Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение
высшего образования «Петрозаводский государственный университет»

Институт математики и информационных технологий
кафедра <название выпускающей кафедры>

Фамилия Имя Отчество

НАЗВАНИЕ РАБОТЫ

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

Направление подготовки бакалавра
<вставить номер и наименование направления из инструкции>
Форма обучения - очная

Допущена к защите.
Заведующий кафедрой,
<степень>
<ФИО>
_____
«____»_____202_ г.

Руководитель
<степень>
<ФИО>
_____
«____»_____202_ г.

Петрозаводск — 20XX

Введение

В настоящее время мобильные устройства приобрели широкое распространение, что обуславливает актуальность разработки программного обеспечения (ПО). Несмотря на высокую производительность современных мобильных устройств, существует ряд задач, для которых ресурсов такого устройства будет недостаточно. Одним из вариантов решения данной проблемы является применение концепции интеллектуальных пространств (ИП, от англ. Smart Spaces) [11].

Концепция ИП является парадигмой программирования, в которой приложение реализуется не монолитным, а распределённым, состоящим из нескольких программ, называемых агентами или процессорами знаний (knowledge processor, KP). Распределённая архитектура описывает, из каких агентов состоит приложение и как осуществляется взаимодействие между ними. Агенты одного приложения работают достаточно независимо друг от друга и могут быть запущены на различных устройствах. Программная платформа Smart-M3 [6] представляет собой реализацию идеи ИП и позволяет создавать такого рода приложения.

Взаимодействие между агентами в платформе Smart-M3 осуществляется через брокер семантической информации (Semantic Information Broker, SIB). Обмен данными осуществляется при помощи механизма публикации и подписки [1]. Суть механизма заключается в следующем: агенты могут публиковать данные в SIB и могут подписываться на некоторые данные. Агент, имеющий подписку, получает обновления информации при её добавлении, изменении, или удалении. Причём публикующий KP не имеет никаких сведений об агентах, подписавшихся на его данные. При разработке приложения, использующего парадигму ИП, необходимо учитывать описанный способ взаимодействия между агентами.

Вся информация, хранящаяся в ИП, описывается при помощи модели данных Resource Description Framework (RDF) [12]. RDF-модель позволяет описывать предметную область и данные, используемые в интеллектуальном приложении. Формальное описание некоторой области знаний с помощью концептуальной схемы называется онтологией [5]. Следовательно, при проектировании приложения, базирующегося на платформе Smart-M3, важным является разработка его распределённой архитектуры, а также онтологии, которая полностью бы описывала предметную область приложения.

На данный момент уже существует ряд приложений, основанных на платформе Smart-M3: сбор контекстных данных на заседаниях [10], интеллектуальный дом [4], поддержка проведения конференций [17], погодное приложение [13].

В данной работе будет рассмотрена разработка интеллектуального приложения SmartScribo, позволяющего пользователю работать через мобильное устройство с несколькими блог-сервисами одновременно (мультиблоггинг). Значительное внимание будет уделено представлению данных блог-сервисов в ИП. Целью работы является построение онтологии представления знаний блогосферы, а также описание механизмов управления и использования этих знаний. Для достижения поставленной цели необходимо решить следующие задачи:

1. Определить структуру и классы знаний блогосферы.
2. Разработать онтологическую модель представления пользовательской информации и блог-данных.
3. Разработать механизмы управления онтологическими данными, а также синхронизации между агентами.
4. Разработать агент, взаимодействующий с блог-сервисом LiveJournal.
5. Предложить способ использования онтологии блогосферы с другими интеллектуальными приложениями.

Онтология интеллектуального приложения должна позволять описывать данные, используемые агентами, и представлять их с помощью RDF

модели. Онтология, описывающая предметную область, является важной составляющей интеллектуального приложения, так как благодаря ей агенты имеют представление о том, какие фрагменты данных необходимо обрабатывать.

Разработка экспериментального прототипа блог-клиента позволит оценить возможности платформы Smart-M3 при реализации подобного рода приложений. Кроме этого агенты, созданные для данной платформы, могут быть использованы в других приложениях, что позволит разрабатывать новое многофункциональное программное обеспечение без лишних затрат на кодирование.

Данная работа состоит из введения, трёх глав, заключения и списка литературы. В первой главе описывается устройство и архитектура программной платформы Smart-M3, а также принципы взаимодействия между собой интеллектуальных агентов. Во второй главе рассматривается проектирование блог-клиента для ИП и особое внимание уделяется способам представления и управления онтологической информации блогов. В третьей главе приводятся способы интеграции блог-клиента с другими интеллектуальными приложениями. В заключении подводятся итоги проделанной работы, формулируются полученные результаты.

Глава 1

Обзор платформы Smart-M3

Smart-M3 — это открытая программная платформа, воплощающая идеи семантического веба и реализующая инфраструктуру обмена информацией между программными сущностями и устройствами. В данной главе будут рассмотрены технологии, используемые в платформе, архитектура платформы и принципы построения интеллектуальных приложений.

1.1 Используемые семантические технологии

Семантический веб [3] – одно из направлений развития Интернет, целью которого является реализация возможности машинной обработки информации, доступной во Всемирной паутине. Основной акцент концепции делается на работе с метаданными, однозначно характеризующими свойства и содержание ресурсов Всемирной паутины, вместо используемого в настоящее время текстового анализа документов. Машинная обработка возможна в семантическом вебе благодаря двум характеристикам.

1. Использование унифицированных идентификаторов ресурсов (URI). Традиционная схема использования таких идентификаторов в современном Интернете сводится к установке ссылок, ведущих на объект, им адресуемый. Очевидным свойством такой ссылки является возможность «загрузки» объекта, на который она указывает. Таким объектом может быть веб-страница, файл произвольного содержания, фрагмент веб-страницы, а также неявное указание на обращение к реально существующему физическому ресурсу по протоколу, отличному от HTTP (например, ссылки mailto:). Концепция семантической паутины расширя-

ет это понятие, включая в него ресурсы, недоступные для скачивания. Адресуемыми с помощью URI ресурсами могут быть, например, отдельные люди, города и другие сущности.

2. Использование онтологий и языков описания метаданных. В семантическом вебе предлагается использовать форматы описания, доступные для машинной обработки (например, семейство форматов, часто упоминаемое в литературе как «Semantic Web family»: RDF, RDF Schema или RDF-S, и OWL), в свою очередь, использующие URI для адресации описываемых и описывающих объектов.

На платформе Smart-M3 используется схема описания данных основанная на RDF. Resource Description Framework (RDF) — это разработанная консорциумом Всемирной паутины модель для представления данных, в особенности – метаданных. RDF представляет утверждения о ресурсах в виде, пригодном для машинной обработки. Ресурсом в RDF может быть любая сущность – как информационная (например, веб-сайт или изображение), так и неинформационная (например, человек, город или некое абстрактное понятие). Утверждение, высказываемое о ресурсе, имеет вид «субъект – предикат – объект» и называется *триплетом*. Утверждение «небо голубого цвета» в RDF-терминологии можно представить следующим образом: субъект – «небо», предикат – «имеет цвет», объект – «голубой». Для обозначения субъектов, предикатов и объектов в RDF используются URI. Множество RDF-утверждений образует ориентированный граф, в котором вершинами являются субъекты и объекты, а рёбра помечены предикатами.

Также для работы с Smart-M3 можно использовать специальные генераторы. Генератор на основе файла, описанного в формате OWL, создаёт приложение, которое может взаимодействовать с платформой.

OWL — язык описания онтологий для семантического веба. Язык OWL позволяет описывать классы и отношения между ними, присущие для веб-документов и приложений. OWL основан на более ранних языках OIL и DAML+OIL и в настоящее время является рекомендованным консорциумом Всемирной паутины.

В основе языка — представление действительности в модели данных

«объект — свойство». OWL пригоден для описания не только веб-страниц, но и любых объектов действительности. Каждому элементу описания в этом языке (в том числе свойствам, связывающим объекты) ставится в соответствие URI.

1.2 Архитектура платформы Smart-M3

В мире существуют миллиарды мобильных устройств. Многие из них обладают неплохими размерами памяти и достаточной вычислительной мощностью. Если любые устройства смогут передавать друг другу информацию, то это качественно изменит жизнь людей, повысит уровень автоматизации по всех областях.

Ключевой идеей Smart-M3 является то, что устройства и программные сущности могут обмениваться между собой информацией. Одни из барьеров во взаимодействии между различными устройствами является разница в форматах информации, которую они распознают и с которой работают. Такие технологии как XML, RDF, OWL обеспечивают методы, позволяющие сфокусироваться на семантике информации а не на её синтаксическом представлении. Организация взаимодействия между устройствами и приложениями, работающими на них, является непростой задачей. Но был выбран прямолинейный подход: есть некоторое количество устройств и «доска», которая может использоваться для обмена информацией между этими устройствами, вместо того, чтобы устройства явно посылали друг другу сообщения. Кроме того, если эта информация будет храниться согласно некоторой онтологии, то станет возможен обмен этой информацией между устройствами, использующими разную модель представления данных, засчёт фокусировки на семантике информации.

M3 расшифровывается как multi-vendor, multi-device and multi-part (много продавцов, много устройств, много частей). Это означает что множество видов устройств могут взаимодействовать друг с другом, например, мобильный телефон, телевизор и ноутбук. Устройства могут состоять из частей, которые подразумеваются как отдельные участники взаимодействия с другим устройством. Кроме этого пользователь свободен в выборе

производителя. Smart-M3 разрабатывается так, чтобы была возможность работы в различных средах, учитывая их ограничения. Технология позволяет создавать программы, которые смогут использовать возможности, предоставляемые окружением устройства.

Подход Smart-M3 отклоняет прямое взаимодействие между устройствами и использует механизм «публикации и чтения». Сущность, предоставляющая информацию не нуждается во взаимодействии с читающей сущностью. Фактически эти две сущности даже могут не знать друг о друге. Smart-M3 предлагает, что публикующая сущность может разместить информацию в специальном месте обмена информации, а читающая сущность взять её оттуда.

Взаимодействие между сущностями в Smart-M3 можно логически разделить на три уровня:

- нижний уровень – «мир устройств», на нём находится множество устройств, соединённых сетью
- средний уровень – «мир сервисов», на нём находятся программы, клиенты и сервисы. Сервисы доступны только внутри конкретного домена. Обмен информацией осуществляется засчёт специальных протоколов обмена сообщениями
- верхний уровень – «умный мир», в нём взаимодействие основано на семантике информации. «Умный мир» унифицирует нижние уровни совместимостью на информационном уровне

Рассмотрим архитектуру платформы Smart-M3 (рис. 1.1). Центром системы является corpus-M3, который можно разбить на две главные части: брокер семантической информации (semantic information broker, далее SIB) и реальное физическое хранилище данных (БД).

SIB — это точка доступа для получения или отправки информации в хранилище. Вся информация в хранилище представлена в виде графа, который соответствует правилам RDF.

M3-agent — это приложение, которое взаимодействует с SIB: публикует или получает отуда информацию. M3-agent может находиться на другом

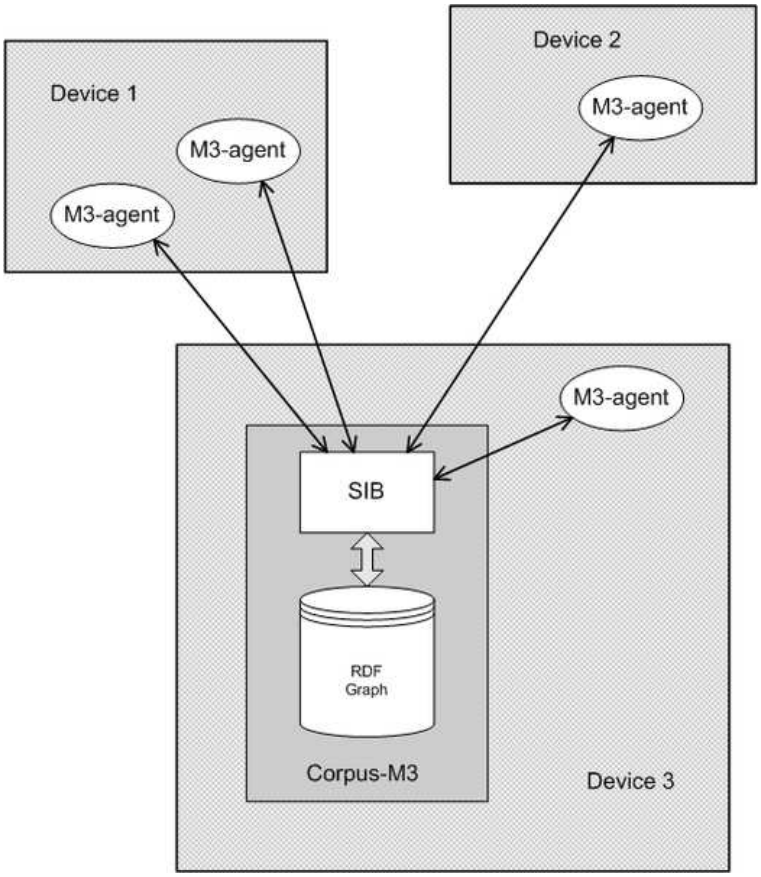

Рис. 1.1: Архитектура Smart-M3

устройстве и взаимодействовать с SIB, используя различные средства связи, поддерживаемые SIB. SIB может поддерживать множество протоколов передачи данных, таких как TCP/IP, Bluetooth и NoTA. В зависимости от окружения где запущен агент, выбирается более подходящая технология передачи информации. Конкретный M3-agent связан с конкретным SIB.

Также M3-agent называют «процессором знаний» (knowledge processor, далее KP), так как он обрабатывает информацию и публикует её в SIB или запрашивает её для обработки.

KP состоит из пользовательского интерфейса, логики обработки данных и «узла» (node). SmartSpace состоит из SIB и хранилища информации. На одном устройстве может работать множество KP. Пользовательский интерфейс для KP может быть не важен и представлять собой, например LCD дисплей или кнопку. «Узел» – это часть KP, которая содержит всю логику и функциональность для связи KP с некоторым SmartSpace (SIB): логика обработки сообщений и обработчики подписки между KP и SIB. KP может содержать множество «узлов» и тем самым

быть соединённым с несколькими SIB.

Идея Smart-M3 приводит к новому взгляду на разработку приложений. Приложение не является монолитным, а состоит из несколько частей — KP, которые взаимодействуют между собой через SIB. Количество агентов может быть различным в зависимости от конкретной ситуации и требований пользователя.

Взаимодействие между агентом и SIB осуществляется через протокол доступа к ИП (Smart Space Access Protocol, SSAP). Основные операции SSAP:

- Join – соединить KP с указанным пространство
- Leave – покинуть ИП
  После выполнения данной операции, другие операции не могут быть выполнены до повторного соединения с ИП.
- Insert – атомарно разместить информацию (RDF-граф) в ИП
- Remove – атомарно удалить информацию
- Update – атомарно обновить информацию
  Обновление является комбинацией операций удаления и добавления, но выполняется атомарно.
- Query – запросить информацию из ИП, используя поддерживаемый язык запросов
- Subscribe – подписаться на некоторую информацию и получать все её изменения
- Unsubscribe – отмена существующей подписки

Протокол SSAP гарантирует, что все операции будут выполнены в том порядке, в котором они были вызваны в агенте.

1.3 Понятие Smart-M3 приложения

Для интеллектуальных приложений необходимо проектировать сценарии, подразумевающие, что каждый из KP может быть запущен на раз-

личных устройствах. Каждый агент, выполняя некоторые действия, представляет собой сервис. Взаимодействие между КР можно рассматривать с двух точек зрения: с точки зрения использования сервисов и изменения данных в ИП. При выполнении операций join и leave агент соединяется или отсоединяется от ИП, а сервис, который он предоставляет, становится доступным/недоступным. При взаимодействии между собой агенты запрашивают и модифицируют данные из ИП. Несмотря на использование механизма публикации-подписки агенты могут взаимодействовать между собой и напрямую по сети, однако данный вид взаимодействия не относится к концепции ИП.

Каждый агент имеет представление о собственном наборе информации, являющейся частью глобального RDF-графа в ИП. Такие знания описываются онтологией агента. Для осуществления взаимодействия между агентами, необходимо наличие пересечения между онтологиями каждого из них. Таким образом в местах пересечения агенты могут отслеживать действия друг друга.

Все агенты можно разделить на три класса по характеру работы с данными: агенты-поставщики (только публикуют новые данные), агенты-потребители (только читают необходимые данные) и агенты, которые обрабатывают полученные данные и на их основе производят и публикуют новые знания.

На данный момент в платформе Smart-M3 существует несколько открытых проблем. Одна из них связана с ситуацией конкуренции между агентами за одни и те же ресурсы. Вследствие осутствия механизма синхронизации, результат такого взаимодействия неопределён.

Другой задачей является осуществление взаимодействия между агентами из разных приложений. Такие агенты обладают разными онтологиями и для решения данной проблемы необходима разработка методов осуществления композиции или пересечения онтологий. Таким образом для комбинации интеллектуальных приложений необходимо пересечение их ИП, которое может быть осуществлено благодаря пересечению онтологических знаний. В таких приложениях будут появляется агенты-посредники, взаимодействующие с несколькими пространствами. Подроб-

нее эта проблема рассматривается в главе 3.

Агенты взаимодействуют с SIB по протоколу SSAP через библиотеки, предоставляющие интерфейс процессора знаний (KP interface, KPI). Большинство таких библиотек предоставляет функции, выполняющие прямой доступ к протоколу SSAP. Соответственно работа агента с онтологическими данными заключается в обмене обработке триплетов. Такой подход является низкоуровневым.

Существует также высокоуровневый подход к осуществлению взаимодействия с онтологическими данными. В таком случае работа происходит с более общими сущностями вместо триплетов. Данный подход предусматривает наличие генератора, создающего библиотеку, основанную на онтологических сущностях [9]. Высокоуровневое взаимодействие позволяет разработчику работать с данными в терминах классов и их свойств.

В проекте SmartScribo, который будет рассмотрен в главе 2, использовалась низкоуровневая библиотека M3-Python KPI (m3_kp). Данная библиотека позволяет реализовывать интеллектуальные агенты на языке Python. В проекте использовалась библиотека m3_kp вследствие того, что она позволяет более простым способом организовывать взаимодействие с SIB, а также на языке Python намного проще выполнять другие операции приложения SmartScribo. Особенностью библиотеки m3_kp является выполнение каждой подписки в отдельном потоке, в отличие от библиотек, использующих языки C/C++, где выполнение подписки в потоке программисту приходится реализовывать самому.

Платформа Smart-M3 обладает широкими возможностями для разработки распределённых приложений и, в отличии от традиционного подхода к реализации приложения, позволяет расширять возможности программ, засчёт кооперации между различными интеллектуальными агентами и онтологического представления информации.

Глава 2

Модель персонального пространства в приложении SmartScribo

В данной главе рассматривается проект блог-клиента для мобильного устройства, использующего парадигму ИП, описывается его архитектура, структура персонального пространства блоггера и онтология для представления предметной области блогов.

2.1 Приложение SmartScribo

В настоящее время блоги являются одним из популярных видов социальных сетей. Блог — это сетевой дневник, основным содержимым которого являются регулярно добавляемые записи, изображения или мультимедиа. Блог-сервисом называется сервер, предоставляющий работу с блогами для пользователей. Совокупность всех блогов в сети называется блогосферой. Для блогов характерны недлинные записи временной значимости, отсортированные в обратном хронологическом порядке (последняя запись сверху). Отличия блога от традиционного дневника обусловливаются средой: блоги обычно публичны и предполагают сторонних читателей, которые могут вступить в публичную полемику с автором (в комментариях к блог-записи или своих блогах).

Популярность блогов настолько высока, что миллионы пользователей публикуют свежую информацию о себе или обмениваются мнениями по различным темам. Данный феномен быстро становится частью жизни человека, и блогосфера является подходящей областью для экспериментов

с использованием идей повсеместных вычислений [7, 2].

Существует множество клиентов, позволяющих работать с блогами, которые могут быть как браузерными, так и автономными. Автономные клиенты чаще всего используются на мобильных устройствах, так как они обеспечивают более эффективную работу с блогами, чем мобильный браузер. В таких клиентах обычно учитываются небольшие размеры экрана и клавиш устройства, а также ограниченность скорости сетевого трафика.

Проект SmartScribo [16, 15] представляет собой приложение, предназначенное для работы с различными блог-сервисами. Общая идея блоггинга в ИП представлена на рис. 2.1. Множество блогов, хранящихся на блог-сервисах и представленных пользователю в HTML, отображаются в отдельные пространства в общем ИП и представляются в RDF-формате. Ключевой особенностью приложения является возможность мультиблоггинга, которая позволяет пользователю работать одновременно с несколькими блогами.

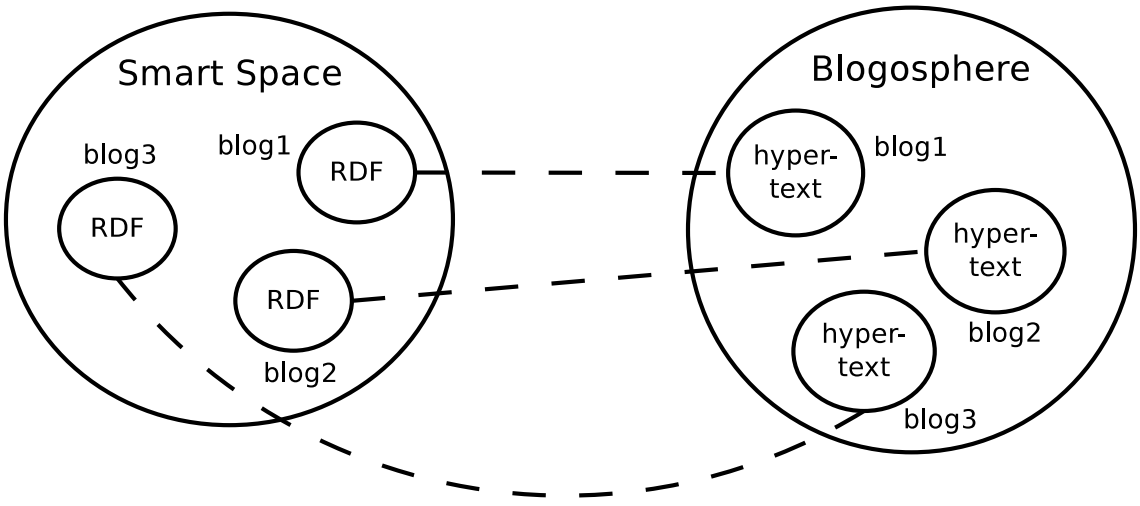

Рис. 2.1: Схема блоггинга в ИП

Основными возможностями приложения SmartScribo являются:

1. Просмотр нескольких блогов пользователя, как единого блога.
2. Получение, отправка, редактирование постов блога.
3. Кросс-постинг: отправка поста в несколько блогов одновременно.
4. Получение, отправка, редактирование комментариев поста.

5. Сортировка и фильтрация списка постов.

6. Просмотр блогов друзей.

Однако клиент-серверная архитектура является недостаточно эффективной для релизации мультиблогового клиента, особенно для мобильного устройства. Параллельная работа с несколькими блог-сервисами приводит к быстрой разрядке аккумулятора и резкому снижению скорости сетевого траффика. Следовательно возникает необходимость использовать альтернативную архитектуру, в качестве которой подходит архитектура приложений, основанных на парадигме ИП.

Распределённая архитектура SmartScribo (рис. 2.2), основанная на концепции ИП, позволяет пользователю взаимодействовать с несколькими блог-сервисами, а также предоставляет возможность работы нескольких клиентов одновременно. В приложении SmartScribo можно выделить три вида агентов.

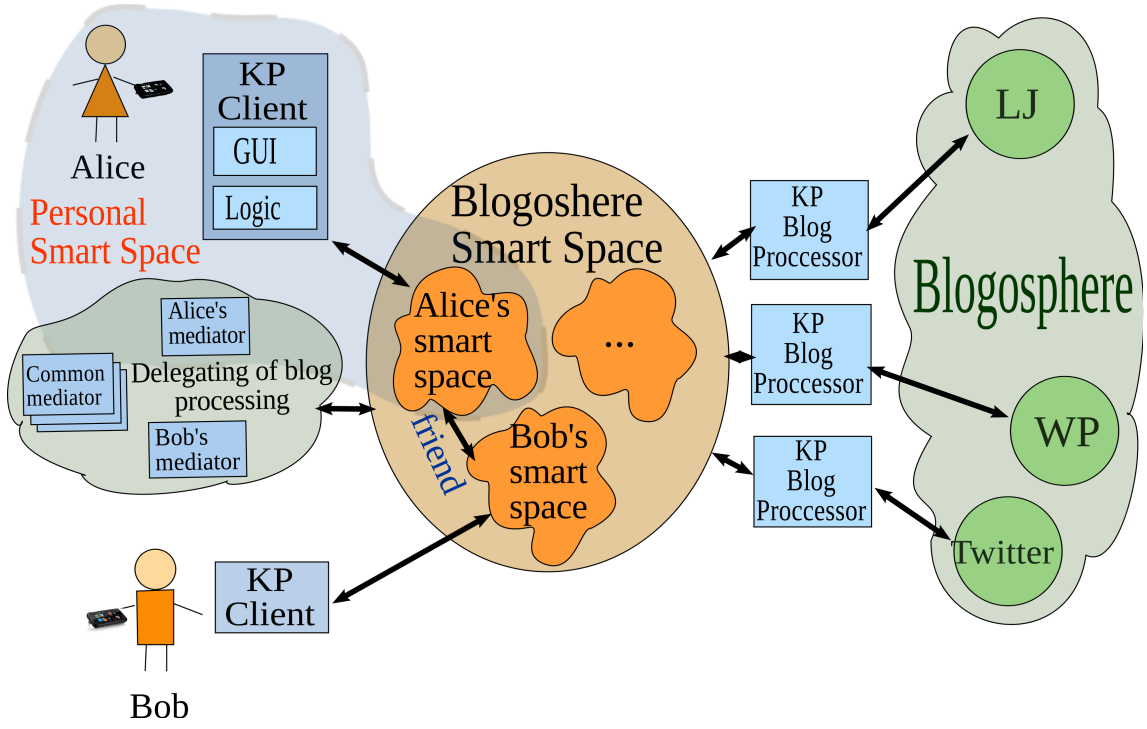

Рис. 2.2: Архитектура SmartScribo

1. Клиентский агент (KP Client) — агент, запускающийся на пользовательском устройстве. Клиентский агент позволяет пользователю работать с блогами и осуществляет данное взаимодействие через ИП. Таких

агентов может быть любое количество, и они могут быть запущены на различных устройствах: персональном компьютере, нетбуке или мобильном телефоне.

2. Агент блог-процессор (KP Blog Processor) — агент, который взаимодействует с определённым блог-сервисом и обрабатывает запросы от всех клиентов. Блог-процессор работает с аккаунтами пользователей, посылает/получает посты и комментарии и частично отражает содержимое блогосферы в ИП.

3. Агент медиатор (KP mediator) — данный агент предназначен для разного рода обработки информации, хранящейся в ИП. Агенты медиаторы могут быть как общими, т.е. использовать данные нескольких блоггеров, так и личными для каждого пользователя. В качестве примера функциональности медиатора могут служить такие операции, как оптимизация и удаление дублирующей информации в ИП или система рекомендаций, выполняющая поиск постов и предлагающая их прочитать пользователю на основе его интересов.

У каждого пользователя SmartScribo в глобальном ИП выделяется персональное пространство блоггера. Клиентский агент взаимодействует чаще всего именно с этим пространством, получая из него данные блогосферы. Кроме этого агент блог-процессор при ответе на запрос клиента также размещает блог данные в пользовательское пространство. Таким образом взаимодействие между клиентскими агентами и блог-процессорами осуществляется именно через персональное пространство блоггера.

Итак учитывая концепцию ИП, заключающуюся в онтологическом представлении знаний и механизме публикации-подписки, одной из наиболее важных задач является описание данных блогосферы, а также способов управления и обмена этими знаниями между агентами.

2.2 Структура персонального пространства блоггера

Блогосфера содержит большой объём информации, однако во время работы пользователя часто требуется доступ только к небольшой области знаний. Вследствие этого, для каждого пользователя мы выделяем

персональное пространство блоггера, содержащее необходимые пользовательские данные. Структура персонального пространства блоггера представлена на рис. 2.3.

Рис. 2.3: Структура персонального пространства блоггера

Знания, хранимые в персональном пространстве, можно разделить на три класса: 1) персональные данные (Person); 2) контекстная информация (Context); 3) блог данные (Blog data).

Персональные данные содержат постоянную или долговременную информацию (которая изменяется через продолжительный промежуток времени) о пользователе. В качестве примера можно привести такие данные как имя, фамилия, возраст, дата рождения, место проживания, адрес электронной почты, интересы и др. Данная информация не связана и не зависит от блог данных или блогосферы и, следовательно, может быть использована другими интеллектуальными приложениями.

Контекстные данные содержат текущие или часто изменяемые характеристики пользователя, такие как текущее местоположение (координаты мобильного устройства), погода в текущей местности, музыкальный трек,

прослушиваемый в данный момент, настроение пользователя. Кроме того контекст может хранить информацию и о блогах, например пост, который читает пользователь в данный момент. Контекстная информация может быть использована агентами медиаторами для осуществления рекомендаций пользователю.

Блог данные содержат информацию о всех аккаунтах пользователя на различных блог-сервисах, постах на каждом аккаунте и комментариях к постам. В ИП одновременно могут находиться не все данные из блогосферы, а только та информация, которая будет необходима пользователю в ближайшее время.

Следует отметить, что перечисленные классы знаний могут быть связаны между собой. Например персональная информация может содержать данные о том, что у пользователя есть несколько аккаунтов на блог-сервисах и, соответственно, образуется связь между персональными и блог данными. Кроме того блог данные, в свою очередь, могут влиять на контекстную информацию, отражая в ней, например, читаемый в данный момент пост или блог.

Пример содержимого персонального пространства блоггера представлен в табл. 2.1 и 2.2. Блог-данные состоят из объектов: аккаунтов, постов и комментариев, и каждый из них обладает собственными свойствами. В случае, если пользователь Alice решит изменить, например, заголовок своего поста (post-1), то в персональном пространстве значение свойства заголовка поста (title) будет изменено.

Таблица 2.1: Пример персональных данных и контекста пространства

Class	**Property**	**Value**
Person	name	Alice
	age	20
	city	Petrozavodsk
	account	account1
	...	...
Context	coordinates	61.7;34.2
	track	Mozart – Lacrimosa
	...	...

Таблица 2.2: Пример блог-данных пространства

Blog data		
Blog object	**Property**	**Value**
account-1	hasPost	post-1
post-1	title	My journey
post-1	text	My journey was so wonderful!

Персональные пространства блоггеров могут взаимодействовать друг с другом и образовывать композицию. Существует несколько способов организации взаимодействия между пользовательскими пространствами.

1. Дублирование данных. В данном случае при взаимодействии с другим пользовательским пространством, необходимые данные копируются в пользовательское пространство через блог-процессор. Такой способ используется если пользователь, данные которого получаются, не пользуется SmartScribo и данные а его аккаунте, постах и комментариях сохраняются в персональном пространстве текущего пользователя. Однако в случае, если такой пользователь будет иметь своё персональное пространство в ИП, то его данные будут продублированы.

2. Установление связи. Композиция заключается в установлении связей между знаниями из разных пользовательских пространств. Примером композиции может быть отражение дружеских отношений на блог-сервисах между двумя пользователями SmartScribo (рис. 2.4). Если аккаунты обоих пользователей являются друзьями на блог-сервисе, то в ИП можно определить связь «Friend» между этими аккаунтами, и в дальнейшем для получения информации блоггером о его друге достаточно будет обратиться в дружеское персональное пространство, минуя обращение к блог-сервису.

3. Использование агента-посредника. При использовании данного подхода агент, работающий с конкретным персональным пространством получает данные из другого персонального пространства через специальный агент посредник. Одним из способов реализации данного подхода является механизм нотификаций, который будет расписан подробнее далее.

Персональное пространство блоггера содержит структурированную

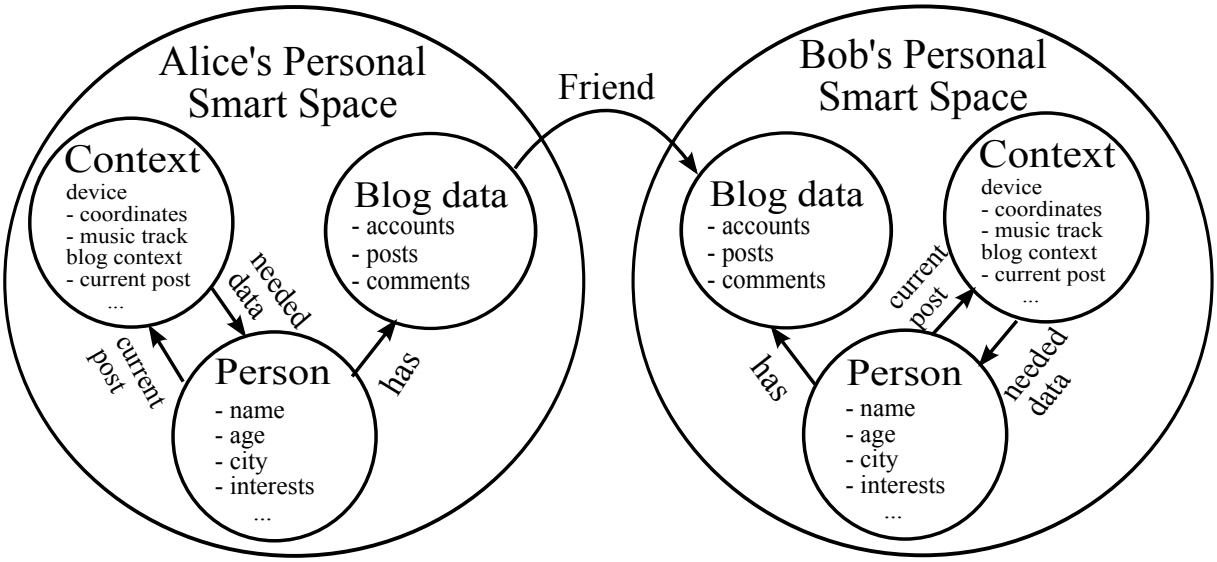

Рис. 2.4: Композиция персональных пространств

информацию о пользователе, его контексте и блог-данных. Для того чтобы использовать пространство в интеллектуальном приложении, необходимо описать его онтологическое представление.

2.3 Онтологическая модель блогосферы

В соответствии с устройством программной платформы Smart-M3 все знания, хранящиеся в ИП, представлются в виде RDF-триплетов. Совокупность RDF-триплетов, описывающих предметную область, образует онтологию, которая может быть представлена в виде классов и свойств. Следовательно для представления и хранения данных персонального пространства, включающих блогосферу, необходимо описать онтологию.

Для онтологического представления персонального пространства блоггера за основу была взята стандартная спецификация FOAF [14]. Она определена как словарь именованных классов и свойств с использованием технологий RDF и OWL. FOAF позволяет описывать некоторые основные данные о пользователе, отношения между пользователями, а также наличие аккаунтов на некоторых сервисах. Профили FOAF могут быть использованы для поиска других людей со схожими характеристиками.

Однако FOAF не позволяет описывать непосредственно содержимое блог-аккаунтов, и следовательно возникает необходимость расширения

данной онтологии для представления блог данных. Использование стандартных онтологий является одним из основных принципов онтологического моделирования, так как позволяет унифицировать фрагменты общей онтологии, вследствие чего другим разработчикам будет проще работать с новой онтологией. Онтология, используемая в SmartScribo, изображена на рис. 2.5. Данная онтология описана в терминах OWL классов и их свойств.

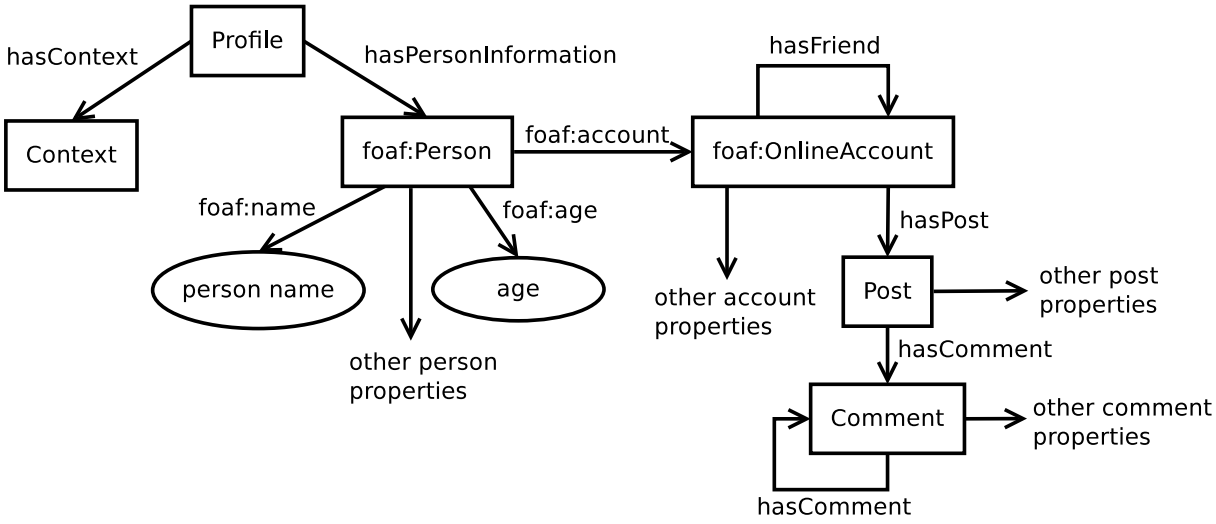

Рис. 2.5: Онтология SmartScribo

Корнем онтологического дерева является класс `Profile`, хранящий в себе остальную структуру знаний персонального пространства и блогосферы. Каждый экземпляр данного класса соответствует только одному пользователю или пользовательскому устройству и имеет свой уникальный идентификатор. Класс связан с персональной информацией о пользователе через свойство «hasPersonInformation» и с контекстными данными через свойство «hasContext». В формате OWL описание данного класса и связь его с классом FOAF имеет следующий вид:

```
<!-- Описание класса Profile -->
<owl:Class rdf:ID="Profile"/>

<!-- Описание свойства-связи personInformation
     между классами Profile и foaf:Person -->
<owl:ObjectProperty rdf:ID="personInformation">
  <rdfs:domain rdf:resource="#Profile"/>
  <rdfs:range rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
</owl:ObjectProperty>
```

Тег `owl:ObjectProperty` определяет объектное свойство (связь) между двумя классами, где тег `rdfs:domain` — это класс, обладающий данным свойством, а `rdfs:range` — тип значения свойства (в объектном свойстве тип значения должен быть другим классом). Ссылка на класс или свойство осуществляется при помощи атрибута `rdf:resource`.

Класс `Context` содержит частоизменяемую и временную контекстную информацию пользователя. Все знания, хранящиеся в этом классе преимущественно имеют текстовый вид, например текущие координаты, имя музыкального трека, настроение пользователя. Однако класс контекстной информации может содержать и ссылки на другие области знаний онтологии, такие как ссылка на пост из блог-данных, читаемый пользователем в данный момент. Контекстные данные выделены в отдельный класс, вследствие того, что множество интеллектуальных агентов могут получать или изменять контекст пользователя. Данный класс расположен на втором уровне иерархии классов в онтологии, поэтому другие агенты могут быстро получить доступ к этим данным, зная только идентификатор профиля пользователя.

Класс `foaf:Person`, взятый из спецификации FOAF, содержит постоянные или редко-меняющиеся характеристики пользователя. В онтологии SmartScribo используются следующие свойства FOAF: имя пользователя («foaf:name»), возраст («foaf:age»), дата рождения («foaf:birthday»), адрес электронной почты («foaf:mbox»). Эти данные вместе с контекстом могут быть использованы агентами-медиаторами SmartScribo для предоставления рекомендаций пользователю. Кроме этого класс `foaf:Person` обладает свойством «foaf:account», которое связывает пользовательскую информацию с классом `foaf:OnlineAccount`, содержащим данные об аккаунтах пользователя. Таким образом через указанное свойство связаны персональная информация и блог-данные.

Класс `foaf:OnlineAccount` представляет собой класс FOAF, описывающий конкретную учётную запись на некотором сервисе. Свойство «foaf:Name» описывает имя аккаунта. В онтологии SmartScribo данный класс расширяется набором дополнительных свойств: «login» – логин пользовательского аккаунта, «password» – пароль учётной записи для осу-

ществления авторизации на сервисе блог-процессором (в ИП хранится в зашифрованном виде), «userpic» – ссылка до изображения иконки пользователя на сервисе, «bdate» – дата рождения, указанная пользователем на сервисе, «groupname» – группа в которой состоит данный аккаунт. Кроме этого аккаунт обладает свойством «hasPost», связывающим этот аккаунт с индивидами класса `Post`, т.е. со всеми постами указанной учётной записи. На языке OWL расширение класса `foaf:OnlineAccount` осуществляется следующим образом:

```
<!-- Описание строкового свойства bdate
     у класса foaf:OnlineAccount -->
<owl:DatatypeProperty rdf:ID="bdate">
  <rdfs:domain rdf:resource="http://xmlns.com/foaf/0.1/OnlineAccount"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>

<!-- Описание свойства-связи hasPost
     между классами foaf:OnlineAccount и Post -->
<owl:ObjectProperty rdf:ID="hasPost">
  <rdfs:domain rdf:resource="http://xmlns.com/foaf/0.1/OnlineAccount"/>
  <rdfs:range rdf:resource="#Post"/>
</owl:ObjectProperty>
```

Тег `owl:DatatypeProperty` определяет свойство у класса, указанного в теге `rdfs:domain`, с типом данных этого свойства, указанным в теге `rdfs:range`.

Класс `Post` содержит полную информацию о конкретном посте на определённом аккаунте блог-сервиса. Данный класс содержит свойства, отражающие основные характеристики поста: «title» – заголовок поста, «text» – текстовое содержимое поста, «pdate» – дата создания или последнего изменения, «tags» – тэги поста, «journal» – необязательное свойство, содержащее имя группы, в которую был отправлен пост. В онтологическом представлении посты не являются упорядоченными, поэтому их упорядочивание по дате должно происходить на стороне получающего агента по свойству «pdate». Класс `Post` использует свойство «hasComment» для связи данного поста с комментариями.

Класс `Comment` содержит данные, относящиеся к комментарию, и

схож с классом `Post`. Данный класс включает в себя свойства «title», «text», «pdate», «journal». Комментарий также обладает свойством «hasComment», так как на блог-сервисах есть возможность отправки комментария на комментарий. Таким образом каждая блог-дискуссия, состоящая из поста и его комментариев, имеет древовидную структуру.

Пример онтологического представления персонального пространства блоггера изображён на рис. 2.6. В данном примере отражены данные из табл. 2.1 и 2.2. Прямоугольниками индивиды соответствующих классов: `Profile`, `foaf:Person`, `Context`, `foaf:Account`, `Post`. Для удобства определения данных в ИП, имя индивида состоит из названия класса и уникального идентификатора, но фактически имя индивида может быть произвольным и уникальным. Овалами на изображении обозначены конкретные значения свойств индивидов. Если пользователь Alice решит изменить, например, заголовок своего поста (post-e87a25d1) на «My weekend», то триплет «post-e87a25d1, title, My journey» будет заменён на триплет «post-e87a25d1, title, My weekend».

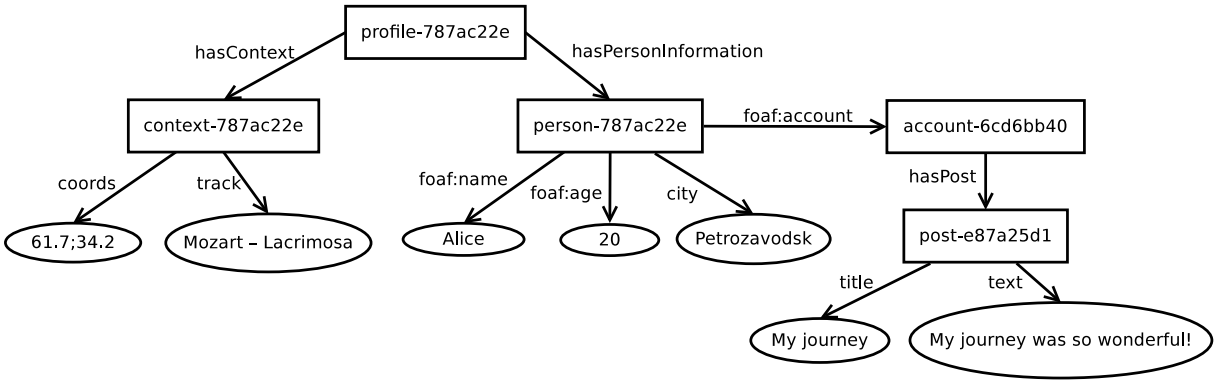

Рис. 2.6: Пример онтологического представления пространства

Стоит отметить, что для управления онтологическими данными блогосферы подписка на объекты онтологии не является эффективной, вследствие того, что даже на одном аккаунте для определения агентом изменений в посте или комментарии необходимо подписаться на большое количество триплетов (должны отслеживаться изменения всех свойств постов или комментариев). Отсюда следует, что необходимо создание более эффективного механизма, позволяющего определять изменения онтологических данных блогосферы.

2.4 Модель нотификаций

Онтологические данные блогосферы состоят из множества однотипных объектов, управление которыми является неэффективным при использовании подписки на их свойства, так как создаётся большое количество подписок и сложность их обработки возрастает. Кроме этого при существовании множества клиентов и блог-процессоров SmartScribo необходимы способы координации взаимодействия этих агентов. В данном разделе мы рассмотрим вариант решения упомянутых проблем — модель нотификаций.

Модель нотификаций — это онтологическая модель, предназначенная для координации взаимодействия между агентами SmartScribo. Нотификация инициирует соответствующего агента выполнить указанную операцию или сообщает о результате выполнения операции. Нотификация представляет собой триплет или набор триплетов, которые соответствуют выполнению действия или его результату.

Общая схема процесса нотификации, рассмотренная на примере взаимодействия клиентского агента и блог-процессора, изображена на рис. 2.7. Процесс нотификации состоит из пяти шагов.

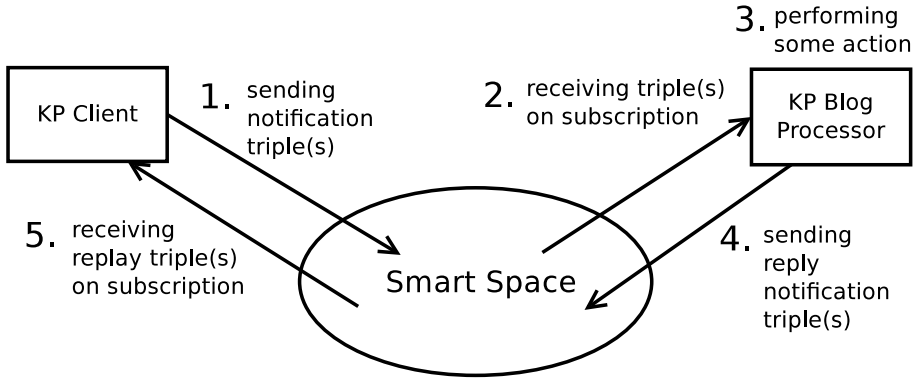

Рис. 2.7: Схема нотификации для двух агентов: клиента и блог-процессора

1. Клиентскому агенту необходимо инициировать выполнение требуемого действия у блог-процессора. Клиент публикует соответствующий триплет нотификации в ИП.

2. Агент блог-процессор получает адресованную ему нотификацию вместе со всеми параметрами нотификации (получение происходит за-

счёт подписки на триплет нотификации).

3. Агент-получатель выполняет необходимую операцию в соответствии с параметрами нотификации.

4. Агент-получатель публикует в ИП ответную нотификацию с результатами выполнений действия.

5. Клиентский агент получает ответную нотификацию, через механизм подписки.

Нотификации могут быть разделены на два типа в зависимости от момента их отправки и ожидания ответной нотификации агентом инициатором: реактивные и проактивные.

Реактивная нотификация инициирует у агента выполнение требуемой операции и высылается в зависимости от действия пользователя. Инициирующий агент ожидает ответной нотификации с результатом выполнения операции. Примером реактивной нотификации может быть нотификация проверки логина и пароля учётной записи или нотификация отправки поста на блог-сервис.

Проактивная нотификация высылается в фоновом режиме и только косвенным образом зависит от действий пользователя. Инициирующий агент чаще всего не ожидает ответной нотификации. В большинстве случаев проактивная нотификация влияет или зависит от контекстной информации пользователя. В качестве примера такой нотификации можно привести нотификацию, инициирующую поиск постов схожих по тематике с постом, читаемым в данный момент.

Модель нотификаций является надстройкой к онтологической модели блогосферы. В каждой нотификации чаще всего содержится связь с онтологическими данными, т.е. триплет нотификации указывает на индивида из онтологии блогосферы.

Нотификации могут быть простыми, состоящими из одного триплета, и сложными, представленными в виде цепочки триплетов. Простая нотификация используется в том случае, когда инициируемому агенту требуется передать только один параметр, необходимый для выполнения операции.

Общий вид простой нотификации следующий:

Notification<`type`>, <`action`>, <`value`>

где `type` содержит тип блог-процессора или идентификатор клиента, которому адресована нотификация, `action` — название действия которое должно быть выполнено или было выполнено (в ответной нотификации), а `value` — имя необходимого индивида в онтологии или результат выполнения действия. Все ответные нотификации являются простыми.

Сложная нотификация представляет собой цепочку триплетов и используется в случае, когда для выполнения действия необходимо передать несколько параметров. Общий вид сложной нотификации следующий:

Notification<`type`>, <`action`>, notif_ind<`id`>
notif_ind<`id`>, <`parameter1`>, <`value1`>
notif_ind<`id`>, <`parameter2`>, <`value2`>
...

где notif_ind<`id`> дополнительный индивид с идентификатором `id`, хранящий все значения параметров нотификации, `parameter` — название параметра и `value` — значение этого параметра (ссылка на онтологические данные или строка).

Нотификации можно классифицировать по типу объекта над которым выполняется действие:

1. Работа с аккаунтами.

Для работы с аккаунтами предназначена только одна нотификация, которая инициирует получение данных об аккаунте с блог-сервиса, а также проверку логина и пароля:

Notification<`service`>, refreshAccount, <`acc_id`>

`service` — тип сервиса (LJ – LiveJournal, TW – Twitter, и т.д.)
`acc_id` — ссылка на индивида необходимого аккаунта (идентификатор индивида)

ответная нотификация, возвращающая результат получения данных аккаунта и проверки логина и пароля, имеет следующий вид:

Notification<`client_id`>, refreshAccount, ok/error

`client_id` — идентификатор индивида профиля пользователя (класс Profile)

Все ответные нотификации имеют аналогичный вид, поэтому далее они указываться не будут.

2. Работа с постами.

- обновление постов указанного аккаунта

  Notification<`service`>, refreshPosts, <`acc_id`>

- отправка поста в собственный блог или в группу

  Notification<`service`>, sendPost, post_notif<`id`>
  post_notif<`id`>, postAcc, <`acc_id`>
  post_notif<`id`>, postId, <`post_id`>
  post_notif<`id`>, journal, <`journal_name`>

  свойство «postAcc» хранит ссылку на индивида аккаунта `acc_id`, от имени которого будет послан пост
  свойство «postId» хранит ссылку на индивида поста `post_id`, данные которого будут посланы на блог-сервис
  свойство «journal» хранит название группы в которую будет опубликован пост (таких триплетов может быть несколько)

- редактирование

  Notification<`service`>, editPost, post_notif<`id`>
  post_notif<`id`>, oldPost, <`post1_id`>
  post_notif<`id`>, newPost, <`post2_id`>

  Пост с идентификатором `post1_id` будет отредактирован на блог-сервисе в соответствии с данными пости с идентификатором `post2_id`. В ИП старый пост будет удалён, а отредактированный пост будет связан с аккаунтом.

- удаление

  Notification<`service`>, delPost, post_notif<`id`>
  post_notif<`id`>, postAcc, <`acc_id`>

post_notif<`id`>, postId, <`post_id`>
post_notif<`id`>, journal, <`journal_name`>

Удаляется пост, имеющий идентификатор `post_id`, с указанного аккаунта `acc_id`, а также из групп `journal_name`

3. Работа с комментариями.

- обновление комментариев указанного аккаунта

  Notification<`service`>, refreshComments, <`acc_id`>

- отправка

  Notification<`service`>, sendComment, com_notif<`id`>
  com_notif<`id`>, comAcc, <`acc_id`>
  com_notif<`id`>, comId, <`com_id`>
  com_notif<`id`>, parId, <`parrent_id`>
  com_notif<`id`>, journal, <`journal_name`>

  Отправка комментария, имеющего идентификатор `com_id`, от имени пользователя аккаунта `acc_id`. Комментарий отправляется на пост или другой комментарий с идентификатором `parrent_id`, либо в группу `journal_name`.

- удаление

  Notification<`service`>, delComment, com_notif<`id`>
  com_notif<`id`>, comAcc, <`acc_id`>
  com_notif<`id`>, comId, <`com_id`>
  com_notif<`id`>, parId, <`parrent_id`>
  com_notif<`id`>, journal, <`journal_name`>

  Удаление комментария, имеющего идентификатор `com_id`, от имени пользователя аккаунта `acc_id`. Комментарий удаляется у пост или другого комментария с идентификатором `parrent_id`, либо в группе `journal_name`.

4. Работа с друзьями.

- добавление пользователя в друзья

  Notification<`service`>, addFriend, friend_notif<`id`>
  friend_notif<`id`>, ownAcc, <`acc_id`>
  friend_notif<`id`>, friendName, <`name`>

  Добавление в аккаунт `acc_id` пользователя с именем `name`.

- удаление пользователя из друзей

  Notification<`service`>, delFriend, friend_notif<`id`>
  friend_notif<`id`>, ownAcc, <`acc_id`>
  friend_notif<`id`>, friendName, <`name`>

  Удаление пользователя с именем `name` у аккаунт `acc_id`.

- обновление списка друзей

  Notification<`service`>, refreshFriends, <`acc_id`>

- обновление данных о постах друзей

  Notification<`service`>, refreshFriendsPosts, <`acc_id`>

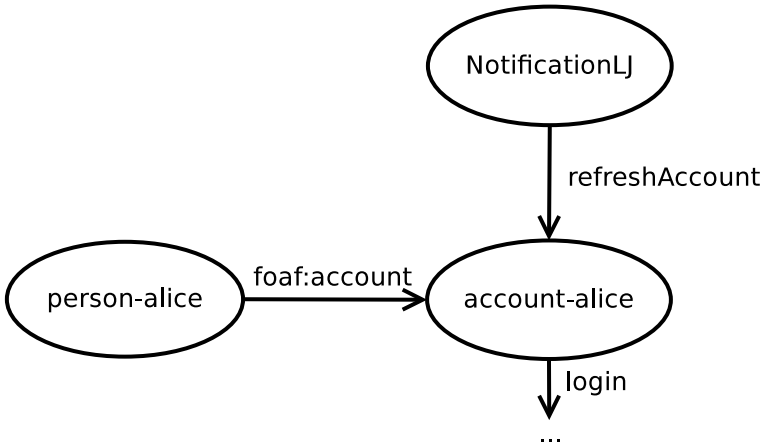

Рис. 2.8: Пример нотификации

Пример использования нотификации представлен на рис. 2.8. Пользователь Alice обладает аккаунтом на блог-сервисе LiveJournal. Информация о пользователе и аккаунте хранится в персональном пространстве в ИП у индивидом с идетификаторами «person-alice» и «account-alice» соответственно. Для синхронизации данных об аккаунте, а также проверки

указанного логина/пароля клиентский агент Alice высылает нотификацию вида «NotificationLJ, refreshAccount, account-alice». Блог-процессор, взаимодействующий с LiveJournal, получает данную нотификацию и зная идентификатор индивида аккаунта Alice (объект в триплете), получает возможность обратиться к свойствам аккаунта (логин, пароль, и т.д.) и выполнить соответствующую операцию.

Таким образом нотификации являются удобным механизмом управления онтологическими данными, позволяющим обрабатывать информацию, разделять подписки агентов на триплеты (каждый агент получает нотификации, адресованные только ему), а также синхронизировать взаимодействия между агентами.

2.5 Реализация блог-процессора

В данном разделе будет рассмотрена работа с онтологическими данными на стороне блог-процессора.

Архитектура блог-процессора представлена на рис. 2.9. В блог-процессоре можно выделить две основных подсистемы: подсистема взаимодействия с блог-сервисом (Blog service connector) и подсистема взаимодействия с ИП (Smart-M3 Node).

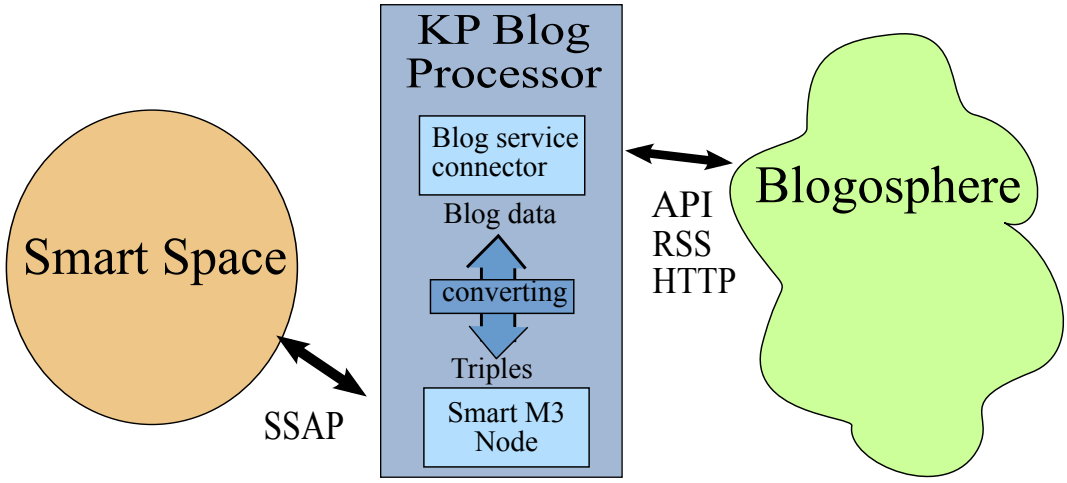

Рис. 2.9: Архитектура блог-процессора

Подсистема взаимодействия с блог-сервисом должна осуществлять следующие функции:

- авторизация на сервисе

- получение информации о пользователе (профиль + списки друзей)
- получение постов для указанного аккаунта
- отправка/изменение/удаление постов
- получение комментариев для указанного аккаунта
- отправка/удаление комментариев
- добавление/удаление друзей

Существует несколько способов взаимодействия с блог-сервисом, каждый из которых обладает своими преимуществами и недостатками.

1. Использование API блог-сервиса. Многие блог-сервисы предоставляют API для взаимодействия с ними. API является удобным способом взаимодействия с сервисом и позволяет получать данные в какомлибо популярном формате. Например LiveJournal предоставляет возможность взаимодействия по протоколу XML-RPC и получения данных в XML формате, который прост в обработке. API позволяет как получать блог-данные с сервиса, так и отправлять их на сервис, однако данный способ взаимодействия не всегда предоставляет все необходимые функции для блоггинга (например на некоторых сервисах нет возможности работы с комментариями) и для работы обязательно необходима авторизация пользователя.

2. Использование RSS. RSS позволяет получать основные данные с блог-сервиса не требуя авторизации пользователя. Данные полученные с помощью RSS также представлены в формате XML и легко могут быть обработаны. Несмотря на это возможности RSS весьма ограничены, так как многие данные невозможно получить с помощью этого способа взаимодействия (возможно получение только фиксированное количество последних добавленных постов, комментарии не поддерживаются). Кроме этого при использовании RSS возможно только получение данных.

3. Использование HTTP запросов. Наиболее сложным способом взаимодействия с блог-сервисом является использование HTTP запросов. Данный подход эмулирует работу веб-браузера и, соответственно, позво-

ляет использовать все возможности блог-сервиса. Однако реализация этого способа взаимодействия трудоёмка в связи с тем, что необходимо определять, какие запросы следует отправлять на сервер, а также производить разбор HTML страниц.

Подсистема взаимодействия с ИП получает данные от подсистемы взаимодействия с блог-сервисом и преобразует их в триплеты, соединяется с SIB, используя протокол SSAP, и осуществляет обработку онтологических данных (публикация, получение, подписка).

В общем случае существует несколько способов обработки онтологических данных агентами. Они различаются способом представления локальных данных в агенте. Основными моделями представления данных являются онтологическая и ОО модели.

1. Онтологическая модель. При использовании онтологической модели представления, данные хранятся в локальной памяти агента в виде графа из RDF-триплетов, схожим с RDF-графом ИП. Некоторые блог-сервисы работают с онтологическим представлением данных, например Yandex предоставляет возможность взаимодействия с блогом с использованием спецификации FOAF.

Преимущества данного подхода заключаются в том, что в локальной памяти хранится структура данных совпадающая с ИП. При работе с локальной копией данных, появляется возможность использования специализированных средств обработки, таких как язык SPARQL для построения сложных запросов. Локальные онтологические данные также могут быть обработаны с помощью и других сторонних приложений. При использовании онтологического представления данных появляется возможность за один запрос получить большой объем данных и далее работать с ними локально, что уменьшит количество взаимодействий с ИП.

Недостатком онтологического представления является требование к частой синхронизации локальных данных с ИП. При любом изменении в локальной копии данных, необходимо отражать эти изменения в ИП, так как это может повлиять на работу других интеллектуальных агентов. Также работа с онтологическими данными не всегда является простой и может потребовать применения или разработки специальных алгоритмов

обработки информации.

2. Объектно-ориентированная модель. Представление данных заключается в преобразовании полученных из ИП RDF-триплетов в объекты, хранящиеся в локальной памяти агента. При необходимости публикации данных в ИП, объект обратно преобразуется в триплеты.

Данный подход позволяет для каждого класса определить конкретные функции, используемые в определённых сценариях. При работе с данными проще осуществлять обработку объектов, чем RDF-графа. Примером может являтся работа с постами блога, которые преобразованы в объекты. Вследствие этого, несложно будет отсортировать все посты по дате в массиве постов-объектов.

Такой подход к представлению данных позволяет работать с объектом, как с выделенным фрагментом онтологии. В данном случае легко определить, какой фрагмент онтологических данных будет модифицирован, и отразить эти изменения в ИП. Преобразование RDF-триплетов в объекты также даёт возможность взаимодействия с сервисом посредством специализированных библиотек, таких как QMF. Такой подход будет разумным, если в QMF появятся плагины для соответствующих блог-сервисов.

Однако постоянное преобразование из триплетов в объекты и обратно приводит к дополнительным расходам ресурсов устройства, а также частому взаимодействию с ИП.

При разработке клиентского агента SmartScribo использовалась ОО-модель представления данных. Таким образом в памяти агента хранятся объекты, соответствующие аккаунтам пользователя, а также постам и комментариям в блогах. Данный подход позволяет осуществлять упорядочивание, сортировку, фильтрацию постов и комментариев, а также объединение всех блогов пользователя в один единый виртуальный блог.

В реализации агента блог-процессора применяется частный случай ОО-модели представления данных. Блог-процессору не требуется долгое время хранить в памяти данные об аккаунтах, постах, комментариях, так как он является транзитным. Блог-процессор получает структуры данных от библиотеки, взаимодействующей с блог-сервисом, преобразует их

в триплеты и публикует в ИП, и аналогично в обратном направлении из ИП на блог-сервис. При таком способе работы не требуется хранить структуры данных в памяти и осуществлять их обработку. Так как один блог-процессор может обрабатывать запросы от нескольких клиентов, то при каждом запросе ему приходится получать из ИП данные авторизации пользователя, что увеличивает количество обращений к ИП. Вариантом улучшения представленной схемы работы является постоянное хранение в памяти агента некоторых пользовательских данных, содержащих данные авторизации и другие часто используемые данные.

Процесс функционирования блог-процессора схож с работой сервера. Блог-процессор ждёт получения нотификации от клиента, которая приходит благодаря механизму подписки, обрабатывает нотификацию и выполняет необходимые операции. Такое поведение пресуще агенту на протяжении всего времени его работы. Однако в качестве улучшения агента возможно также периодическое обновления некоторых блог-данных в ИП через определённый промежуток времени.

Одним из важных моментов реализации блог-процессора является организация подписки на триплеты нотификации. Возможны два способа подписки на нотификации:

1. одна подписка на все виды нотификаций
2. отдельная подписка на каждое требуемое действие

При использовании одной подписки на все виды нотификации блог-процессор подписывается на триплет вида:

$$\text{Notification}\langle\texttt{type}\rangle, \texttt{ANY}, \texttt{ANY}$$

где `type` — это тип сервиса с которым взаимодействует блог-процессор, а `ANY` — шаблон в триплете, обозначающий что вместо него может находиться любое значение. При таком виде подписки все нотификации, пришедшие блог-процессору, будут обрабатываться последовательно, что может привести к длительному ожиданию при одновременной работе нескольких агентов.

В случае использования отдельной подписки на каждую операцию, в агенте создаётся несколько обработчиков подписок на триплеты вида:

$$\text{Notification}\langle\texttt{type}\rangle, \langle\texttt{action}\rangle, \texttt{ANY}$$

где `action` — это название операции которую необходимо выполнить блог-процессору. В агенте создаётся такое количество подписок на указанные триплеты, сколько различных операций используется в модели нотификаций. Каждая подписка запускается в отдельном потоке, поэтому несколько разных операций могут выполняться параллельно. В связи с этим время ожидания клиентского агента до выполнения операции снижается.

Несмотря на это в ситуациях, когда важна последовательность выполнения операций из нотификаций, данные могут быть обработаны не в нужном порядке и результат, соответственно, будет некорректным. Кроме этого при параллельной обработки одних и тех же данных в связи с отсутствием критических секций, результат также может быть некорректным. Ещё одним недостатком является создание большого количества подписок. В текущей реализации SIB общее количество подписок является фиксированным и ограниченным. При превышении лимита подписок происходит ошибка и завершение работы серверов, управляющих ИП. В связи с этим общее количество подписок, используемых агентом, должно быть минимальным.

Итак при получении нотификации, блог-процессор, в зависимости от операции, выполняет соответствующее действие. Для выполнения действия агент использует параметр, являющийся объектом в триплете простой нотификации, либо запрашивает все параметры сложной нотификации, используя ссылку на индивида данной нотификации. Получив необходимые параметры, блог-процессор также запрашивает из ИП дополнительные данные пользователя, такие как логин, пароль, идентификатор поста на блог-сервисе и другие.

При организации получения данных из ИП агент блог-процессор может использовать два вида запросов: template-запрос и WQL-запрос.

При использовании template-запросов указывается триплет или набор

триплетов с наличием маски `ANY` вместо любой из составляющих триплета. Таким обазом будут получены все триплеты из ИП, удовлетворяющие заданному шаблону. Если для получения необходимых данных требуется пройти цепочку из нескольких триплетов, то для каждого результата template-запроса должен быть запущен следующий запрос и так далее, пока не будет достигнут конечный триплет цепочки.

Язык WQL позволяет организовывать более сложные запросы и получать данные за один запрос, вместо последовательности нескольких template-запросов. Особенностью WQL-запроса является то, что его результатом является массив строковых значений, а не массив триплетов. Для осуществления WQL запроса необходимо указать корневой идентификатор — субъект или объект триплета, от которого будет строиться запрос, и саму строку запроса. Строка запроса состоит из оператора и предиката или подзапроса. В WQL в основном используется два оператора: *seq* — строит последовательность выражений подзапросов или определяет объект триплета по субъекту и *inv* — определяет субъект триплета по объекту. Приведём пример использования WQL запроса. Пусть у нас есть набор триплетов: «profile-787ac22e, hasPersonInformation, person-787ac22e», «person-787ac22e, foaf:account, account-6cd6bb40». Для того чтобы определить по идентификатору аккаунта, профиль на котором данный аккаунт находится в качестве корневого узла указывается «account-6cd6bb40» и выполняется следующий запрос:

$$[seq, [inv, \text{foaf:account}], [inv, \text{scribo:personInformation}]]$$

таким образом в запросе осуществляется последовательность двух подзапросов с оператором *inv*, где происходит продвижение по онтологическому дереву вверх к профилю.

Итак, онтологическая модель пространства блоггера позволяет хранить данные блогосферы и осуществлять управление ими. Другой важной задачей является определение способов использования блоггинга в других интеллектуальных приложениях.

Глава 3

Применение модели персонального пространства в других интеллектуальных приложениях

Парадигма ИП обеспечивает коллективное использование ресурсов и контекстно-зависимых сервисов между распределёнными приложениями. Однако разные интеллектуальные приложения работают независимо друг от друга. Концепция ИП позволяет осуществлять интеграцию различных интеллектуальных приложений, тем самым добавлять дополнительные функциональные возможности одного приложения в другое. В настоящее время не существует стандартной схемы интеграции нескольких независимых приложений. В данной главе будут рассмотрены способы использования онтологии блог-клиента SmartScribo с другими интеллектуальными приложениями.

3.1 Блоггинг в интеллектуальной системе автоматизированного проведения конференций

Устройство системы проведения конференций

Интеллектуальная система автоматизированного проведения конференций (Smart Conference System, SC) — это приложение, осуществляющее помощь в процессе проведения конференции [17]. Система SC управляет визуальными данными, доступными для участников конференции: текущий слайд презентации и расписание конференции. Выступающий участ-

ник также имеет возможность переключать слайды при помощи мобильного устройства.

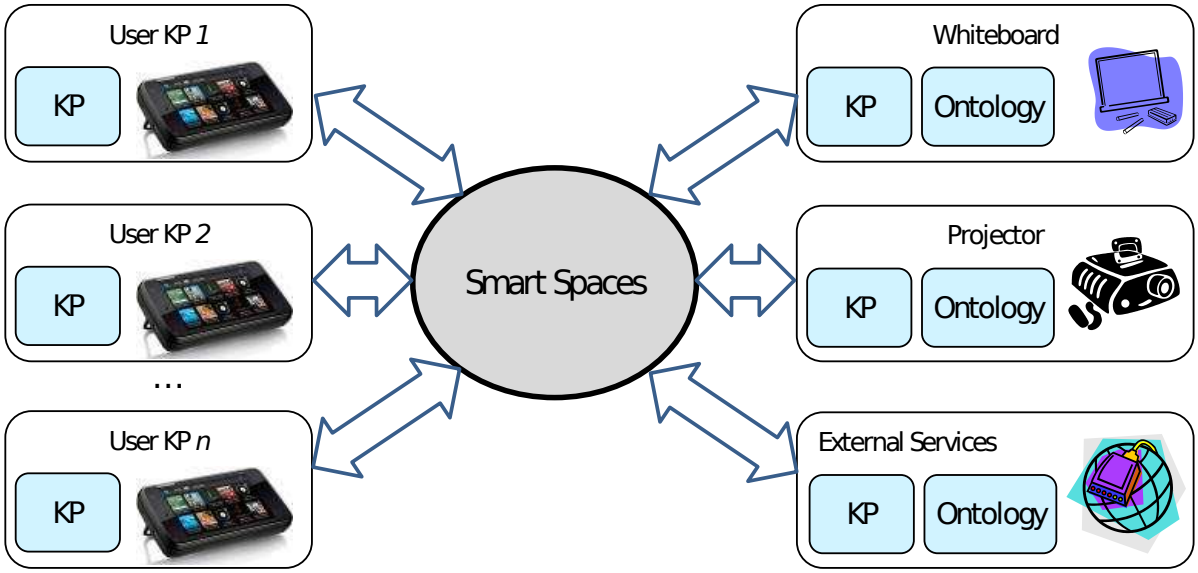

Рис. 3.1: Архитектура системы SC

Как и любое интеллектуальное приложение SC имеет распределённую архитектуру (рис. 3.1). Система включает в себя три вида агентов.

1. Проектор (projector). Агент-проектор отображает текущий слайд презентации на экране конференции через мультимедиа проектор. Данные агент загружает файл с презентацией выступающего при начале выступления, а также осуществляет переход между слайдами презентации по требованию пользователя.

2. Доска (whiteboard). Агент-whiteboard отображает расписание конференции на общем экране (через мультимедиа проектор), уведомляет выступающего о том, что время выступления подходит к концу, отменяет презентацию пользователя, если данный участник отсутствует. Кроме этого агент позволяет председателю конференции изменять порядок участников.

3. Клиент (SC-client). Клиентский агент запускается на мобильном устройстве каждого участника конференции. Клиент отображает расписание конференции, позволяет просмотреть текущий слайд презентации, персональную информацию каждого участника, а также информацию о презентациях, позволяет изменять данные в пользовательском профиле.

Выступающий участник имеет возможность управлять своей презентацией: осуществлять переход между слайдами.

Каждый из агентов системы SC использует общее пространство конференции. Онтологическое описание пространства конференции представлено на рис. 3.2. Оно состоит из трёх основных классов: менеджер пользовательской информации (user information manager), менеджер событий (event manager) и менеджер проектора (projector manager). Менеджер пользовательской информации хранит данные о профилях пользователей, видео выступающих, а также ссылку на текущий слайд. Кроме этого данный класс хранит информацию о расписании конференции и временных интервалах. Профиль пользователя содержит информацию о пользователе (имя, фото, электронный адрес, номер телефона, интересы, язык) и о презентации (имя презентации, ключевые слова и ссылка расположения презентации, для скачивания её другими агентами).

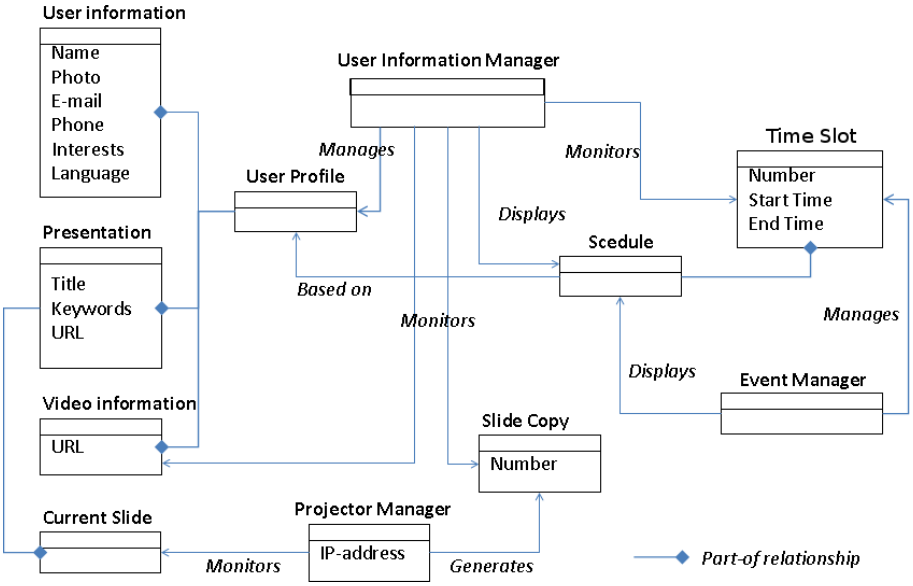

Рис. 3.2: Онтология SCS

Менеджер проектора создаёт скриншот и хранит ссылку на текущий слайд, который будет отображаться на экранах мобильных устройств. Менеджер событий содержит расписание конференции и управляет временными слотами. Расписание конференции создаётся в соответствии с профилями пользователей.

Итак система SC является удобным интеллектуальным приложением для проведения конференций. Несмотря на это функционал данной систе-

мы может быть расширен при помощи других сервисов в ИП, что позволит проводить конференцию на более высоком уровне.

Идея интегрирования функций SmartScribo в систему SC

Одним из таких дополнений может быть проведение дискуссий между участниками конференций. Помимо устных дискуссий после выступления расказчика, удобным также является проведение электронной онлайн дискуссии во время или после выступления. Часто при выступлении участника конференции у слушателя могут возникнуть вопросы, и в таком случае он может сразу опубликовать свой вопрос, а другие участники смогут его увидеть и, возможно, прокомментировать. Выступающий участник после выступления может зачитать и ответить на все вопросы. Кроме этого другие участники могут устроить дебаты на различные темы презентаций, в том числе и те, которые уже были рассказаны. Хорошим дополнением к этому является возможность клиента SC просмотреть полностью презентацию участника конференции и, таким образом, пользователь может вспомнить некоторые детали.

Нашей задачей являлась интеграция двух интеллектуальных приложений: SC и SmartScribo [8]. Внедрение возможностей блоггинга в конференцию позволило бы участникам вести нити дискуссий. Пост в блог конференции соответствует одному выступлению. Участники могут оставлять комментарии на пост (задавать вопросы) или отправлять комментарии на комментарии (отвечать на вопросы, дискутировать). Все участники могут просматривать и оставлять записи при помощи клиента SmartScribo со своих мобильных устройств. Блог-процессор SmartScribo будет синхронизировать данные дискуссии между ИП и блог-сервисом, поэтому пользователи также смогут принять участие в дискуссии, используя обычный веб-браузер. Таким образом ключевой задачей в интеграции двух приложений является координация между знаниями пространства конференции, блог-сервисом и пространством блогосферы.

Вследствие того, что не существует определённой схемы интеграции интеллектуальных приложений, для достижения этой цели можно при-

менять два подхода:

1. Построение общей онтологии.
2. Использование агента-посредника.

Первый подход основывается на построении общей онтологии, которая будет известна и будет использоваться всеми интегрируемыми интеллектуальными приложениями. В таком подходе оправдано использование стандартных онтологий, таких как FOAF для описания пользовательской информации. Таким образом онтология приложений будет представлять собой одно дерево зависимостей между классами знаний, а каждое приложение будет взаимодействовать с общим пространством, объединяющим онтологические данные из разных приложений.

Однако данный подход требует изначального знания общей онтологии разработчиками всех интегрируемых приложений, в противном случае готовое приложение необходимо будет модифицировать, что приведёт к дополнительным затратам. Кроме этого использование общей онтологии не решает задачу синхронизации данных, когда между знаниями разных приложений существует некоторая функциональная зависимость. В таком случае преобразования данных должны выполнятся каким-либо агентом, что приводит к добавлению дополнительной функциональности, не требуемой при автономной работе одного из приложений.

Второй подход основывается на использовании некоторого агента-посредника, которые знает онтологии (или фрагменты онтологий) интегрируемых интеллектуальных приложений. В данном подходе каждое приложение имеет своё собственное независимое ИП. Агент-посредник ведёт наблюдение за знаниями нескольких приложений и в определённые моменты осуществляет требуемые операции. Такой агент выполняет синхронизацию данных между разными онтологиями (при добавлении или изменении в одной из них), а также осуществляет преобразование данных при различиях в представлениях одних и тех же данных в разных онтологиях или при наличии функциональной зависимости между знаниями.

При осуществлении интеграции функций блоггинга в конференцию был выбран второй подход: использование агента-посредника. Общая схе-

ма интеграции представлена на рис. 3.3.

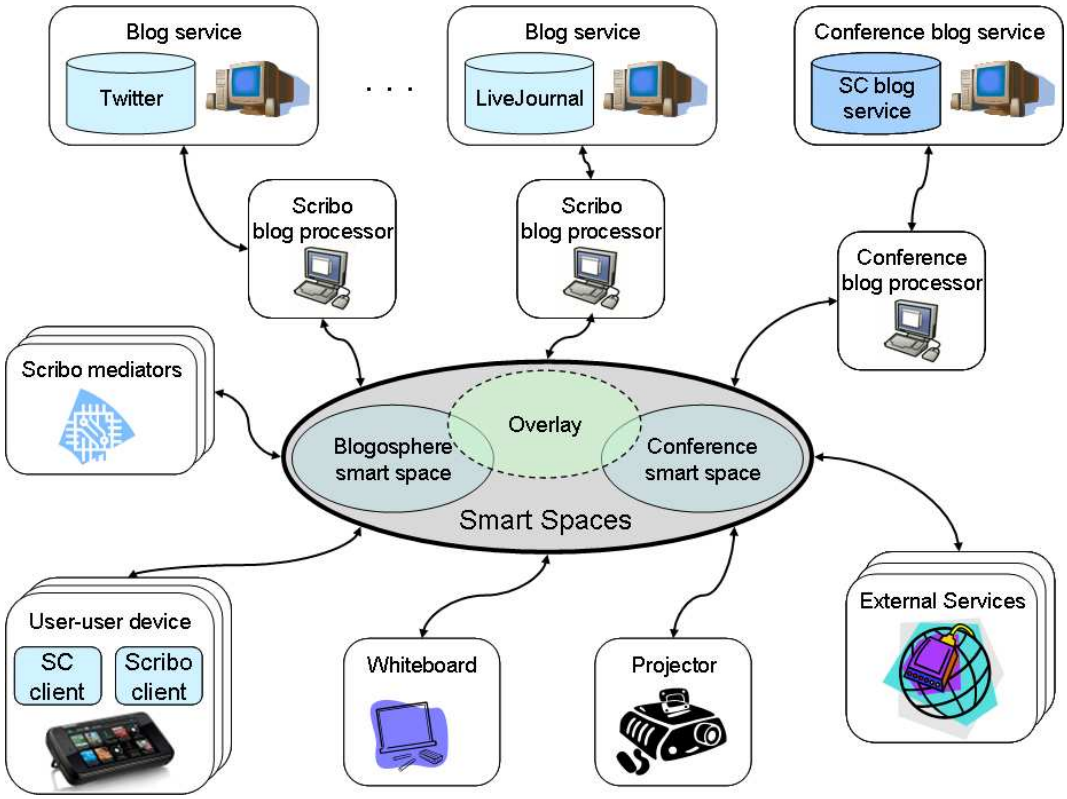

Рис. 3.3: Схема интеграции SCS и SmartScribo

Для проведенния блог-дискуссий запускается собственный сервер, использующий блог-машину LiveJournal. Блог является первичным хранилищем всех дискуссий, связанных с конференцией.

Для каждой конференции система SC создаёт отдельный блог. Блог состоит из постов, каждый из которых соответствует одному выступлению. Также некоторые посты могут быть посвящены другим темам, связанным с конференцией, например обсуждение организации. Таким образом SC проходит стадию инициализации и определяет структуру дискуссий.

Для связи программы конференции и дискуссий используется блог-процессор SmartScribo в качестве агента-посредника. Данный агент следит за расписанием в пространстве конференции. При добавлении или изменении элементов расписания, агент отражает эти модификации на блог-сервисе. Связь между агентом-посредником и пространством конференции является однонаправленной, т.е. агент получает данные из SC и преобразует расписание в структуру блога.

Взаимодействие агента-посредника с пространством блогосферы реа-

лизуется так же, как и в обычных блог-процессорах SmartScribo. В отличие от связи агента с пространством конференции, связь с пространством блогов является двунаправленной, так как блог-процессор публикует в нём посты и комментарии, а также получает из него комментарии других пользователей и синхронизирует эти данные с блог-сервисом. В целях снижения влияния других пользователей на блог конференции все посты (а возможно и комментарии) публикуются от имени SC (абстрактного пользователя с правами администратора). Все остальные пользователи не имеют права модифицировать структуру блога конференции. В текущей реализации предполагается, что существует также один общий аккаунт для всех участников конференции, в дополнение к аккаунту администратора. Используя общий аккаунт пользователи могут оставлять комментарии анонимно, либо подписываться вконце сообщения.

Для того, чтобы интегрирующий агент мог взаимодействовать с ИП различных приложений, он должен обладать знаниями о тех фрагментах онтологий, которые необходимы при выполнении сценариев интеграции. Следовательно, агент-посредник должен иметь некоторую оверлейную онтологию.

В общем случае оверлейная онтология — это онтология, имеющая представление о фрагментах двух или более других онтологий. Такая онтология концептуально или онтологически связывает онтологии различных интеллектуальных приложений. Оверлейная онтология содержит данные других онтологий, которые могут быть одинаковы (в таком случае выполняется синхронизация этих данных), могут быть схожи (тогда определяется семантическая близость и могут выполнятся некоторые операции преобразования данных) или могут зависеть друг от друга (тогда выполняются соответствующие действия, разрешающие эту функциональную зависимость).

Для определения измененииий в покрываемых онтологиях могут использоваться два способа: подписка на сами данные, которые входят в оверлейную онтологию (и получение в дальнейшем их изменений) или запрос этих данных в начале работы, если в будущем данные не будут меняться); либо использование модели нотификаций, которая кроме ин-

циирования действия у одного агента, позволяет сообщать также какие фрагменты онтологии (данных) были модифицированы. Однако об оповещении нотификацией должны заботиться те агенты, которые вносят изменения в данные. В случаях частого и массового изменения некоторых данных онтологии, выгоднее использовать нотификации, а не подписку на все эти данные.

Оверлейная онтология блог-процессора SC, содержащая фрагменты онтологии конференции и блогосферы, представлена на рис. 3.4. Таким образом агент-посредник может отслеживать и синхронизировать знания между двумя интеллектуальными приложениями. При работе с онтологическими данными блогосферы блог-процессор использует нотификации. Однако текущая версия SC не поддерживает модель нотификаций, а также не позволяет коренным образом изменять расписание во время конференции (возможно только изменение порядка выступающих). В связи с этим агент-посредник не использует подписку на данные выступлений, т.к. они не изменяются, а получает их и преобразует в блог-структуру в начале работы.

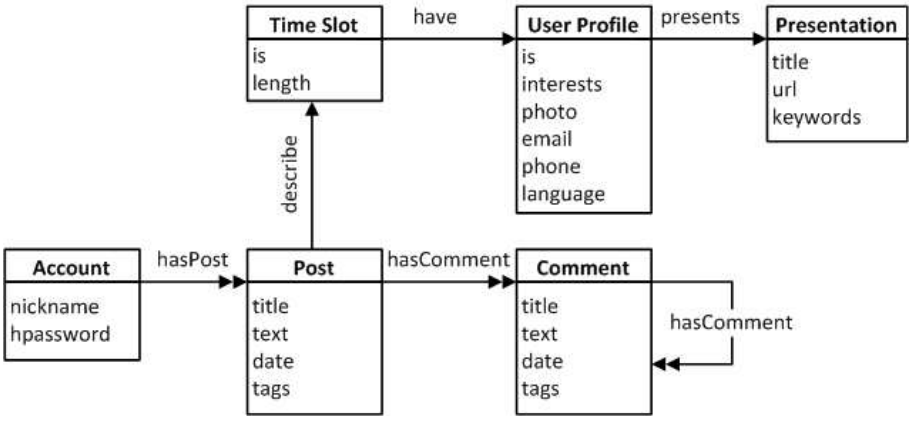

Рис. 3.4: Оверлейная онтология блог-процессора

В онтологиях конференции и блогосферы существует отношение один к одному между классами `Post` из SmartScribo и `TimeSlot` из SC. Свойство «describe» связывает эти классы. Данное свойство означает, что существует только один пост в блоге для каждого временного слота, соответствующего какому-либо выступлению. Такой пост содержит информацию о докладе, запланированном на указанное время.

3.2 Интеграция с приложением M3-Weather

Описание M3-Weather

Приложение SmartScribo может взаимодействовать и с другими интеллектуальными приложениями. В данном разделе будет рассмотрена возможность использования в блоггинге контекстных данных пользователя и текущем местоположении и погоде.

Проект M3-Weather [13] представляет собой прототип туристического приложения. Данное приложение при помощи глобальной системы позиционирования (Global Positioning System, GPS) определяет координаты текущего месторасположения и, используя их, отображает на экране пользовательского устройства прогноз погоды для текущей местности. Процесс получения информации о погоде состоит из следующих действий.

1. Определение текущих координат планшета с помощью GPS.
2. Определение ближайшего населенного пункта.
3. Определение и отображение прогноза погоды в текущей местности.
4. Периодическое самостоятельное обновление данных о погоде.

Согласно парадигме программирования ИП, приложение разбивается на агенты, каждый из которых будет выполнять некоторую отдельную функцию. Данные агенты в дальнейшем могут использоваться в других интеллектуальных программах.

Архитектура M3-weather представлена на рис. 3.5. Приложение может быть представлено в виде четырёх агентов, два из которых работают на мобильном устройстве, а другие два — на сервере. На устройстве запускается агент, взаимодействующий с GPS модулем, и агент, отображающий информацию о погоде на экране планшета. Следовательно на сервере должен работать агент, взаимодействующий с сервисом определения города по координатам, и агент, взаимодействующий с сервисом погоды. Перечислим функции, которые выполняет каждый из агентов.

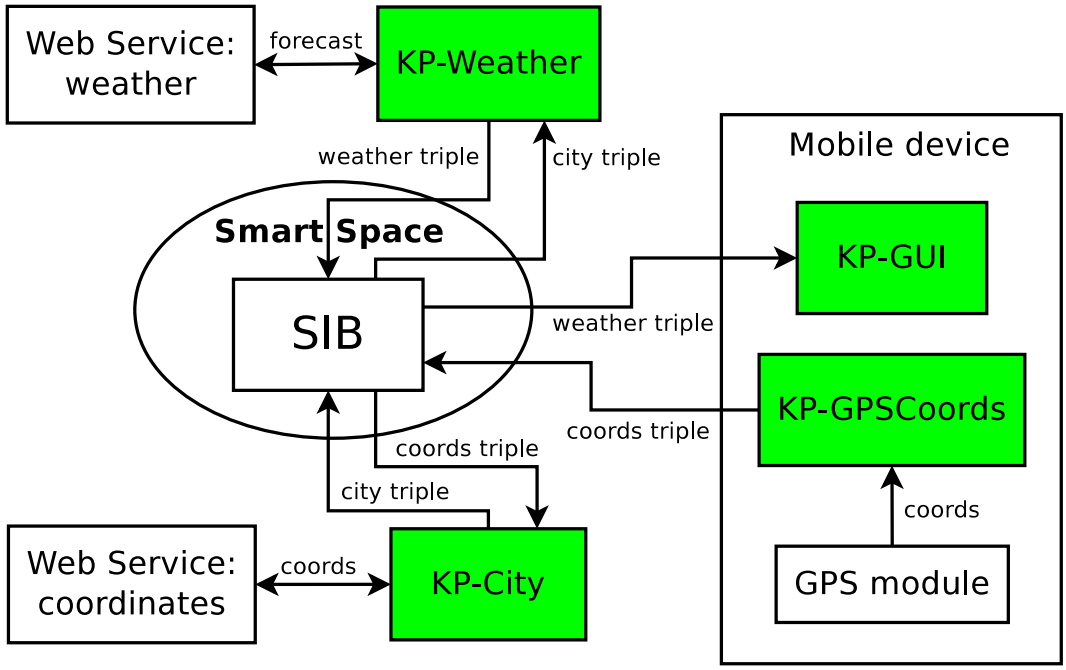

Рис. 3.5: Архитектура M3-Weather

1. Агент, взаимодействующий с GPS модулем (KP-GPSCoords)
   - получение с помощью GPS модуля координаты текущего месторасположения
   - периодическое обновление координат
   - сохранение полученных координат в SIB
2. Агент, взаимодействующий с веб-сервисом определения города по координатам (KP-City)
   - получение текущих координат месторасположения из SIB
   - отправка полученных координат веб-сервису и получение названия ближайшего города
   - сохранение полученных данных о городе в SIB
3. Агент, взаимодействующий с веб-сервисом прогноза погоды (KP-Weather)
   - получение текущего названия города из SIB
   - отправка названия города веб-сервису и получение информации о погоде в этом городе

- сохранение информации о погоде в SIB

4. Агент, взаимодействующий с экраном планшета (KP-GUI)

   - получение актуальных данных о городе и погоде из SIB
   - отображение на экране планшета информации о городе и погоде

Для того, чтобы агенты могли взаимодействовать между собой, необходим общий формат данных, который бы полностью описывал предметную область и позволял представлять информацию в ИП.

Онтология M3-Weather (рис. 3.6) представляет собой структурированное описание данных о координатах, городе и погоде. В онтологии выделяется три основных класса: координаты (`Coords`), город (`City`) и погода (`Weather`). Стрелка с надписью `type` указывает на тип сущности (в данном случае объекты являются классами). Каждый класс обладает свойствами, которые имеют тип `DatatypeProperty`. Стрелка с надписью `domain` обозначает, к какому объекту относится указанное свойство. Рассмотрим подробнее назначение каждого свойства.

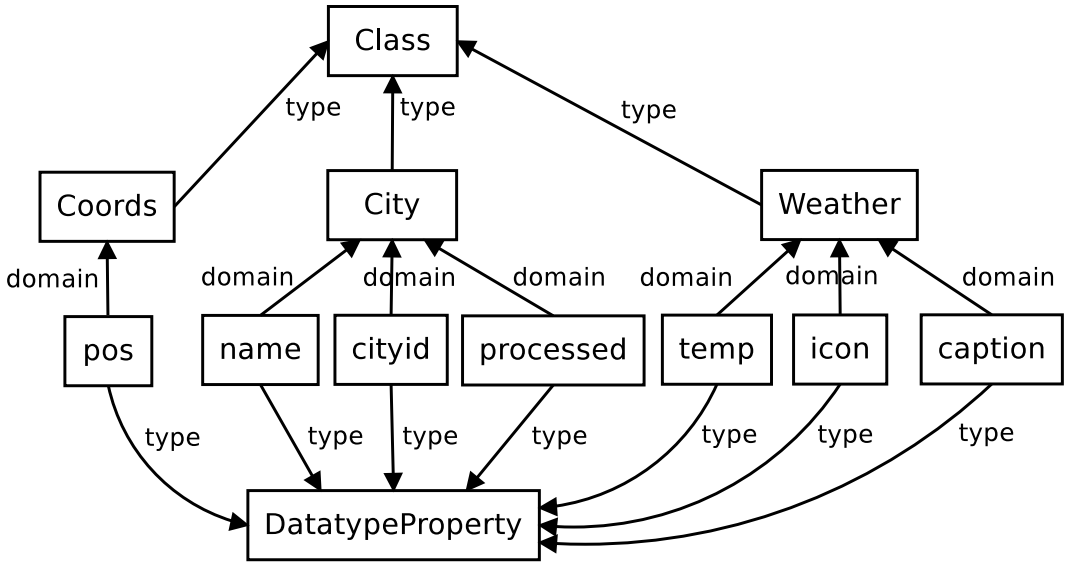

Рис. 3.6: Онтология M3-Weather

1. класс `Coords`

   обладает только одним свойством `pos`, которое хранит данные о текущем местоположении (широта и долгота)

2. класс `City`

   - свойство `name` хранит название ближайшего города

- свойство `cityid` предназначено для хранения идентификатора города, используемого для определения погоды на веб-сервисе
- свойство `processed` содержит информацию о том, был ли найден прогноз погоды для данного города (значение yes или no)

3. класс `Weather`

- свойство `temp` содержит температуру
- свойство `icon` содержит номер иконки, иллюстрирующей погоду
- свойство `caption` содержит краткое описание погоды

Для хранения конкретных данных в онтологии, необходимо создать индивида класса. Описание индивида представляет собой триплет вида "<имя_индивида>, type, <класс>". Для установки значения свойства индивида задаётся триплет вида "<имя_индивида>, <имя_свойства>, <значение>".

Онтология M3-Weather позволяет хранить многопользовательские данные. Чтобы представить данные нескольких пользователей, каждому пользовательскому устройству присваивается уникальный идентификатор. При создании индивидов каждого из классов онтологии к имени индивида добавляется данный идентификатор. Таким образом устройство сможет получить данные, предназначенные только для него.

Рассмотрим порядок взаимодействия между всеми агентами, а также взаимодействие с онтологическими данными. Напомним, что обмен данными между агентами осуществляется при помощи механизма публикации и подписки.

KP-GPSCoords получает данные о текущем месторасположении от GPS-модуля и записывает их в индивид класса `Coords` в SIB с некоторой периодичностью. При публикации данных генерируется триплет вида `coords<id> - pos - <lat>;<lon>`, где `<id>` — это уникальный идентификатор пользовательского устройства, `<lat>` — широта, а `<lon>` — долгота.

KP-City подписывается на индивидов класса `Coords` (триплеты, публикуемые KP-GPSCoords). При появлении в SIB новых координат или при изменении существующих, агент получает эти данные и отправляет

запрос с координатами веб-сервису геокодирования. Сервер возвращает список ближайших к данному местоположению городов. Из списка выбирается наиболее ближайший город и далее отправляется запрос с названием города веб-сервису погоды. Если город присутствует в базе данных веб-сервиса погоды, то возвращается его идентификатор. При отсутствии города в базе данных выбирается следующий ближайший город из списка, и процесс повторяется, пока не будет получен идентификатор.

После получения необходимых данных, KP-City публикует их в SIB, как свойства индивида `City`. Также к индивиду добавляется свойство `processed` со значением `no`. Это свойство показывает, что для данного индивида ещё не был определён прогноз погоды.

KP-Weather подписан на индивидов класса `City`, имеющих значение свойства `processed - no`. При появлении такого индивида, агент получает идентификатор города и отправляет запрос на веб-сервис погоды для получения прогноза. При получении данных о погоде, он публикует их в SIB, а также меняет у индивида `City` значение свойства `processed` на `yes`.

KP-GUI работает на устройстве и имеет графический интерфейс. Агент подписан на индивидов классов `City` и `Weather` и, при их создании или обновлении, отображает на экране устройства актуальные данные о текущем городе и погоде.

Использование данных M3-Weather в SmartScribo

Данные, получаемые приложением M3-Weather, можно использовать в других интеллектуальных приложениях. Например данные о текущих координатах, городе или погоде могут быть использованы в блог-клиенте SmartScribo при отправке поста в собственный блог во время путешествия.

Интеграцию данных M3-Weather в SmartScribo можно осуществить при помощи создания общей онтологии. Вследствие того, что информация о координатах, текущем городе и погоде является контекстом пользователя, а в онтологии SmartScribo в профиле пользователя существует класс `Context`, то достаточно добавить в этом классе свойства, связывающие контекстную информацию блоггера с данными M3-Weather.

Данные, публикуемые M3-Weather, представляют собой набор триплетов, субъекты которых соответствуют трём классам знаний: координаты (coords<id>), город (city<id>), погода (weather<id>). Все три индивида содержат в своём названии одинаковый идентификатор `id`, который является уникальным идентификатором пользовательского устройства в ИП. Таким же идентификатором может обладать профиль пользователя в онтологии SmartScribo. Таким образом агенты M3-Weather могут записывать свои данные в персональное пространство блоггера.

Объединение онтологий указанных интеллектуальных приложений достигается путём добавления связей между контекстом (индивидом контекста у профиля пользователя) и индивидами M3-Weather. Пусть свойство «curCoords» связывает контекст с координатами, «curCity» – с текущим городом и «curWeather» с текущей погодой. Триплеты которые публикуют агенты M3-Weather имеют вид:

coords-<id>, pos, <coordinates>
city-<id>, name, <cityname>
weather-<id>, temp, <temperature>
...

где <coordinates>, <cityname>, <temperature> — конкретные значения соответствующих свойств. Для связки контекстных данных блоггера с данными M3-Weather, агенты обращаются к профилю блоггера `profile-<id>`, через свойство «hasContext» получают доступ к индивиду контекста и публикуют следующие триплеты:

context-<id>, curCoords, coords-<id>
context-<id>, curCity, city-<id>
context-<id>, curWeather, weather-<id>

Таким образом данные о координатах, городе и погоде становятся контекстом блоггера. Агенты SmartScribo могут получать эти данные через индивида класса `Context`. Такая информация может быть автоматически вставлена в сообщение при написании поста во время путешествия.

Заключение

В ходе выполнения работы были достигнуты следующие результаты:

1. Описана структура персонального пространства пользователя и классы знаний, содержащиеся в нём.
2. На основе структуры персонального пространства построена онтологическая модель блогосферы, используемая в приложении SmartScribo, которая позволяет описывать персональные и контекстные данные пользователя, а также данные блогосферы.
3. Изучена спецификация FOAF на основе которой базируется онтология SmartScribo.
4. Разработан механизм нотификаций, позволяющий координировать работу агентов приложения и управлять онтологическими данными.
5. Реализован блог-процессор взаимодействующий с блог-сервисом LiveJournal.
6. Предложена схема интеграции функций блоггинга SmartScribo в систему SC, что позволяет проводить обсуждение выступлений на конференции.
7. Предложен способ использования данных интеллектуального туристического приложения M3-Weather в SmartScribo.

Результаты работы опубликованы в статьях [8, 15, 16].

В дальнейшем планируется доработка блог-процессора, а также разработка агентов-медиаторов, выполняющих обработку онтологических данных и предоставляющих расширенные возможности блоггинга для пользователя. Для реализации агентов-медиаторов потребуется расширение модели нотификаций и добавление в неё проактивных нотификаций, используемых в интеллектуальных сценариях блоггинга.

Библиографический список использованной литературы

1. Baldoni R. The evolution of publish/subscribe communication systems / R. Baldoni, M. Contenti, A. Virgillito // Future Directions in Distributed Computing, ser. Lecture Notes in Computer Science. — Springer, 2003. — №2584. — С. 137–141.

2. Benson E. Talking about data: sharing richly structured information through blogs and wikis / E. Benson, A. Marcus, F. Howahl, D. Karger // Proc. 9th Int'l Conf. on The Semantic Web, ser. ISWC'10. Springer-Verlag. — 2010. — С. 48–63.

3. Berners-Lee T. Scientific publishing on the semantic web / T. Berners-Lee, J. Hendler // Nature. — 2001. — №410. — С. 1023–1024.

4. Främling K. Smart Spaces for Ubiquitously Smart Buildings / K. Främling, I. Oliver, J. Honkola, Jan Nyman // Proc. 3rd Int'l Conf. Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM 2009). — 2009. — С. 295–300.

5. Guarino N. The Ontological Level: Revisiting 30 Years of Knowledge Representation / N. Guarino // Conceptual modeling: Foundations and applications. — Springer-Verlag, 2009. — С. 52–67.

6. Honkola J. Smart-M3 information sharing platform / J. Honkola, H. Laine, R. Brown, O. Tyrkkö // The 1st Int'l Workshop on Semantic Interoperability for Smart Spaces (SISS 2010) in conjunction with IEEE ISCC 2010. — IEEE Computer Society, 2010. — С. 295–300.

7. Karger D. R. What would it mean to blog on the semantic web? / D. R. Karger, D. Quan // Web Semant. — 2005. — №3. — С. 147–157.

8. Korzun D. Blogging in the Smart Conference System / D. Korzun, I. Galov, A. Kashevnik, N. Shilov, K. Krinkin, Y. Korolev // Proc. 9th Conf. of Open Innovations Framework Program FRUCT. — 2011. — С. 63–73.

9. Korzun D. Generating Modest High-Level Ontology Libraries for Smart-M3 / D. Korzun, A. Lomov, P. Vanag, J. Honkola, S. Balandin // Proc. 4th Int'l Conf. Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM 2010). — 2010. — С. 103–109.

10. Oliver I. Context gathering in meetings: Business processes meet the Agents and the Semantic Web / I. Oliver, E. Nuutila, S. Törmä // The 4th Int'l Workshop on Technologies for Context-Aware Business Process Management (TCoB 2009) within Proc. Joint Workshop on Advanced Technologies and Techniques for Enterprise Information Systems. — 2009.

11. Oliver I. Dynamic, localised space based semantic webs / I. Oliver, J. Honkola, J. Ziegler // Proc. IADIS Int'l Conf. WWW/Internet 2008. — IADIS Press, 2008. — С. 426–431.

12. Resource Description Framework (RDF): Concepts and Abstract Syntax [Электронный ресурс]. Режим доступа: http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/

13. Samoryadova A. M3-Weather: A Smart-M3 World-Weather Application for Mobile Users / A. Samoryadova, I. Galov, P. Borovinskiy, K. Kulakov, D. Korzun // Proc. 8th Conf. of Open Innovations Framework Program FRUCT. — 2010. — С. 25–35.

14. The friend of a friend (foaf) project [Электронный ресурс]. Режим доступа: http://www.foaf-project.org/

15. Zaiceva D. A Blogging Application for Smart Spaces / D. Zaiceva, I. Galov, A. Sannikov, D. Korzun // Proc. 9th Conf. of Open Innovations Framework Program FRUCT. — 2011. — С. 154–163.

16. Zaiceva D. Mobile multi-blogging in Smart-M3: Architecture and scenarios / D. Zaiceva, I. Galov, A. Sannikov, D. Korzun // Proc. 8th Conf. of Open Innovations Framework Program FRUCT. — 2010. — С. 225–233.

17. Кашевник А. М. Интеллектуальная система автоматизированного проведения конференций / А. М. Кашевник, Ю. Вальченко, М. М. Ситаев, Н. Г. Шилов // Тр. СПИИРАН. — 2010. — №14. — С. 228–243.