знать принципиальные различия между этими языками. В частности, следует отметить наличие в языке как мета-информации (имён данных), так и наличии формализма, определяющего вид корректного документа.

[1] Наблюдался клинический случай: настройка IP-адреса веб-сервиса в программе происходила путём удаления предыдущего сгенерённого кода и повторного его создания.

Кроме того, следует упомянуть и древний формат csv, не являющейся языком размётки и предназначенный в основном для представления табличной информации. Тем не менее, теоретически для представления сообщения может использоваться и он.

6.5 Спецификация документов на языке размётки

Спецификация формата входного и выходного сообщения (например, через XSD) позволяет однозначно описать множество корректных сообщений, что является плюсом XML. Однако, с точки зрения РС, вполне возможна ситуация, когда система начнёт присылать сообщения (или ответы на сообщения), несколько не соответствующие ранее полученной схеме XSD, или же будет менять схему раз в квартал.

Поэтому, хотя наличие схемы XSD безусловно упрощает обмен информацией, не следует отбрасывать документ, который не полностью удовлетворяет схеме, если это мешает продолжить работу системы. Например, перестановка двух полей местами не является проблемой, так же как и удаление неиспользуемого поля. Поэтому используемый анализатор документов не должен быть чрезмерно «строгим», если это мешает достижению целей системы.

Второй проблемой является изменение схемы в отсутствие разработчика системы. В идеале система должна сама пытаться приспособится под схему XSD, если между новым и старым форматом существует изоморфизм и известно новое название старых полей (мы можем пытаться использовать анализ распределения их значений, или же имена просо не менялись). Разработке подобных методов и средств может быть посвящён исследовательский курсовой проект.

Остаётся напомнить, что на многих платформах существуют средства автоматического генерации класса по XSD и сериализатора из/в XML-документ объекта этого класса. С одной стороны, это чрезвычайно полезно, с другой стороны, именно его слепое использование может привезти к тому, что после добавление невинного поля remark в сообщении ваша система «поймает» исключение вместо того, чтобы уведомить администратора и спокойно продолжить свою работу. Если у системы есть доступ к актуальной XSD-схеме, после обнаружения исключения при разборе документа надо пересоздать класс по новой схеме[1] в надежде, что все используемые поля класса остались на месте.

[1] Если язык это позволяет, конечно.