ФЕДЕРАЛЬНОЕ АГЕНСТВО ПО ОБРАЗОВАНИЮ
ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
ПЕТРОЗАВОДСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

На правах рукописи

Ломов Александр Андреевич

МОДЕЛИ И МЕХАНИЗМЫ ДЛЯ АВТОМАТИЗАЦИИ ПРОГРАММИРОВАНИЯ КОСВЕННОГО ВЗАИМОДЕЙСТВИЯ АГЕНТОВ ИНТЕЛЛЕКТУАЛЬНЫХ ПРОСТРАНСТВ

05.13.11 — Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей

Диссертация на соискание ученой степени
кандидата технических наук

Научный руководитель:
кандидат физико-математических наук,
доцент Корзун Дмитрий Жоржевич

Петрозаводск — 2014

Оглавление

Перечень сокращений и условных обозначений

ИП — интеллектуальное пространство.

ОО — объектно-ориентированный.

ОС — операционная система.

ПО — программное обеспечение.

Процессор знаний (KP от англ. Knowledge Processor) — программный агент для интеллектуального пространства на платформе Smart-M3.

Свойство-данные — отношения между экземплярами классов и RDF-литералами или типами данных, определяемых XML Schema.

Свойство-объект — отношения между экземплярами двух классов.

Семантический веб — направление развития Всемирной паутины, целью которого является представление информации в виде, пригодном для эффективной машинной обработки.

Т-шаблон — RDF-тройка, где компоненты заменены маской.

.NET — программная платформа корпорации Microsoft для создания приложения различных типов, на различных языках программирования.

ANSI C — стандарт языка программирования C, предложенный Американским национальным институтом стандартов.

C# — объектно-ориентированный язык программирования.

CodeGen — генератор программного кода онтологической библиотеки программного инструмента SmartSlog.

IoT (Internet of Things) — концепция “Интернет физических устройств”.

Java — ряд программных продуктов и спецификаций, предоставляющих систему для разработки прикладного программного обеспечения.

KP-интерфейс (Knowledge Processor Interface) — реализует клиентскую часть протокола SSAP для взаимодействия агента KP с SIB.

OSGi (Open Services Gateway Initiative) — спецификация динамической модульной архитектуры для Java-приложений.

OWL (Web Ontology Language) — язык описания онтологий для Семантического веб.

POSIX (Portable Operating System Interface for Unix) — набор стандартов, описывающих интерфейсы между прикладной программой и операционной системой.

RDF (Resource Description Framework) — модель Семантического веб для представления данных и метаданных.

RDF-тройка — утверждение о ресурсе, состоящее из трёх компонентов, имеет вид: “субъект - предикат - объект”.

RPC (Remote Procedure Call) — вызов удалённых процедур.

SIB (Semantic Information Broker) — семантический информационный брокер.

Smart-M3 — платформа с открытым исходным кодом, реализующая концепцию интеллектуальных пространств с косвенным взаимодействием агентов в IoT средах.

SmartSlog — программный инструмент разработки агентов на основе онтологий проблемной области для интеллектуальных пространств на платформе Smart-M3.

SPARQL — язык запросов к данным, представленным на основе модели RDF.

SSAP (Smart Space Access Protocol) — протокол доступа к интеллектуальному пространству на платформе Smart-M3.

URI (Uniform Resource Identifier) — унифицированный идентификатор ресурсов.

UUID (Universally Unique Identifier) — уникальный универсальный идентификатор.

XML Schema — язык описания структуры XML-документа.

Введение

Развитие технологий Интернета физических устройств (“Интернет вещей”, IoT) [54] и сервисно-ориентированных систем для вычислительных сред “повсеместных вычислений” (от англ. ubiquitous computing) [30] позволяет применять на практике парадигму интеллектуальных пространств. Интеллектуальное пространство (ИП) обеспечивает динамическое множество участников вычислительной среды контекстно-зависимой информацией, сервисами и персонифицированными рекомендациям [4, 30, 71, 51]. Участниками выступают автономные агенты на разнообразных устройствах вычислительной среды. Агент — это программа, прикладная логика которой основана на 1) наблюдении состояния среды и происходящих изменений, 2) интерпретации наблюдений для определения собственных действий и 3) внесении своих изменений в среду. Интеллектуальные пространства различаются по типу вычислительной среды, способу организации взаимодействия агентов и представлению разделяемого агентами информационного содержимого.

Широкое применение получили интеллектуальные пространства, разворачиваемые в локализованных физических окружениях (напр., в доме или офисе — системы Z-Wave, KNX, C-Bus). В случае открытых ИП состав участников, представленных агентами на разнообразных вычислительных устройствах, динамически меняется (напр., ИП в общественных местах — конференц-зал, зона отдыха и т.п.). Возникает проблема интероперабель-

ности (возможности взаимодействия) участников, включая вопросы как сетевого взаимодействия агентов, так и разделения ими информационного содержимого.

Выделяют интеллектуальные пространства с косвенным взаимодействием агентов на основе онтолого-ориентированного представления и обработки разделяемого агентами информационного содержимого. Такие ИП характеризуются следующими свойствами. Вычислительная среда удовлетворяет условиям IoT — разнообразные устройства локализованы в физическом окружении и могут выполнять сетевые коммуникации как друг с другом, так и с внешними системами. Взаимодействие агентов — косвенное, на основе базовых моделей “классная доска” [8, 70] и “публикация/подписка” [64, 21]. Разработчик использует примитивы доступа агента к ИП для реализации косвенного взаимодействия, определяя обработку разделяемого информационного содержимого ИП. Последнее представлено в общем хранилище на основе модели RDF (Resource Description Framework) [20, 28] и интерпретируется агентами при помощи онтологий проблемной области на языке OWL (Web Ontology Language) [76, 77]. Модель RDF и OWL-онтологии наследуются из Семантического веб [25, 16]. Выделяют основные типы косвенного взаимодействия: интеллектуальное соединение (ожидание некоторых событий для начала действий), централизованное хранилище (доступ к хранилищам информации), концентратор информации (получение и интеграция информации из распределенных источников) и глобальный посредник (организация обмена информацией между агентами).

Для программирования взаимодействия в логике агента в терминах

OWL-онтологий (классы, свойства, индивиды) необходимо их представление в программном коде и наборы функции для работы с этими представлениями. Создание такого представления возможно на основе модельно-ориентированного подхода [84], а функции могут быть реализованы на уровне промежуточного ПО. Онтологическая модель проблемной области определяет информационные объекты (далее — объекты), посредством которых агент взаимодействует с другими агентами. Промежуточное ПО позволяет связать такие объекты с конструкциями языка программирования и представить объекты в программном коде логики агента структурами данных [10, 11, 53, 74]. На уровне промежуточного ПО реализуются механизмы для автоматизации программирования косвенного взаимодействия агентов с использованием объектов онтологической модели. Разработчику для использования механизмов предоставляются шаблоны программного кода. Основную сложность для построения таких механизмов создает наблюдаемое разнообразие аппаратно-программных платформ в IoT-средах.

Целью работы является повышение эффективности разработки агентов в терминах проблемной области за счет автоматизации программирования косвенного взаимодействия агентов на основе технологий Семантического веб и с учетом разнообразия аппаратно-программных вычислительных платформ.

В ходе исследования были поставлены и решены следующие **задачи**.

1. Анализ существующих методов разработки программных агентов и основных типов их взаимодействия для определения путей упрощения разработки за счет применения модельно-ориентированного подхода в условиях IoT-сред.

2. Разработка метода программирования косвенного взаимодействия агентов с использованием объектов онтологической модели проблемной области и механизмов программирования для упрощения реализации основных типов взаимодействия.

3. Разработка специализированных моделей косвенного взаимодействия агентов на основе базовых моделей для автоматической интеграции агентом объектов из ИП, автоматической синхронизации агентом объектов и локальной обработки агентом групп объектов с автоматическим построением запросов к ИП в виде транзакций.

4. Разработка механизмов программирования для использования предлагаемых моделей взаимодействия и автоматизации программирования логики агентов при реализации основных типов взаимодействия агентов.

5. Реализация и апробация программного инструмента для применения предлагаемых метода и механизмов программирования взаимодействия агентов с учетом разнообразия аппаратно-программных вычислительных платформ.

Объектом исследования являются программные агенты интеллектуального пространства, применяющие 1) базовые модели “классная доска”, “публикация/подписка” для взаимодействия агентов и 2) RDF-модель и OWL-онтологии для представления и обработки разделенного информационного содержимого.

Предметом исследования являются специализированные модели косвенного взаимодействия и механизмы программирования для основных типов взаимодействия агентов, обеспечивающие 1) автоматизирован-

ную разработку программного кода логики агента с учетом разнообразия аппаратно-программных вычислительных платформ, 2) выполнение агентом интеграции, синхронизации и обработки информационного содержимого при его взаимодействии в ИП на уровне объектов онтологической модели проблемной области.

Используемые методы. Результаты выполненных в работе исследований и проектных работ основаны на методах системного программирования, объектно-ориентированном подходе к проектированию и программированию, автоматизированных системах онтологического моделирования, моделях взаимодействия и протоколах распределенных систем, технологиях Семантического веб.

Научная новизна и **положения, выносимые на защиту**.

1. Метод программирования косвенного взаимодействия агентов на основе онтологических библиотек, отличающийся от существующих методов поддержкой разнообразных аппаратно-программных вычислительных платформ для выполнения агентов и позволяющий автоматизировать программирование основных типов взаимодействия, реализуемых в логике агента.

2. Модель взаимодействия агентов на основе многоэлементной сессии, отличающаяся от существующих моделей поддержкой параллельных сеансов сетевого доступа с автоматической интеграцией информации в локальном хранилище агента и позволяющая организовывать и управлять группами подписок в рамках сеансов для упрощения программирования сетевых операций доступа к нескольким ИП.

3. Модель взаимодействия агентов на основе операции подписки, от-

личающаяся от существующих моделей поддержкой отслеживания изменений в ИП на уровне объектов онтологической модели с автоматической синхронизацией этих объектов в локальном хранилище агента и позволяющая уменьшить в логики агента объем программируемых сетевых операций доступа к ИП.

4. Модель взаимодействия агентов на основе локальной обработки группы объектов, отличающаяся от существующих моделей возможностью программировать операции над группой объектов онтологической модели и позволяющая автоматически формировать агенту запрос к ИП на множественное изменение в виде одной транзакции и поддерживать семантическую целостность информационного содержимого.

5. Механизмы программирования косвенного взаимодействия для логики агента в созданном программном инструменте разработки SmartSlog платформы Smart-M3 для применения предложенного метода, отличающиеся от существующих механизмов автоматизацией программирования основных типов взаимодействия в ИП на уровне объектов онтологической модели и поддерживающие разработку агентов для разнообразных вычислительных устройств.

Практическая ценность. Предложенные специализированные модели взаимодействия агентов используются в полученном методе программирования косвенного взаимодействия агентов. Механизмы программирования реализованы автором в инструменте разработки с открытым исходным кодом SmartSlog платформы Smart-M3. Инструмент позволяет а) повысить качество разработки программного кода за счет связывания программных конструкций с онтологической моделью проблем-

ной области, б) снизить трудозатраты на разработку и сопровождение программных агентов и в) разрабатывать агентов для различных аппаратно-программных вычислительных платформ. В частности, инструмент SmartSlog применяется при разработке системы интеллектуального зала SmartRoom, автоматизирующей проведение мероприятий коллективной деятельности (конференции, собрания и др.).

Диссертационные исследования проводились в рамках реализации комплекса мероприятий Программы стратегического развития ПетрГУ на 2012-2016 г.г., были поддержаны грантом КА179 «Комплексное развитие регионального сотрудничества в области открытых инноваций в ИКТ» в рамках Karelia ENPI — совместной программы Европейского союза, Российской Федерации и Республики Финляндия, получили финансовую поддержку со стороны Минобрнауки России в рамках базовой части государственного задания №2014/154 в сфере научной деятельности, НИР № 1481.

Апробация работы. Основные положения и результаты диссертационной работы представлялись на международных конференциях Open Innovations Association FRUCT: 2011 г. (FRUCT 9, FRUCT 10), 2012 г. (FRUCT 12) и 2013 г. (FRUCT 13), семинаре «Проблемы современных информационно-вычислительных систем» в МГУ им М. В. Ломоносова, 2010 г.; международной конференции Annual International Workshop on Advances in Methods of Information and Communication Technology (AMICT), ПетрГУ, 2010 г.; международной конференции Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM), 2010 г.; конгрессе Information Systems and Technologies (IS&IT'11), 2011 г.; научных семинарах СПИИРАН, 2013-2014 гг.; научном семинаре ИИММ КНЦ РАН,

2013 г.; научном семинаре кафедры системного программирования СПбГУ, 2014 г.; научном семинаре ИПМИ КарНЦ РАН, 2014 г.

Публикации. Основные результаты по материалам диссертационной работы опубликованы в 10 печатных работах, среди них 3 работы из перечня ВАК, 1 работа индексируется в базе данных Scopus. Получено свидетельство о регистрации программы ЭВМ (выдано Федеральной службой по интеллектуальной собственности).

Структура и объем диссертации. Диссертация объемом 137 машинописных страниц состоит из введения, 4 глав, заключения, списка использованной литературы и приложения. Текст диссертации содержит 29 рисунков и 8 таблиц. Список использованной литературы содержит 98 наименований.

Во введении обосновывается актуальность работы и формулируется ее цель, приводятся основные положения, выносимые на защиту, отмечается их научная новизна, практическая значимость и приведено содержание работы по главам.

В первой главе предлагается общая характеристика решаемых в работе задач. Выполнен обзор парадигмы интеллектуальных пространств и технологий Семантического веб. Рассматриваются особенности и основные типы косвенного взаимодействия агентов в ИП. Представлены два уровня разработки логики агента при программировании косвенного взаимодействия и особенности использования промежуточного ПО, обеспечивающего программирование в терминах проблемной области. Приводятся базовые задачи программирования логики агента, возникающие при реализации основных типов взаимодействия, и варианты автоматизации решений этих

задач на уровне промежуточного ПО.

Во второй главе предлагается метод программирования косвенного взаимодействия агентов с применением программных “онтологических” библиотек в качестве промежуточного ПО. Определены шаги метода и схема использования онтологической библиотеки для разработки агента. Рассматривается емкостная и временна́я трудоемкость онтологической библиотеки. Предлагаются специализированные модели косвенного взаимодействия на основе многоэлементной сетевой сессии, операции подписки для отслеживания изменения объектов проблемной области и локальной обработки группы объектов. Показано, как применение таких моделей позволяет автоматизировать решение базовых задач программирования взаимодействия в логике агента.

В третьей главе представлены механизмы программирования косвенного взаимодействия в логике агента на основе предложенных метода и специализированных моделей взаимодействия. Механизм генерации программного кода онтологической библиотеки определяет построение частной онтологической модели на основе выбора объектов из онтологий проблемной области для генерации программного кода. Механизм программирования взаимодействия на основе многоэлементной сессии включает в себя схему использования сессии и операций управления для переключения в программном коде состояний (есть/нет подключение к ИП) сетевых сеансов и подписок. Механизм программирования взаимодействия на основе операции подписки для отслеживания изменений в ИП на уровне объектов проблемной области определяет, какие объекты могут быть заданы для отслеживания изменений и формат RDF-представлений для подписываемых

объектов и получаемых уведомлений. Механизм программирования взаимодействия на основе локальной обработки группы объектов реализует дополнительное свойство для объекта онтологической модели, в котором происходит обработка объектов и автоматическое формирование агентом запросов к ИП виде транзакции, обеспечивающих семантическую целостности информационного содержимого ИП.

В четвертой главе рассматривается платформа Smart-M3 для развертывания ИП и возможности ее использования совместно с известными агентскими платформами. Для платформы Smart-M3 представлена апробация разработанных механизмов. Последние реализованы автором в программном инструменте SmartSlog, который поддерживает предложенный метод разработки и позволяет генерировать онтологические библиотеки. Исследуется вопрос эффективности использования инструмента SmartSlog при разработке агентов для различных аппаратно-программных платформ. Практическая эффективность показана на примере разработки агентов системы интеллектуального зала SmartRoom для ООО “ОптиСофт”. Переносной вариант системы SmartRoom используется для проведения секций международных конференций Ассоциации открытых инноваций FRUCT.

Автор выражает искреннюю признательность своему научному руководителю доценту Д. Ж. Корзуну, сформулировавшему направление исследований и оказавшему сильное влияние на структуру и стиль изложения этой работы, которая не состоялась бы без его постоянного внимания. Автор благодарен профессорам А. В. Смирнову, В. И. Городецкому и ведущему научному сотруднику А. М. Кашевнику за критические замечания в

ходе работы над диссертацией. Автор также признателен своим оппонентам — профессору А. Н. Терехову и доценту М. Г. Пантелееву.

1. Особенности разработки интеллектуальных пространств

В главе приводится общая характеристика решаемых в работе задач. Рассматриваются интеллектуальные пространства и технологии Семантического веб, ориентированные на разработку сервисно-ориентированных систем для условий IoT-сред. Рассмотрены характеристические свойства интеллектуальных пространств. Отмечаются особенности исследуемых в работе интеллектуальных пространств и представлены два уровня разработки агентов для таких пространств. Определяется роль промежуточного ПО, используемого при разработке агентов. Такое ПО реализует механизмы программирования косвенного взаимодействия агентов и позволяет автоматизировать решение базовых задач программирования взаимодействия в логике агента на уровне объектов проблемной области.

1.1. Интеллектуальные пространства

Концепция повсеместных вычислений декларирует создание систем, которые естественным и незаметным для человека образом автоматизируют решение задач повседневной жизни, предугадывая потребности пользователя и адаптируясь под текущую ситуацию [97, 4, 14]. В такой системе используется множество информационных технологий, поддерживающих вычислительные и коммуникационные функции для различных устройств

в окружении человека. В рамках развития концепции повсеместных вычислений появились такие понятия как “smart environment” (умное окружение) и “smart objects” (умные объекты, вещи) [35], “intelligent space” [48, 30] или “smart space” (интеллектуальное пространство) [4, 31, 54].

Умной вещью может быть кресло, определяющие занято ли оно или кровать, определяющая положение тела и состояние человека (спит или нет). Умные вещи также используются в IoT-средах [45]. В работе [2] определяется общий случай IoT-среды, где “умное окружение” с “умными вещами” представляет собой распределенную информационную систему, состоящую из а) сенсорной беспроводной сети, б) сервера для централизованного сбора информации и управления, в) сети “умных объектов”, оснащенных локальными встроенными интеллектуальными системами, г) набора мобильных устройств, обеспечивающих пользовательский интерфейс для доступа к системе и сервисам.

Интеллектуальным пространством (ИП) называют вычислительную среду, которая обеспечивает находящееся в ней динамическое множество участников контекстно-зависимой информацией, разнообразными сервисами и персонифицированными рекомендациям в понятной и ненавязчивой форме и по возможности в упреждающей манере [4, 30, 71, 51]. Необходимо отметить, что обычно ИП локализуется в рамках ограниченного физического пространства (напр., улица, дом, комната). Участниками выступают автономные агенты на разнообразных устройствах вычислительной среды. Следуя работе [51], а также работе [15], в которой приведен анализ “слабых” и “сильных” определений агента и представлены основные свойства агента (восприятие, интерпретация, действие), будем далее использовать

общий термин агент, включая в него и умные объекты, и сервисы.

В настоящее время общепринятой архитектуры ИП не существует. Один из возможных вариантов ИП рассматривается в [4]. Разнообразие участвующих вычислительных устройств, коммуникационных протоколов и источников информации требует разработки новых моделей взаимодействия агентов. Такие модели позволят системе эффективно решать задачи, связанные с динамическим участием агентов в ИП, совместным формированием информационного содержимого ИП, определением и прогнозированием контекста, поиском и построением нужных сервисов и др. [4, 17, 6, 2, 83, 31, 23].

Характеристические свойства и возможные подходы к их реализации показаны на рис. 1.1. Свойствами ИП выступают: вычислительная среда, модель сетевого взаимодействия и способ организации разделяемого информационного содержимого.

<table>
<tr><th>Свойство</th><th colspan="4">Подход / Модель</th></tr>
<tr><td>Организация информационного содержимого</td><td colspan="2">Индивидуальные хранилища участников</td><td colspan="2">Общее информационное хранилище</td></tr>
<tr><td rowspan="2">Сетевое взаимодействие</td><td>Обмен сообщениями</td><td>Публикация/ подписка</td><td>Классная доска</td><td></td></tr>
<tr><td colspan="2">Прямое</td><td colspan="2">Косвенное</td></tr>
<tr><td rowspan="2">Вычислительная среда</td><td colspan="2">Глобально распределенная</td><td>Локали зованная</td><td>Локальная</td></tr>
<tr><td>Вирту альная</td><td>Физи ческая</td><td colspan="2">Социально-ориентированная</td></tr>
</table>

Рис. 1.1. Характеристические свойства интеллектуальных пространств.

В монографии [80] определены три типа вычислительной среды, на основе которой может разворачиваться ИП.

1. Виртуальная вычислительная среда, поддерживающая постоянный доступ агентов к необходимым сервисам (напр., к Интернет-сервисам). Программные агенты, взаимодействующие в среде, не зависят от ограничений физического окружения, так как существуют только в виртуальной среде. Агенты могут действовать как локально, так и в глобальном масштабе (удаленно) при взаимодействии с другими агентами.

2. Физическая среда (среда физических устройств) для работы агентов, которые представляют объекты реального мира в виртуальной среде. Агенты работают на устройствах различных типов, включая датчики и контроллеры (напр., система охраны для определения незаконного проникновения в помещение). Такие агенты локализованы в физическом пространстве и могут взаимодействовать как между собой, так и с внешними системами.

3. Социальная среда, где люди образуют среду для взаимодействия через агентов. Агенты могут работать на мобильных телефонах, планшетах, закрепленных на одежде устройствах. Агенты используются как средства, предоставляющие человеку помощь и рекомендации для организации социальных связей, решения совместных задач с другими людьми, проведения мероприятий и др.

В общем случае [75], ИП разворачивается на основе гибридной среды, включающей особенности трех типов вычислительной среды представленных выше. Вычислительная среда может быть локальной, локализованной или глобально распределенной. В случае локальной вычислительной среды ИП является закрытой системой и не имеет доступа к внешним системам

или источникам информации. Доступ к такому ИП осуществляется только там, где оно развернуто (напр., в доме). В случае локализованной среды ИП имеет доступ к внешним источникам информации и к ИП открыт доступ для внешних устройств. Например, ИП развернуто в ресторане — посетитель может подключиться к ИП, находясь как в помещении ресторана, так и из другого места вне ресторана. В случае глобально распределенной среды (напр., через Интернет) ИП представлено сервисами, физическое местоположение которых не определено, но они обеспечивают взаимодействующих агентов необходимой контекстной информацией.

Таким образом, ИП определяет зависимости между такими неоднородными информационными источники как (а) информация о физическом окружении с измерительных устройств (напр., сенсоры и умные объекты в доме или автомобиле), (б) информация от окружающих сервисов (напр., Интернет-сервисы), (в) персональная информация о пользователях (напр., персональный агент на мобильном телефоне). Для систем ИП предоставляет вычислительную инфраструктуру, которая “позволяет участникам накапливать и применять знания о текущем окружении и адаптироваться для улучшения деятельности в этом окружении” [30]. Такая абстракция допускает построение информационной канонической модели и предметных посредников, что обеспечивает интероперабельность при интеграции неоднородных информационных ресурсов, (см., напр., [6]). Предполагается высокий уровень масштабирования инфраструктуры, что позволяет рассматривать ИП как кандидата для реализации технологии IoT [54].

Агенты в ИП обмениваются информацией на основе прямого и косвенного взаимодействия. При прямом взаимодействии участники связыва-

ются друг с другом для обмена информацией. Используются вызовы удаленных процедур (RPC, от англ. Remote Procedure Call) и протоколы обмена сообщениями для передачи информации. Так, в проекте MavHome [31] используется технология CORBA. Прямое взаимодействие требует реализации для всех частных случаев (поддержка различных протоколов обмена сообщениями, необходимость поиска нужного участника). Тем самым для решения проблемы интероперабельности множества агентов выгодно использовать косвенное взаимодействие, наследуемое из асинхронной коммуникационной модели распределенных вычислений [43, 38].

Косвенное взаимодействие агентов организуется через общее пространство — модель “классная доска” (blackboard) [70]. В такой модели каждый агент имеет доступ к “классной доске”, на которой отражается информация, которую агент хочет сообщить другим агентам. В случае “классной доски”, имеется возможность видеть систему, в которой функционируют агенты (контекст работы) в целом, что положительно сказывается на ее целостности [8]. Для ИП такой “классной доской” является ее информационное содержимое.

Для отслеживания изменений в информационном содержимом применяется модель “публикация/подписка” [64]. Агент подписывается на изменения информационного содержимого при помощи запроса (напр., поискового запроса), определяя ту информацию, в отслеживании которой он заинтересован. При изменении отслеживаемой информации агенту высылается уведомление с описанием изменений, которые произошли в информационном содержимом ИП.

Использование приведенных выше моделей косвенного взаимодей-

ствия можно видеть в проекте CLM [57]. Источники информации (сенсоры) передают значения своих параметров диспетчеру. Подписчики через диспетчера получают уведомления об изменениях информации.

Пример 1.1 (“Умный дом”). В качестве примера, рассмотрим систему “умный дом”. В системе используются сенсоры (температура, влажность, освещенность), различное оборудование (вентилятор, кондиционер, музыкальный проигрыватель), персональные агенты пользователей (мобильные телефоны, планшеты). Агенты на различных устройствах обмениваются информацией, формируя в информационном содержимом ИП контекст работы всей системы. Анализ агентами информационного содержимого и отслеживание его изменений позволяет агентам подстраиваться под текущее состояние системы или корректировать состояние для выполнения поставленных перед ними целей. Так, персональные агенты передают в ИП информацию о человеке. Местонахождение человека может быть определено по информации в ИП, публикуемой агентами сенсоров, установленных в доме. Агенты на устройствах отслеживают местоположение человека и при нахождении его в определенной области реагируют включением/выключением света, музыки, приветствием и другими действиями.

В работе рассматриваются ИП с косвенным взаимодействием агентов, где информация представлена и обрабатывается на основе технологий Семантического веб [25, 51, 93, 67]. Используются соответствующие стандарты для форматов сериализации и интероперабельности данных, на основе которых разрабатываются семантические сервисы и персональные агенты [16]. Такие ИП разворачиваются на основе гибридного окружения, удовлетворяющего условиям IoT. Это способствует более тесной интеграции

физической и виртуальной среды, где взаимодействие производится между людьми и различными устройствами (физическими или виртуальными) посредством агентов [50]. Взаимодействие агентов выполняется на основе базовых моделей взаимодействия “классная доска” и “публикация/подписка”. На основе последних в диссертационной работе разрабатываются специализированные модели взаимодействия, позволяющие упростить программирование взаимодействия агентов в ИП.

Доступ любого агента к информационному содержимому ИП выполняется через информационного брокера, обрабатывающего запросы на чтение и запись. Для таких ИП характерно динамическое изменение агентами информационного содержимого. Для интерпретации информационного содержимого ИП в терминах объектов и их семантических связей используются онтологий проблемной области (не обязательно стандартизованные). Состояние отдельных агентов отражается в общем информационном содержимом ИП, формируя контекст работы всей системы. При этом информация в ИП может быть интегрирована с информацией из источников Семантического веб.

Далее рассмотрены а) технологии Семантического веб, обеспечивающие в ИП интероперабельность множества агентов при разделении информационного содержимого за счет единообразного представления данных и интерпретации информации на основе онтологий, б) модели взаимодействия агентов и особенности реализации косвенного взаимодействия в ИП на основе моделей представления и обработки информации Семантического веб и в) особенности разработки агента для автоматизации программирования косвенного взаимодействия в ИП.

1.2. Технологии Семантического веб для представления и обработки информации

В Семантическом веб введена модель RDF [39, 85], описывающая утверждения в виде троек, состоящих из субъекта, предиката и объекта. Субъект и объект соответствуют понятиям в проблемной области (человек, машина, дом и т.п.), а предикат (также называемый свойством) определяет отношение между ними (владелец, друг и т.п.). Имена для субъектов, предикатов и объектов определяются унифицированным идентификатором ресурсов (URI, от англ. Uniform Resource Identifier) [20]. Объект также может являться литералом, чтобы определять конкретные данные (напр., имя, возраст). Множество RDF-троек можно представить в виде RDF-графа, где вершинами являются субъекты и объекты, а дуги помечены предикатами.

Современные методы автоматической обработки данных, доступных в Интернет, как правило, основаны на частотном и лексическом анализе текстового содержимого. В Семантическом веб утверждения кодируются при помощи RDF и ориентированы на эффективную машинную обработку. Интерпретируются такие утверждения (определяются связи между утверждениями, ограничения) с помощью онтологий для получения логических заключений. Под онтологией будем понимать формальное определение понятий и отношений между ними [90, 46].

Информация о каком-либо объекте может находиться в различных источниках информации, которые представлены RDF-хранилищами. Доступ к последним основан на SPARQL точках доступа [42]. Язык запросов

SPARQL позволяет формировать семантические запросы для получения информации из RDF-хранилищ [36]. Полученная из RDF-хранилищ информация может быть интегрирована в один RDF-граф. Последний можно анализировать и при помощи машины логического вывода получать новые утверждения.

В ИП информационное содержимое также находится в RDF-хранилище. Доступ к информационному содержимому выполняется через специфический для конкретного ИП интерфейс. Последний может поддерживать функциональность SPARQL точек доступа. Представление утверждений с помощью модели RDF решает проблему интероперабельности информации в ИП при взаимодействии агентов и дает возможность проводить логический вывод как в ИП (на основе информационного содержимого), так и с использованием информации из внешних источников.

Примитивы доступа к информационному содержимому ИП сводятся к семантическим запросам, их выполнение использует методы сопоставления для поиска в RDF-хранилище [37, 40]. Так, в примере 1.1 “Умный дом” (с. 25) возможен запрос: “получить список аудио устройств, воспроизводящих сейчас звук”.

Утверждения интерпретируются с помощью онтологий, соответствующих стандартам языков RDF Schema [28] и OWL (Web Ontology Language) [76]. В работе рассматриваются онтологии, описанные на языке веб-онтологий OWL. В основе OWL онтологий лежат математические формализмы, называемые дескрипционными логиками [33]. Онтологии используются для описания проблемной области прикладной системы агентов, взаимодействующих в ИП. На их основе агенты интерпретируют ин-

формационное содержимое ИП. В языке OWL используется модель данных “объект-свойство” [77]. Язык основан на словаре RDF Schema, но предоставляет бóльшую выразительность для моделирования и позволяет определять разнообразные ограничения [98]. Объекты, процессы и явления проблемной области представляются OWL-онтологией, используя классы, свойства, индивиды (экземпляры класса).

Агент, работая с информационным содержимым ИП, получает нужную для него информацию о текущем состоянии системы и на основе онтологии может индивидуально интерпретировать это состояние. Проблемная область системы, в которой взаимодействует агент, описывается онтологической моделью. Модель определяет информационные объекты (далее — объекты), посредством которых агенты взаимодействуют между собой в ИП. При этом агенту не доступны все детали проблемной области, т.к. он отвечает лишь за частные, возложенные на него задачи. Таким образом, для отдельного агента разработчику требуется частная онтологическая модель. Последняя определяет подмножество объектов, требуемых агенту для взаимодействия с другими агентами. Такие объекты (их представление) можно использовать в программном коде при реализации действий агента как участника системы (логика агента). В этом случае разработчику необходимо программное обеспечение, предоставляющее интерфейс доступа к ИП с поддержкой необходимых моделей взаимодействия агентов.

1.3. Взаимодействие агентов в интеллектуальном пространстве

В многоагентных системах состояние агента зависит от текущего контекста всей системы или его части [22]. Для реагирования на изменения

контекста агенты обмениваются информацией, взаимодействуя между собой. При разработке программных агентов для многоагентных систем используются стандарты, создаваемые организацией FIPA (Foundation for Intelligent Physical Agents). Они описывают архитектуру, способы, протоколы прямого взаимодействия агентов и пр. [79, 91]. Прямое взаимодействие предполагает применение протоколов, использующих наборы сообщений для передачи информации. Например, на платформе Jade [24] агенты могут обмениваться сообщениями друг с другом.

В случае косвенного взаимодействия также могут быть использованы стандарты FIPA, но с учетом специфики использования общего хранилища и операции подписки. Для ИП в работе [67] рассматриваются типы косвенного взаимодействие и определены четыре основных типа (будут представлены далее в п. 1.4). На основе поддерживаемых ИП примитивов доступа (получить/удалить/поместить RDF-тройки и др.) и операции подписки разработчик программирует взаимодействие агентов в ИП, реализуя один или несколько основных типов взаимодействия в логике агента.

Для решения задач диссертационного исследования в работе разрабатываются специализированные модели косвенного взаимодействия агентов в ИП с использованием технологий Семантического веб. В результате решения поставленных задач разработчику будут предоставлены механизмы программирования основных типов косвенного взаимодействия в терминах проблемной области.

Реализация таких механизмов программирования возможна на уровне промежуточного ПО [86]. Место промежуточного ПО в структуре программной реализации агента показано на рис. 1.2. Промежуточное ПО

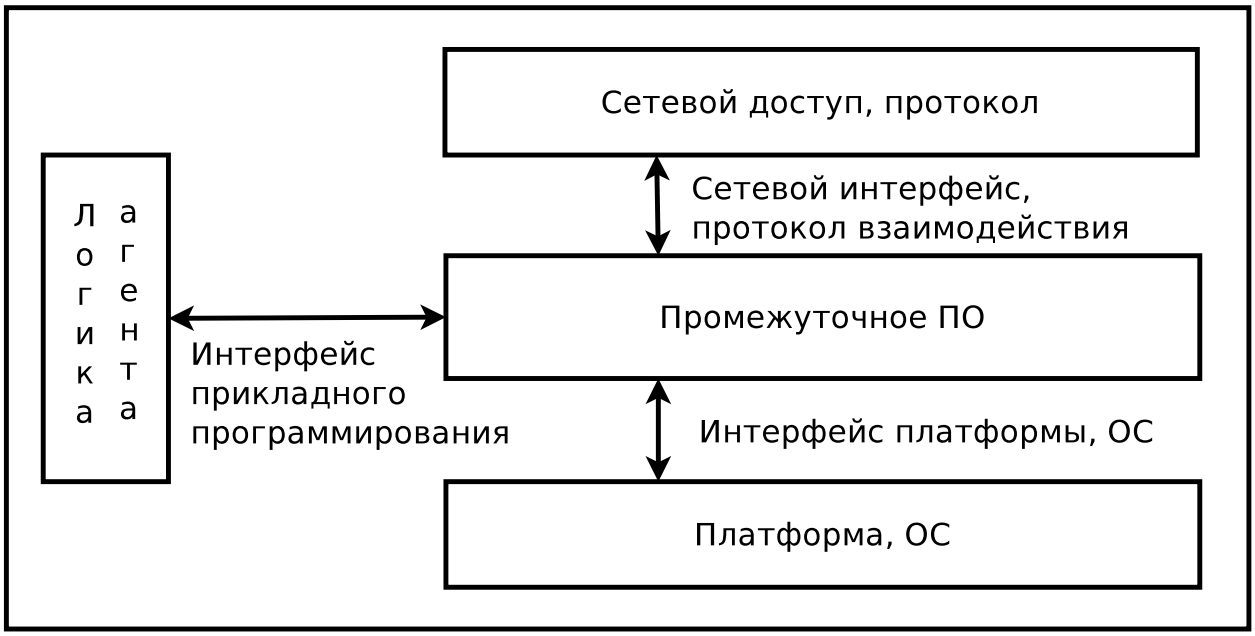


Рис. 1.2. Место промежуточного ПО в общей структуре программной реализации агента.

предоставляет разработчику агента дополнительный уровень абстракции и интерфейс прикладного программирования (API, от англ. Application Programming Interface), скрывающий низкоуровневые интерфейсы аппаратных и программных компонентов среды (платформа, сетевые интерфейсы, протоколы, операционная система). На уровне промежуточного ПО возможна организация параллельных вычислительных потоков, возникающих при доступе к одному или нескольким ИП, интеграции информации из различных ИП, передаче или получении информации и обработке обновлений по подписанной информации.

Данная диссертационная работа направлена на развитие промежуточного ПО, основанного на предлагаемых автором методе программирования косвенного взаимодействия агентов, специализированных моделях взаимодействия и механизмах программирования взаимодействия. Промежуточное ПО позволит реализовать косвенное взаимодействие агентов —

Таблица 1.1

Основные типы косвенного взаимодействия.

	Тип	Описание
1	Интеллектуальное соединение.	Ожидание изменения контекста работы системы для продолжения выполнения действий агента. Изменение контекста зависит от работы других агентов, и эта зависимость может изменяться со временем.
2	Централизованное хранилище.	Отслеживание изменения данных в общем хранилище информации.
3	Концентратор информации.	Получение информации из распределенных источников для ее дальнейшей обработки.
4	Глобальный посредник.	Обмен и преобразование информации представленной различными форматами или онтологиями.

получение, интеграцию информации из различных источников и отслеживание изменений. Промежуточное ПО может работать как поверх стандартизированных протоколов FIPA, так и специализированных протоколов для конкретной реализации ИП. Задачи, связанные с целями взаимодействия агентов (напр., делегирование выполняемой работы, объединение для совместной обработки, управление и др.), интерпретацией информационного содержимого, обучением, а также ряд других, выходят за рамки данной работы.

1.4. Программирование косвенного взаимодействия агентов

В работе [67] определены основные примитивы доступа (получить, удалить, обновить RDF-тройки, операция подписки) характерные для исследуемых в работе ИП. Прикладной разработчик использует такие примитивы доступа для программирования взаимодействия в логике агента с учетом типов взаимодействия, которые необходимо реализовать. Основные типы взаимодействия, определенные в [67], представлены в табл 1.1.

Разработчик сталкивается со следующими проблемами при реализа-

ции основных типов взаимодействия в логике агента для ИП [7, 23].

1. Программирование взаимодействия с другими агентами в ИП. В логике агента необходимо реализовать сетевой доступ к ИП (в общем случае, в виде набора параллельных вычислительных потоков к одному и более ИП). Программный код оперирует с определенными структурами данных при чтении/записи информационного содержимого.

2. Программирование реакции агента на изменения информационного содержимого ИП, вносимые другими агентами. В логике агента необходимо реализовать подписку на объекты в информационном содержимом ИП для отслеживания изменений. В программном коде задается реакция на поступающие изменения с внесение ответных изменений.

На решение этих проблем влияют, во-первых, ограничения целевого устройства для программного агента: язык программирования, вычислительные ресурсы, организация вычислительных потоков, средства сетевых коммуникаций. Во-вторых, трудоемкость разработки зависит от уровня абстракции для представления в программном коде обрабатываемого информационного содержимого.

Два уровня разработки программного агента для ИП показаны на рис. 1.3. При низкоуровневой разработке (уровень RDF-представления) в программном коде используется представление в виде RDF-троек, т.е. без манипуляций в коде с целостными объектами онтологической модели. Сетевой доступ к ИП осуществляется через RDF-ориентированный ИП-интерфейс. В последнем реализуется протокол сетевого доступа агента к ИП и набор базовых примитивов доступа для работы агента с информационным содержимым ИП. Примитивы доступа позволяют манипулировать

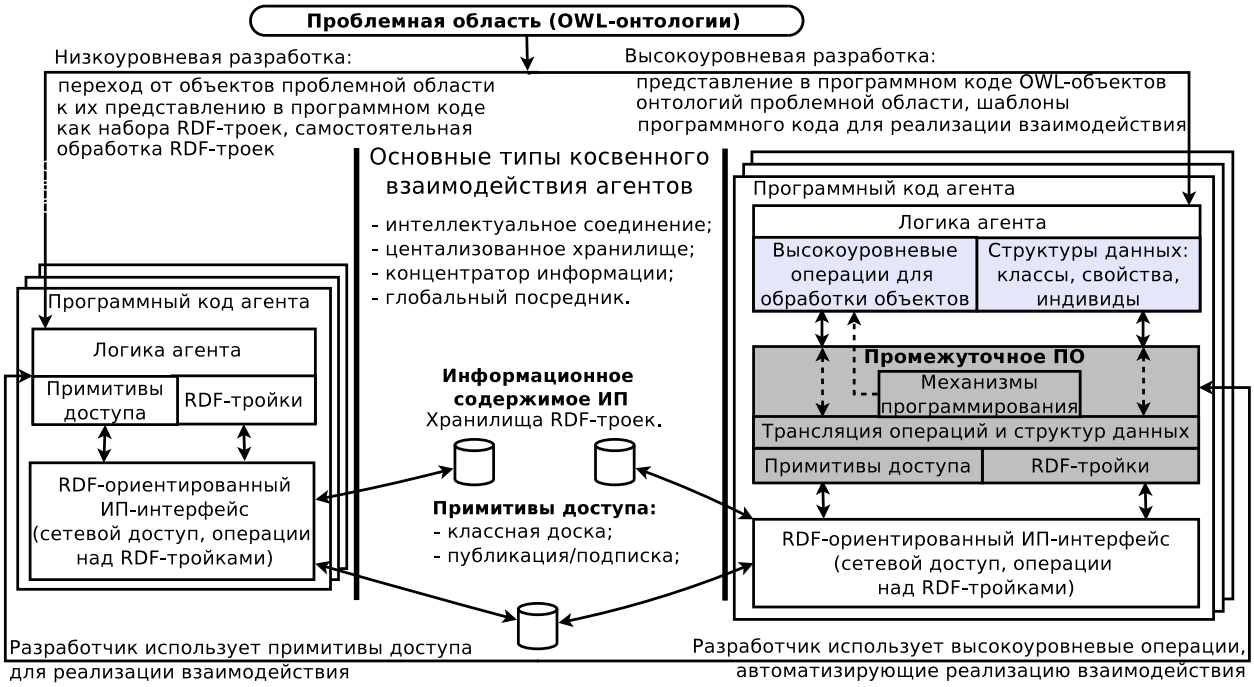


Рис. 1.3. Структура программного кода агента в зависимости от низкоуровневого (слева) и высокоуровневого (справа) способа разработки.

с RDF-тройками и выполнять поиск в RDF-хранилище. В то же время, практически значимые проблемные области приводят к большим наборам RDF-троек. Для автоматической обработки такие объемы могут не являться критичными, но использование RDF-троек в логике агента приводит к громоздкому и труднообозримому программному коду. Более выгодно использовать высокоуровневый способ, программируя логику агента на основе представления объектов проблемной области.

Высокоуровневая разработка (уровень OWL-представления) скрывает операции с RDF-тройками в промежуточном ПО. Разработчику предоставляется дополнительный уровень абстракции в виде структур данных для онтологических объектов (OWL классы, свойства и индивиды) и интерфейс прикладного программирования. Промежуточное ПО увеличивает общий объем программного кода агента, но, при этом

1) предоставляет механизмы программирования на основе моделей взаимодействия, которые включают шаблоны программного кода, используемые разработчиком при программировании косвенного взаимодействия в логике агента,
2) проводит преобразование между представлением объектов проблемной области в программном коде и структурами данных ИП-интерфейса
3) скрывает от разработчика низкоуровневые детали (использование ИП-интерфейса, примитивов доступа, операции подписки).

Таким образом, повышается эффективность разработки и упрощается программирование взаимодействия за счет приближения к терминам проблемной области. В результате, уменьшается программный код, который непосредственно создается и тестируется разработчиком.

Для поддержки высокоуровневой разработки агентов необходимо реализовать и предоставить разработчику части выделенные темным фоном на рис. 1.3. Таким образом, в ходе диссертационной работы необходимо а) реализовать промежуточное ПО и механизмы программирования и б) предоставить разработчику высокоуровневые операции (на основе промежуточного ПО) и возможность генерации структур данных для объектов проблемной области. При этом требуется решить проблемы, которые возникают при реализации основных типов взаимодействия (табл. 1.1). Для решения определены базовые задачи программирования логики агента.

Базовые задачи программирования логики агента приведены на рис. 1.4. Их решение позволит избежать проблем программирования логики агента, связанных с представлением объектов проблемной области в

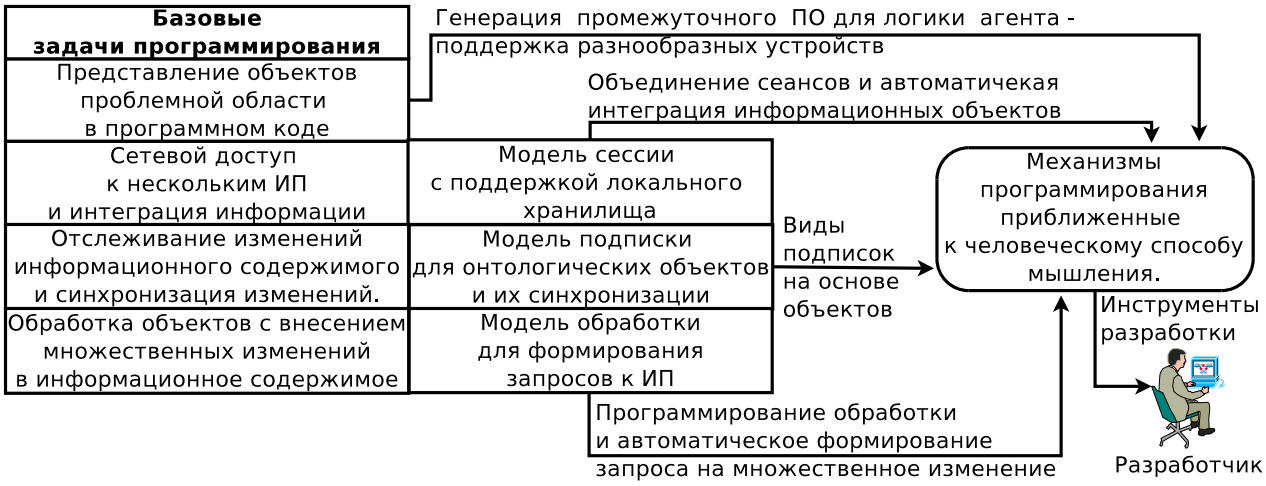


Рис. 1.4. Базовые задачи программирования логики агента.

виде структур данных, параллельным доступом к одному или нескольким ИП, обработкой информационного содержимого и реакцией на изменения в ИП. На уровне OWL-представления решение допускает автоматизацию: промежуточное ПО предоставляет механизмы программирования взаимодействия с другими агентами через разделение в информационном содержимом объектов онтологической модели проблемной области. Для такой автоматизации и реализации механизмов программирования необходимы специализированные модели взаимодействия. Такие модели разрабатываются на основе базовых моделей взаимодействия в ИП (“классная доска”, “публикация/подписка”) и направлены на автоматизацию реализации основных типов косвенного взаимодействия в терминах проблемной области.

Для удобного использования разработчиком механизмов программирования косвенного взаимодействия в логике агента необходимо объединить их в рамках одного метода. В работе [11] определено, что использование онтологий при разработке ПО позволяет получить 1) инструменты, работающие ближе к человеческому способу мышления, 2) упроще-

ние повторного использования знаний за счет уже определенных в других онтологиях понятий, 3) повышение уровня автоматизации за счет автоматических логических рассуждения и 4) повышение надежности и сокращение стоимости сопровождения ПО за счет использование формальной логики. Таким образом, выгодно использовать особенности онтолого-ориентированного подхода [10, 11, 53], где онтологии применяются на всех основных этапах разработки. Для автоматизированного построения программного кода используется модельно-ориентированный подход к разработке ПО [78, 10].

В результате, разработчик моделирует при помощи онтологий проблемную область, получая общую онтологическую модель проблемной области [1]. На ее основе формируются частные онтологические модели проблемной области агентов. Такие модели определяют наборы объектов, на основе которых агент взаимодействует с другими агентами через обмен информации в ИП. В диссертационной работе предполагается использовать связывание частной онтологической модели со структурами целевого языка программирования агента. Для трансляции модели в программный код агента можно использовать методы из модельно-ориентированного подхода к разработке ПО. Такое представление онтологической модели в программном коде позволит автоматизировать процессы интеграции, проверки корректности и ограничений для поступающей к агенту информации.

Использование возможностей онтолого-ориентированного подхода также позволить автоматизировать разработку для базовых задач программирования логики агента (см. 1.4). Для реализации возможностей связанных с доступом к распределенной информации, интеграцией информа-

ции, синхронизацией локальных объектов и обработкой групп объектов необходимы специализированные модели взаимодействия. На основе моделей будут разрабатываться программные механизмы и инструменты для разработки агента.

1.5. Выводы

Рассмотрена парадигма интеллектуальных пространств для разработки многоагентных систем с косвенным взаимодействием и использованием технологий Семантического веб. Определены базовые задачи программирования логики агента для таких ИП и роль промежуточного ПО для автоматизации решения базовых задач. Решение базовых задач необходимо для обеспечения взаимодействия агентов в ИП. Автоматизация решений базовых задач и применение высокоуровневой разработки агентов на основе модельно-ориентированного подхода позволят повысить эффективность программирования взаимодействие в логике агента. Далее в работе предлагается метод программирования косвенного взаимодействия агентов, основанный на промежуточном ПО и специализированных моделях взаимодействия агентов.

2. Метод программирования косвенного взаимодействия агентов

В главе предлагается метод программирования косвенного взаимодействия, автоматизирующий разработку агентов ИП. Метод использует программные онтологические библиотеки в качестве промежуточного ПО и требует специализированные модели взаимодействия агентов. Определены шаги метода и схема использования онтологической библиотеки для разработки агента. Предлагаются специализированные модели взаимодействия на основе: 1) многоэлементной сессии параллельных сетевых сеансов с поддержкой автоматической интеграции информации из различных ИП, 2) операции подписки для описания отслеживаемых изменений в терминах проблемной области и синхронизации, локально хранимых объектов проблемной области, с изменениями в ИП, 3) локальной обработки группы объектов проблемной области для выполнения агентом запроса на множественное изменение информационного содержимого в виде транзакции. На основе предлагаемых метода и моделей взаимодействия будут разработаны механизмы программирования взаимодействия агентов.

Результаты автора, представленные в данной главе, опубликованы в [66, 44, 59].

2.1. Программирование взаимодействия агентов в терминах проблемной области на основе программной онтологической библиотеки

В ходе диссертационного исследования разработан метод, на основе которого генерируется промежуточное ПО, и используются предлагаемые специализированные модели взаимодействия для автоматизации решения задач программирования логики агентов (см. рис. 1.4 на с.36).

Разработанный метод определяет следующие шаги.

Шаг 1. Онтологическое моделирование проблемной области.

Шаг 2. Формирования частной онтологической модели проблемной области агента на основе выбора объектов для взаимодействия с другими агентами и автоматическая генерация программного кода онтологической библиотеки.

Шаг 3. Программирование логики агента и взаимодействия с другими агентами, используя структуры данных и функции из онтологической библиотеки.

Онтологическая библиотека реализуется как промежуточное ПО для высокоуровневой разработки агентов, которое а) включает структуры данных, представляющие объекты частной онтологической модели и б) предоставляет разработчику агента интерфейс прикладного программирования, параметризируемый объектами онтологической модели.

Генерация программного кода онтологической библиотеки выполняется на основе модельно-ориентированного подхода к разработке ПО. Онтологическая модель определяет объекты (классы, свойства, индивиды),

посредством которых агент взаимодействует с другими агентами в ИП. Онтологическая модель формируется на основе OWL-онтологий. Для заданного агента формируется частная онтологическая модель с необходимыми для работы агента объектами из исходных онтологий проблемной области.

Существуют похожие методы для описания содержания сообщений при прямом взаимодействии агентов. Например, в среде MASDK [5] разработчик использует графическое средство, позволяющее сформировать диаграммы для описания агентов и их взаимодействия. Диаграммы онтологий позволяют определить, какие понятия проблемной области нужно использовать в программном коде для описания содержания сообщений, спецификации параметров и переменных. Среда MASDK формирует программную библиотеку для некоторого класса агентов. С помощью такой "библиотеки класса агентов" прикладной разработчик программирует логику обработки сценариев взаимодействия агентов.

В предлагаемом в данной диссертационной работе методе онтологическая библиотека может использоваться как одним агентом, так и несколькими агентами. Агент может использовать различные онтологические библиотеки, если это необходимо для выполнения его задач. Так, агент, реализованный в среде MASDK, может использовать сторонние программные библиотеки, в том числе, рассматриваемые онтологические библиотеки. Таким образом, разработчик получает преимущества, предлагаемые средствами разработки среды MASDK, и расширяет возможности агента для косвенного взаимодействия в ИП при помощи онтологической библиотеки.

Схема использования онтологической библиотеки для шагов 1–3

предлагаемого метода показана на рис. 2.1. Онтологическая библиотека состоит из частей l_{data} и l_{func}. В части l_{data} определены структуры данных для представления объектов частной онтологической модели агента на целевом языке программирования. Она генерируется на основе частной онтологической модели и при изменении последней (напр., при изменении онтологий) должна быть сгенерирована заново.

Инвариантная функциональная часть l_{func} содержит функции для программирования взаимодействия агента в общеонтологических терминах — “класс”, “свойство”, “индивид”. Тем самым, программная реализация l_{func} ориентирована на целевую аппаратно-программную платформу без привязки к конкретной проблемной области. Разработчик использует в логике агента вызовы программных функций из l_{func} с параметрами на основе структур данных из l_{data}. Для сетевого доступа агента к ИП онтологическая библиотека автоматически выполняет преобразования между структурами данных l_{data} и RDF-представлением ИП-интерфейса. Предпо-

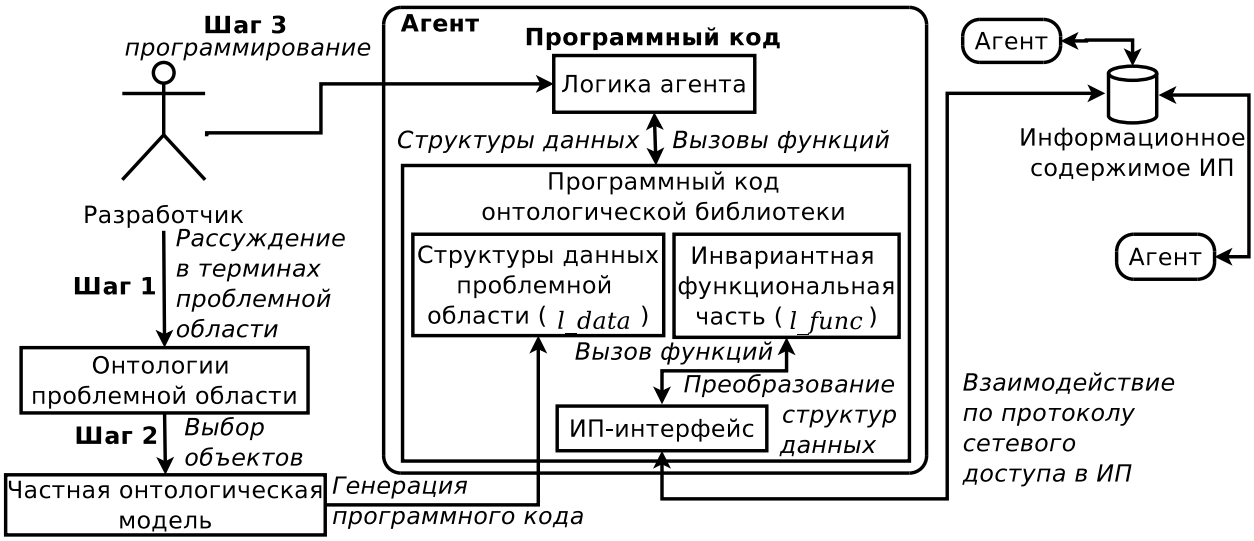


Рис. 2.1. Схема использования онтологической библиотеки в рамках метода программирования косвенного взаимодействия.

лагается возможность использования различных ИП-интерфейсов посредством программных адаптеров.

Онтологическая библиотека позволяет следующим образом повысить эффективность разработки агентов. Разделение онтологической библиотеки на две части повышает гибкость разработки, так как позволяет использовать а) часть l_{func} для любого агента (в рамках заданной аппаратно-программной платформы) независимо от его частной онтологической модели и б) часть l_{data} при программировании множества агентов, для которых частные онтологические модели совпадают. Последняя ситуация нередко возникает как в рамках одной системы, так и различных систем для одной проблемной области. Использование онтологической библиотекой различных ИП-интерфейсов обеспечивает независимость от конкретной реализации протокола доступа к ИП и позволяет разрабатывать агентов для широкого круга аппаратно-программных платформ, выбирая ИП-интерфейс для целевой платформы.

Для анализа необходимого объема ресурсов, вызываемых переходом к разработке на уровне OWL-представления и использованием онтологической библиотеки, введены емкостная и временна́я оценки. Для маломощных устройств эти затраты должны быть ограничены. Емкостная трудоемкость онтологической библиотеки оценивается как

$$R_{\text{mem}} = v_{\text{data}} + v_{\text{func}} + n_{\text{loc}}C_{\text{obj}} + n_{\text{ses}}C_{\text{ses}} + n_{\text{sub}}C_{\text{sub}} \tag{2.1}$$

где величины v_{data} и v_{func} — необходимый объем памяти для загрузки программных реализаций l_{data} и l_{func} на устройство. Параметры C_{obj}, C_{ses}, C_{sub} отражают средние затраты по памяти для хранения на стороне агента од-

ного локального индивида, сеанса и подписки. Параметры n_{loc}, n_{ses}, n_{sub} — это число соответственно локальных индивидов, сеансов и подписок, используемых логикой агента.

Время выполнения функций из l_{func} зависит от автоматических преобразований между структурами данных из l_{data} и RDF-представлением ИП-интерфейса. Временна́я трудоемкость онтологической библиотеки оценивается как

$$R_{\text{proc}} = \lambda n_{\text{th}}(C_{\text{obj-tpr}} + C_{\text{tpr-obj}}) \tag{2.2}$$

где $C_{\text{obj-tpr}}$ и $C_{\text{tpr-obj}}$ — средние вычислительные затраты на преобразование (из представления OWL-объектов в RDF-тройки и обратно), λ — интенсивность вызова функций онтологической библиотеки в одном вычислительном потоке, n_{th} — число параллельных вычислительных потоков. Параметры $C_{\text{obj-tpr}}$ и $C_{\text{tpr-obj}}$ зависят от количества хранимых локально индивидов n_{loc} и классов и свойств в l_{data}. При использовании частной онтологической модели уменьшаются значения l_{data}, $C_{\text{obj-tpr}}$ и $C_{\text{tpr-obj}}$. Объем памяти v_{func} может быть уменьшен отключением в части f_{func} не используемых, либо не поддерживаемых конкретной платформой или устройством возможностей. Например, может быть ограничение на создание параллельных вычислительных потоков, в этом случае в части f_{func} асинхронные функции могут быть отключены.

В примере 1.1 “умный дом” онтологии описывают 1) людей и их персональные данные, 2) сенсоры физического окружения и 3) бытовое оборудование. Поскольку агент сенсора оперирует с небольшим набором измеряемых параметров и операции доступа к ИП сводятся к простым операциям чтения/записи, то генерируемая для него часть l_{data} не требует трудоемких

программных конструкций.

Таким образом, метод определяет использование онтологических библиотек для разработки агентов, но для программирования взаимодействия агента (третий шаг метода) необходима реализация в части l_{func} механизмов. Последние позволят автоматизировать решение базовых задач программирования взаимодействия в логике агента (см. рис. 1.4 на с. 36). Для реализации таких механизмов в методе используются предлагаемые специализированные модели косвенного взаимодействия агентов, где объекты онтологической модели выступают базовыми элементами при программировании взаимодействия в логике агента.

2.2. Модель взаимодействия агентов на основе многоэлементной сессии для организации сетевого доступа к интеллектуальным пространствам

Программирование логики агента использует примитивы сетевого доступа к ИП. Во время работы агент устанавливает сеанс сетевого доступа к ИП, который управляет обменом данных, определением прав на передачу данных и работает на сеансовом уровне поверх транспортного уровня модели OSI [13]. Предлагаемая модель направлена на решение базовой задачи сетевого доступа к нескольким ИП и интеграции информации на стороне агента. В результате, разработчик сможет реализовать основной тип взаимодействия “концентратор информации” (см. п. 1.1 на с. 32).

Когда агент взаимодействует в одном или нескольких ИП, то при получении информации из ИП нужно анализировать связи между объектами по частной онтологической модели и проводить интеграцию информации

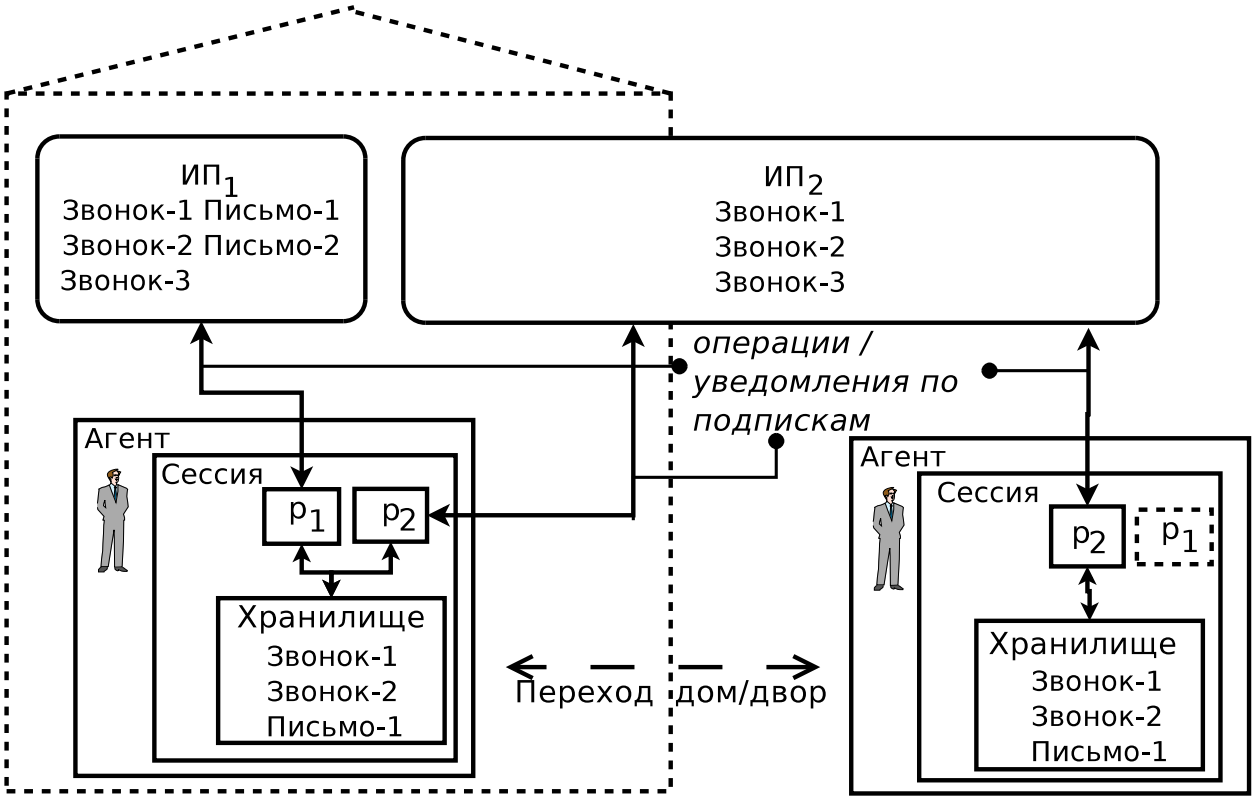


Рис. 2.2. Пример использования агентом параллельных сеансов сетевого доступа к двум интеллектуальным пространствам.

в локальных объектах. В рамках одного сеанса может быть установлено несколько подписок. Получаемая по ним информация также интегрируется на стороне агента в локально хранимых объектах. В логике агента сеансы не создают параллельных вычислительных потоков, но подписки могут автоматически создавать такие потоки для проверки и обработки уведомлений. Подробнее про модель взаимодействия на основе подписки и обработку уведомлений в п.2.3.

Пример 2.1 (“Умный дом: переключение между ИП”). Рассмотрим, как использовать параллельные сеансы на основе примера 1.1 “Умный дом”. Пусть $ИП_1$ развернуто внутри дома, а $ИП_2$ развернуто как в доме, так и снаружи (рис. 2.2). В $ИП_1$ доступна информация о звонках и электронных

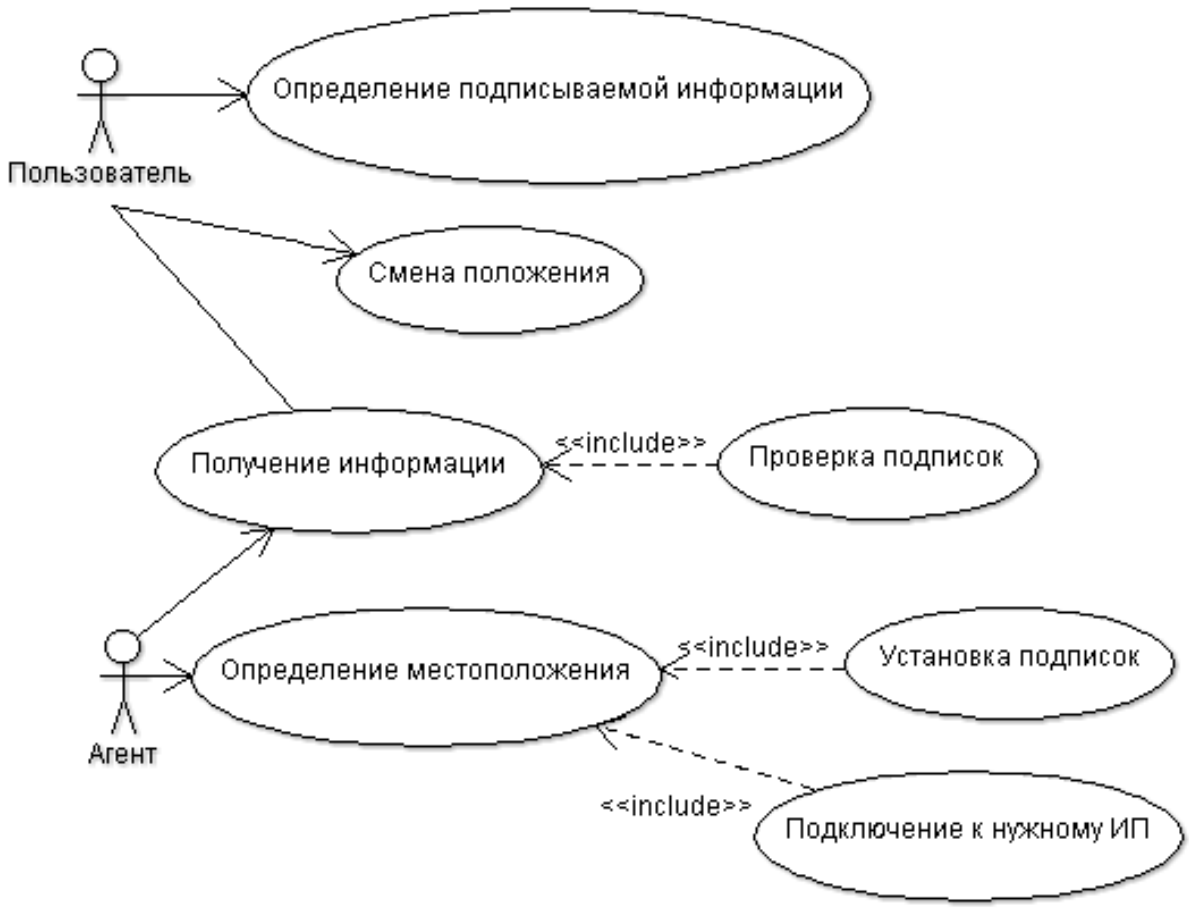


Рис. 2.3. UML-диаграмма прецедентов использования параллельных сеансов в условиях изменения местоположения пользователя.

письмах, в $ИП_2$ — о звонках. В доме агент использует два параллельных сеанса (p_1, p_2), снаружи — один (p_2). При перемещении агента необходимо управлять сеансами, подписками внутри сеансов и локальным хранилищем информации. Так, при выходе из дома нужно завершить сеанс p_1 и все подписки, установленные к $ИП_1$, оставив работать сеанс p_2. При возвращении в дом сеанс p_1 и подписки должны быть восстановлены. Агенту нужно провести синхронизацию хранилища с информацией из $ИП_1$ — добавить данные о письмах и звонках в локальное хранилище с учётом дублирования.

UML-диаграмма прецедентов на рис. 2.3 показывает возможности использования параллельных сеансов для случая из примера 2.1. Для пользователя необходимо автоматическое подключение его агента к ИП и возобновления подписок при изменении местоположения. В результате, пользователь будет получать необходимую информацию независимо от своего текущего местоположения.

Для поддержки параллельных сетевых сеансов на стороне агента требуется решить следующие задачи.

1. Реализовать управление сетевыми сеансами и подписками (есть подключение к ИП или нет) для временного прекращения работы сеанса или подписки и возобновления работы, без необходимости явно задавать параметры сеанса или подписки.

2. Организовать хранение и интеграцию на стороне агента информации из различных ИП с интеграцией информации на стороне агента в виде объектов онтологической модели и поддержкой синхронизации этих объектов с информационным содержимым ИП.

Решение данных задач позволит решить базовую задачу сетевого доступа к нескольким ИП и интеграции информации (см. рис. 1.4 на с. 36). В модели взаимодействия предлагается объединить работу сетевых сеансов агента в одну сессию и реализовать локальное хранилище информации, разделяемое между всеми сеансами сессии. Для сессии определяются операции управления состояниями сетевых сеансов и подписок (установлено или нет соединение с ИП). Операции управления используются разработчиком при программировании взаимодействия в логике агента. В локальном хранилище агента содержатся объекты частной онтологической моде-

ли, которые используются агентом для взаимодействия в ИП. В локальном хранилище выполняется интеграция информации — преобразование в объекты с учетом ограничений и связей частной онтологической модели. Интеграция информации в хранилище происходит в двух случаях: при получении информации посредством сетевого сеанса и с уведомлением об изменениях в ИП по подписке. Таким образом, разработчик использует сессию с набором сеансов и подписок подключенных к одному или нескольким ИП. Получаемая информация автоматически интегрируется в локальном хранилище, объекты из которого могут быть использованы для взаимодействия в ИП. На основе подписок происходит синхронизация локального хранилища. Объекты, изменения которых требуется отслеживать, задаются разработчиком при программировании логики агента.

Сеансы и подписки объединяются в сессии s, реализуемой на прикладном уровне сетевой модели OSI [13]. Такая сессия является частным случаем многоэлементной сессии [32] и используется агентом как (P, Q, D), где P — множество сеансов сетевого доступа к ИП, Q — множество подписок для отслеживания изменений в ИП, D — множество объектов онтологической модели в локальном хранилище (классов, свойств, индивидов). Всё множество объектов частной онтологической модели агента обозначим как O. Для использования сеансов и подписок в сессии необходимы операции преобразования между RDF-тройками и структурами данных из l_{data}. Данные операции реализуются в части l_{func} онтологической библиотеки.

Для сессии s сеанс $p \in P$ представим как (Q_p, O_p), где $Q_p \in Q$, $O_p \in D$. Каждый сеанс или подписка находится в одном из двух состояний: активное или состояние ожидания. Таким образом, множества сеансов P

и подписок Q — это дизъюнкции $P_{\text{ac}} \cup P_{\text{wt}}$ и $Q_{\text{ac}} \cup Q_{\text{wt}}$, где P_{ac} и Q_{ac} — множество активных сеансов и подписок, P_{wt} и Q_{wt} — множество сеансов и подписок в состоянии ожидания. Для сеанса активное состояние означает установленное сетевое соединение с ИП при помощи ИП-интерфейса. Для активной подписки ожидаются уведомления об изменениях в информационном содержимом ИП. В состоянии ожидания сеансы и подписки определены только на стороне агента, сетевое соединение с ИП отсутствует. В состоянии ожидания в сессии хранятся данные для возобновления работы сетевых сеансов и подписок. Следовательно, при программировании логики агента вместо явного завершения сетевого сеанса и последующей установки нового можно использовать компактные программные конструкции для переключения состояний сеансов и подписок.

При использовании сессии емкостная трудоемкость онтологической библиотеки является частным случаем оценки (2.1) с учетом состояния сеансов и подписок:

$$R_{\text{mem}} = v_{\text{data}} + v_{\text{func}} + n_{\text{loc}} C_{\text{obj}} +$$

$$n_{\text{ac\_ses}} C_{\text{ses}} + n_{\text{ac\_sub}} C_{\text{sub}} + n_{\text{wt\_ses}} C_{\text{ses}}^{\text{wt}} + n_{\text{wt\_sub}} C_{\text{sub}}^{\text{wt}} \tag{2.3}$$

где $n_{\text{ac\_ses}}$, $n_{\text{ac\_sub}}$ и $n_{\text{wt\_ses}}$, $n_{\text{wt\_sub}}$ — это количество сеансов и подписок в активном состоянии и состоянии ожидания. Параметры $C_{\text{ses}}^{\text{wt}}$ и $C_{\text{sub}}^{\text{wt}}$ — это параметры, выражающие средние затраты по памяти для хранения агентом служебных данных о сеансах и подписках в состоянии ожидания ($C_{\text{ses}}^{\text{wt}} < C_{\text{ses}}$ и $C_{\text{sub}}^{\text{wt}} < C_{\text{sub}}$). Таким образом, разработчик использует состояние ожидания для хранения неиспользуемых в данный момент сеансов и подписок, что позволяет экономить ресурсы вычислительного устройства.

Таблица 2.1
Операции управления в сессии сеансами, подписками и объектами онтологической модели.

Операция	Формальное описание
Регистрация сеанса, подписки или объекта онтологической модели в сессии; объекты онтологической модели помещаются в локальное хранилище.	$REG(s,x) = \begin{cases} (P_s \cup \{x\}, Q_s, D_s), & \text{если } x \in P \text{ и } x \notin P_s; \\ (P_s, Q_s \cup \{x\}, D_s), & \text{если } x \in Q \text{ и } x \notin Q_s; \\ (P_s, Q_s, D_s \cup \{x\}), & \text{если } x \in O \text{ и } x \notin D_s; \end{cases}$
Удаление сеанса, подписки или объекта онтологической модели из сессии; объекты онтологической модели удаляются из хранилища.	$DEL(s,x) = \begin{cases} (P_s \setminus \{x\}, Q_s, D_s), & \text{если } x \in P_s; \\ (P_s, Q_s \setminus \{x\}, D_s), & \text{если } x \in Q_s; \\ (P_s, Q_s, D_s \setminus \{x\}), & \text{если } x \in O_s; \end{cases}$
Активация сессии приводит к активации всех зарегистрированных сеансов.	$ACTIVE(s) \to ACTIVE(s,p)$ для $\forall p \in P_s$
Активация сеанса или подписки. Активация сеанса приводит к активации подписок, связанных с сеансом в сессии.	$ACTIVE(s,x) = \begin{cases} (P_s = \{P_{\text{wt}}\ x\} \cup \{P_a \cup x\}\}, Q_s, D_s) \to \\ \quad ACTIVE(s,q) \text{ для } \forall q \in Q_x, \\ \quad \text{если } x \in P_s; \\ (P_s, Q_s = \{Q_{\text{wt}}\ x\} \cup \{Q_{\text{ac}} \cup x\}\}, D_s), \\ \quad \text{если } x \in Q_s; \end{cases}$
Переход сессии в состояние ожидания переводит в состояние ожидания все сетевые сеансы.	$WAIT(s) \to WAIT(s,p)$ для $\forall p \in P_s$
Переход сеанса или подписки в состояние ожидания. Переход сеанса в состояние ожидания переводит в состояние ожидания все подписки, связанные с сеансом в сессии.	$WAIT(s,x) = \begin{cases} (P_s = \{P_{\text{ac}}\ x\} \cup \{P_{\text{wt}} \cup x\}\}, Q_s, D_s) \to \\ \quad WAIT(s,q) \text{ для } \forall q \in Q_x, \text{ если } x \in P_s; \\ (P_s, Q_s = \{Q_{\text{a}}\ x\} \cup \{Q_{\text{wt}} \cup x\}\}, D_s), \\ \quad \text{если } x \in Q_s; \end{cases}$

Управление сеансами, подписками и локальными объектами онтологической модели выполняется на основе набора операций

$$U = \{REG, DEL, ACTIVE, WAIT\},$$

формальное описание которых представлено в табл. 2.1. Здесь REG и DEL — операции ассоциации сеанса, подписки и объекта онтологической

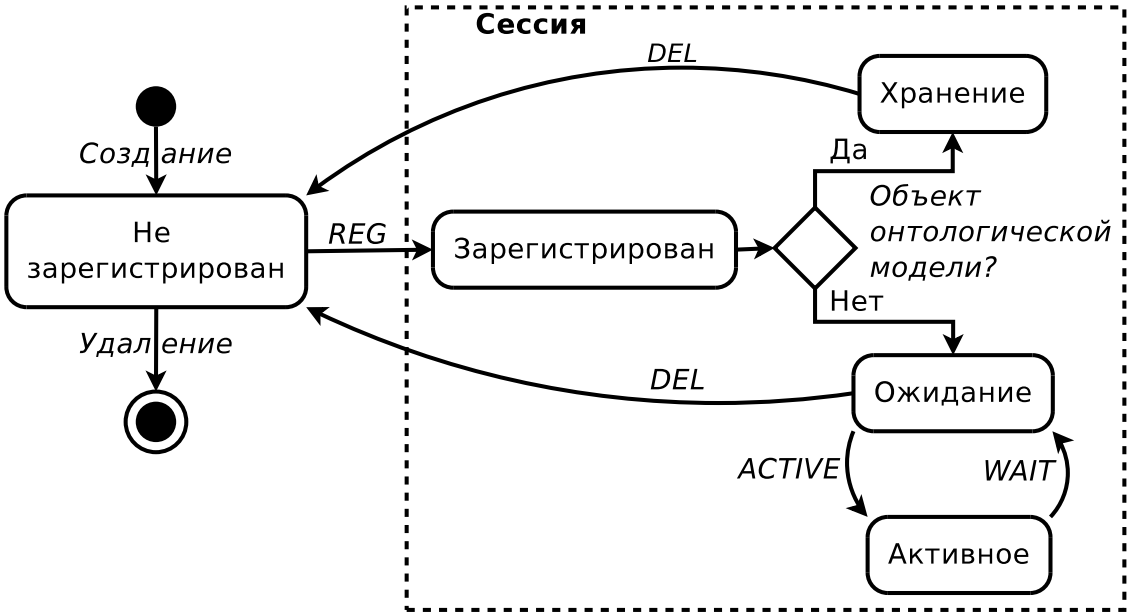


Рис. 2.4. UML-диаграмма состояний для жизненного цикла в сессии сеанса, подписки и объекта онтологической модели.

модели с сессией и удаления их из сессии, $ACTIVE$ и $WAIT$ — операции активации и ожидания для управления состоянием всей сессии, отдельных сеансов и подписок.

UML-диаграмма состояний на рис. 2.4 описывает жизненный цикл сеанса, подписки и объекта онтологической модели в сессии, который начинается при регистрации в сессии и заканчивается удалением из сессии. Объекты онтологической модели после регистрации помещается в локальное хранилище D. На основе множества объектов, представляющих классы из онтологической модели, определяются возможные типы индивидов, которые могут быть зарегистрированы в сессии. Множество объектов, представляющих свойства, задает типы данных и связей, которые могут быть ассоциированы с индивидами. Сеансы и подписки после регистрации находятся в состоянии ожидания. Изменение состояния выполняется операциями $ACTIVE$ и $WAIT$. Данные операции могут выполняться многократно

для отдельного сеанса или подписки. Состояние сеанса определяет состояние всех подписок в рамках этого сеанса. Состояние сессии определяется состоянием всех сеансов и подписок. После удаления из сессии сеанс, подписка или объект онтологической модели не может быть использован для операций в сессии.

Таким образом, для использования сессии разработчику нужно определить 1) структуру локального хранилища при помощи регистрации объектов онтологической модели (классов, свойств) и 2) в каких ИП взаимодействует агент, регистрируя сеансы и подписки. После этого разработчик может использовать операции управления состоянием для сеансов и подписок в логике агента. Действия, связанные с интеграцией информации в локальном хранилище, проводятся автоматически на основе зарегистрированных классов и свойств. При помощи подписок локальное хранилище агента синхронизируется с информационным содержимым ИП. В результате, для разработчика упрощается программирование в логике агента параллельного взаимодействия в одном или нескольких ИП.

2.3. Модель взаимодействия агентов на основе операции подписки для отслеживания изменений информационного содержимого на уровне объектов проблемной области

Для отслеживания изменений в информационном содержимом ИП используется операция подписки. Предлагаемая модель направлена на решения базовой задачи отслеживания изменений в хранилищах информации и синхронизации изменений. Модель позволит определять подписываемую информацию в виде классов, индивидов и их свойств и автоматически об-

новлять локальные объекты онтологической модели, синхронизируя их с информационным содержимым ИП. В результате, разработчик сможет реализовать основные типы взаимодействия “интеллектуальное соединение” и “централизованное хранилище” (см. п. 1.1 на с. 32). Для решения используется шаблон проектирования “издатель-подписчик” базовой модели взаимодействия “публикация/подписка”, определяющей зависимость “один ко многим” между взаимодействующими участниками распределенной системы [3, 64]. Такой подход позволяет создавать системы с проактивным предоставлением нужной информации и доставкой сервисов [12, 63, 75].

Пример 2.2 (“Умный дом: управление климатом”). Рассмотрим задачу управления климатом на основе примера 1.1 “Умный дом” (с. 25). UML-диаграмма прецедентов на рис. 2.5 описывает использование операции подписки агентами при управлении климатом и освещенностью. Агенты климатического оборудования могут отслеживать количество людей в заданном помещении. Этот параметр определяется количеством описаний людей в ИП — агенты пользователей публикуют такое описание при входе пользователя в помещение и удаляют его при выходе из помещения. Управление оборудованием использует значения текущих показателей от агентов сенсорного оборудования, для чего организуются дополнительные подписки. На основе поступающих изменений от сенсоров агенты климатического оборудования настраивают требуемые климатические параметры, создавая благоприятные условия для пользователей в помещении.

Для высокоуровневого способа разработки программной логики агента необходимо использовать предлагаемую специализированную модель взаимодействия, поддерживающую онтологические объекты проблемной

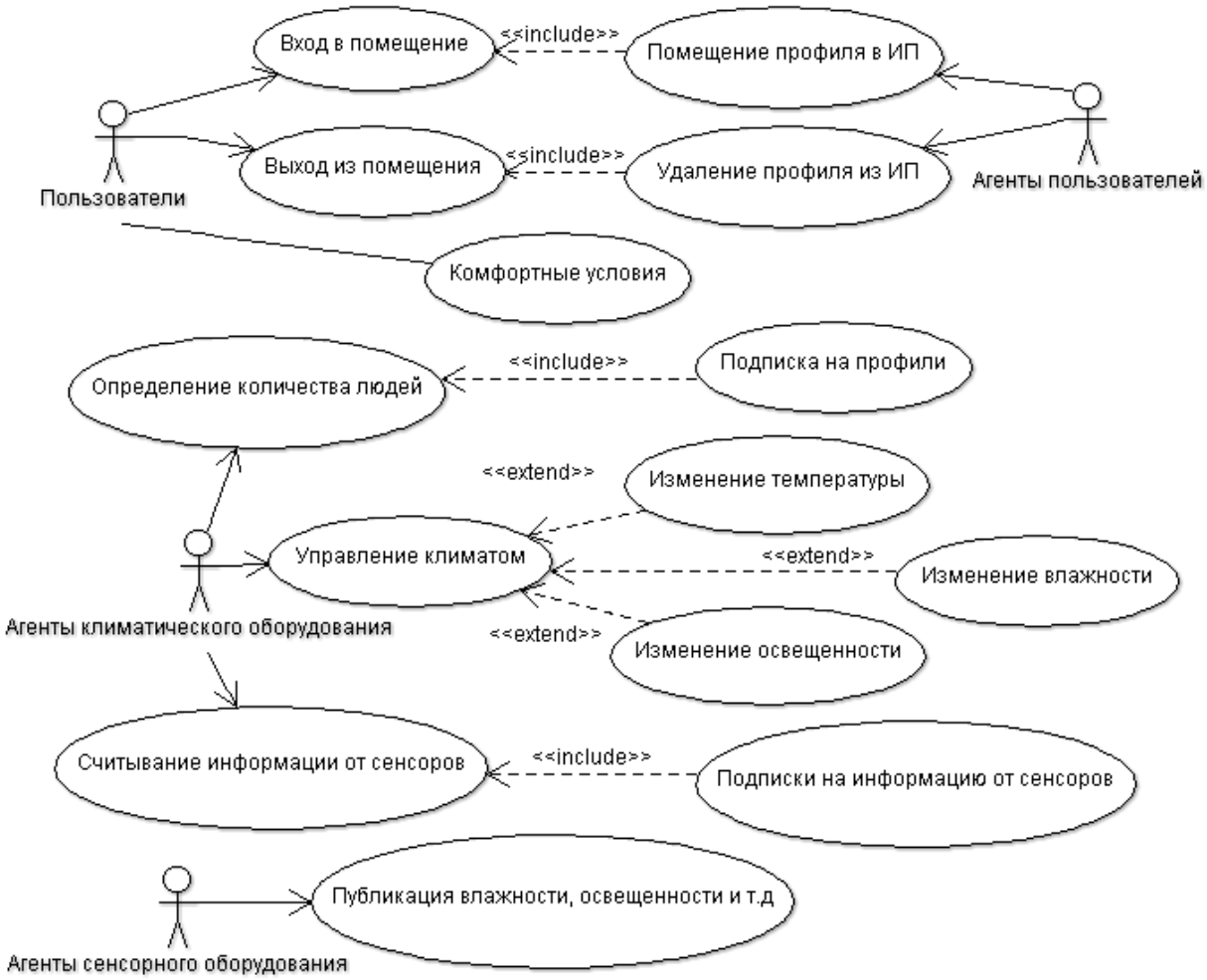


Рис. 2.5. UML-диаграмма прецедентов для использования операции подписки в примере управления климатом в помещении.

области. Предлагаемая модель наследует три группы участников из базовой модели взаимодействия: 1) подписчики, заинтересованные в получении информации, 2) издатели, публикующие информацию и 3) сервис-посредники, уведомляющие подписчиков о об изменениях информации. В базовой модели издателями и подписчиками являются агенты, а сервис-посредником выступает само ИП.

В предлагаемой модели также определяются операции для трансляции данных, учитывающие специфику подписываемой и получаемой ин-

формации при взаимодействии агента в ИП (RDF-тройки и объекты онтологической модели). Модель позволяет использовать структуры данных для объектов онтологической модели из части l_{data} онтологической библиотеки. Агент-подписчик использует объекты онтологической модели для подписывания на изменения информационного содержимого ИП. На основе объектов онтологической библиотекой (часть l_{func}) автоматически формируется поисковый запрос для выборки RDF-троек и передается в ИП. Запрос сохраняется на уровне общего хранилища RDF-троек. При изменении части информационного содержимого, попадающего под такой запрос, агент-подписчик получает уведомления. Последние содержат RDF-тройки, на основе которых автоматически обновляются хранимые на стороне агента-подписчика объекты онтологической модели.

Предлагаемая модель взаимодействия используется агентом как $(O_{\text{src}}, D_{\text{sub}}, D_{\text{ind}})$, где O_{src} — множество объектов онтологической модели для отслеживания в ИП, D_{sub} — множество RDF-представлений отслеживаемых объектов в формате ИП-интерфейса, D_{ind} — множество RDF-представлений поступающих изменений в формате ИП-интерфейса. При программировании логики агента разработчику нужно оперировать непосредственно только с объектами из O_{src}, обновление которых на стороне агента выполняется автоматически. Множества RDF-представлений D_{sub} и D_{ind} скрыты от разработчика. В модели определены операции *SUBDATA* и *UPDATE*. Операция $SUBDATA(O_{\text{src}})$ транслирует объекты O_{src} в представление D_{sub} для подписки на уровне ИП-интерфейса. Операция $UPDATE(D_{\text{ind}})$ обновляет объекты O_{src} на основе представлений D_{ind}. Параметрами операции подписки становятся объекты из O_{src}. В результа-

те, разработчик оперирует небольшим количеством объектов O_{src} вместо множества RDF-представлений ИП-интерфейса ($|D_{\text{ind}}| \gg |O_{\text{src}}|$).

Таким образом, модель позволяет использовать объекты онтологической модели при подписывании на изменения информационного содержимого. Обновление подписанных объектов происходит автоматически на стороне агента, синхронизируя их с информационным содержимым ИП. В результате, уменьшается объем программируемого разработчиком кода в логике агента. Разработка ведется на уровне объектов проблемной области, а низкоуровневые операции с RDF-тройками и сетевое взаимодействие с ИП (получение и обработка уведомлений) выполняются автоматически.

2.4. Модель взаимодействия агентов на основе локальной обработки группы объектов для формирования запросов на множественные изменения в интеллектуальном пространстве

Предлагаемая модель направлена на решения базовой задачи обработки объектов с внесением множественных изменений в информационное содержимое. В результате, разработчик сможет реализовать основной тип взаимодействия “глобальный посредник” (см. п. 1.1 на с. 32).

Предлагаемая модель позволяет программировать правило обработки для определения зависимостей между объектами проблемной области и формировать общий запрос на изменения в ИП. Для решения используется подход, сходный с трансляцией исходного запроса (напр., для получения данных) в другой формат или в набор запросов (напр., для получения информации из разных источников) [72]. В предлагаемой модели

определяется 1) событие, отслеживаемое агентом, 2) правило обработки, содержащее операции над объектами онтологической модели и 3) конечный запрос, формируемый правилом обработки. В логике агента наступление события может быть определено а) удаленно — при анализе информационного содержимого (посредством семантических запросов) или получения уведомления об изменениях в ИП по подписке и б) локально — при обнаружении изменений в работе устройства, окружающей среды (посредством сенсоров) или локальной информации (пользователь персонального агента поменял свой профиль). При наступлении события агент выполняет ряд действия, программируемых разработчиком в правиле обработки для автоматического формирования нового запроса на изменение информационного содержимого ИП. Запрос отражает реакцию агента на изменение текущего контекста работы системы. Таким образом, взаимодействие агентов определяется как множество событий и правил их обработки.

Пример 2.3 (“Умный дом: управление группой климатического оборудования”). Расширим пример 2.2 “Умный дом: управление климатом” (с. 54) на основе взаимодействия агентов климатического оборудования через агента-посредника, которому делегируется анализ контекстной ситуации в помещении. В этом случае, агенты климатического оборудования получают уведомления от агента-посредника с указаниями по управлению работой оборудования. Контекстная ситуация складывается из текущих значений параметров оборудования и показателей сенсоров. Частная онтологическая модель для агента-посредника включает объекты из двух онтологий: описание сенсоров и климатического оборудования. Агент-посредник связывает параметры оборудования и показатели сенсоров в

одну группу. Так, изменение количества людей в помещении (сенсор) может потребовать изменения поддерживаемой влажности или скорости работы вентилятора (климатическое оборудование). Агент-посредник реализует обработку таких зависимостей. Если в ИП изменяется объект сенсора, то агент-посредник изменяет в ИП значения других объектов, что в свою очередь активирует агентов климатического оборудования.

Предлагаемая модель взаимодействия используется агентом как (e, F, q, r), где e — отслеживаемое событие (обнаруживается локально или как изменение в ИП), F — множество возможных операции над объектами, q — правило обработки, привязывающее операции к объектам, r — набор {операция-объект} (запрос на обработку информационного содержимого).

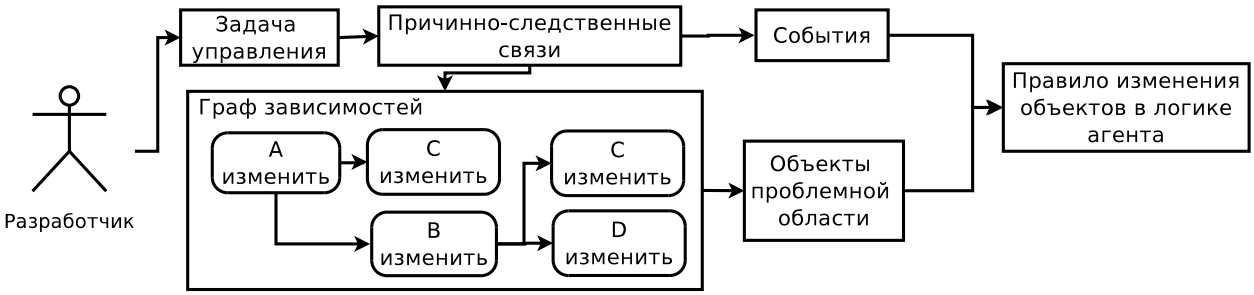


Рис. 2.6. Построение правила обработки объектов проблемной области.

Сценарии использования ИП основаны на причинно-следственных связях в проблемной области. Схема на рис. 2.6 показывает построение правила изменения объектов онтологической модели. Определяются события (причины) и граф зависимостей, в котором представлены цепочки обработки группы объектов проблемной области (следствия). Для заданного события e в логике агента программируется правило обработки q. Такое правило реализует обработку группы объектов (на стороне агента), формируя в итоге запрос r с операциями из F для изменения агентом объектов

в ИП. С учетом параллельной активности других агентов операции из r надо объединить в один сетевой запрос от агента к ИП для выполнения транзакции на внесение множественных изменений.

Отслеживание события и обработка группы объектов показана на рис. 2.7. Агент B изменяет информационное содержимое ИП. Агент A отслеживает наступление события e (напр., по подписке). При наступлении события e запускается правило обработки q, которое использует операции из F для формирования запроса r. Все операции из запроса r объединяются в одну транзакцию, которая выполняется при помощи ИП-интерфейса. В результате объединения уменьшается количество сетевых операции к ИП, а выполнение операции неделимым образом гарантирует семантическую целостность изменений информационного содержимого ИП.

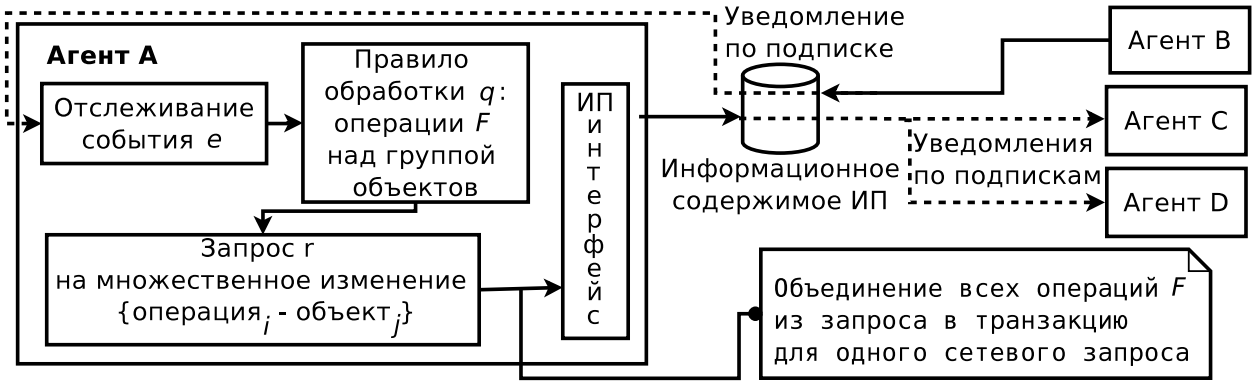


Рис. 2.7. Схема использования правила для обработки группы свойств.

Таким образом, модель позволяет программировать правило обработки, используя операции над объектами проблемной области как реакцию на отслеживаемое агентом событие. На основе правила обработки автоматически формируется запрос на множественное изменение, объединяющий отдельные сетевые операции, что снижает сетевую нагрузку. В результа-

те, разработчик проектирует взаимодействие агента как набор событий и реакций на события в виде изменений объектов онтологической модели.

2.5. Выводы

Предложенный метод программирования косвенного взаимодействия агентов основан на генерации промежуточного ПО, реализуемого в виде онтологической библиотеки для программирования логики агента в терминах проблемной области. Метод поддерживает формирование частной онтологической модели, которая включает только необходимые агенту для взаимодействия в ИП объекты проблемной области. В результате, уменьшается объем программного кода в части со структурами данных онтологической библиотеки. В методе используются предложенные специализированные модели взаимодействия агентов. В отличие от базовых моделей косвенного взаимодействия они поддерживаю взаимодействие с использованием объектов онтологической модели и ориентированных на автоматизацию решения базовых задач программирования логики агента. На основе метода и моделей будут разработаны соответствующие механизмы, которые позволят повысить эффективность разработки и упростить сопровождение программного кода агента.

3. Механизмы программирования косвенного взаимодействия агентов

В главе представлены разработанные механизмы программирования, требуемые предложенным методом программирования косвенного взаимодействия агентов. Механизм генерации программного кода онтологической библиотеки позволяет разработчику агента формировать частную онтологическую модель за счет выбора объектов из онтологий и на ее основе генерировать программный код на целевом языке программирования. Механизм программирования логики агента на основе модели многоэлементной сессии позволяет разработчику использовать сессии и операции управления для переключения состояния (есть/нет подключение к ИП) сетевого сеанса и подписки в программном коде логики агента. Механизм программирования логики агента на основе операции подписки позволяет разработчику определять объекты онтологической модели для отслеживания их изменений в информационном содержимом ИП. Механизм программирования логики агента на основе локальной обработки группы объектов позволяет определять правило обработки в дополнительном свойстве объекта онтологической модели для множественного изменения свойств объектов.

Результаты автора, представленные в данной главе, опубликованы в [60, 61, 7].

3.1. Механизм генерации программного кода онтологической библиотеки по онтологиям проблемной области

В предлагаемом методе программирования разработчик выбирает объекты онтологической модели для взаимодействия в ИП, а части l_{data} онтологической библиотеки генерируется на основе сделанного выбора (см. п. 2.1 на с. 40). Предлагаемый механизм предоставляет разработчику средство выбора объектов для формирования частной онтологической модели и средство генерации программного кода онтологической библиотеки. Последний автоматически генерирует программный код части l_{data}. Таким образом, в методе автоматизируется решение базовой задачи программирования логики агента — представление объектов проблемной области в программном коде (см. табл. 1.4 на с.36).

При использовании механизма разработчик выполняет следующие три шага: 1) построение общей модели проблемной области на основе онтологий $O = o_1 + o_2 + ... + o_n$ при помощи инструмента Protégé, 2) формирование частной онтологической модели m для агента на основе выбранных разработчиком объектов из O, 3) генерация программного кода части l_{data} онтологической библиотеки для реализации работы с m в логике агента. Шаги показаны на рис. 3.1.

Для моделирования проблемной области можно использовать известный графический инструмент Protégé [69], обладающий широкими возможностями для работы с онтологиями. Для выбора объектов разрабатывается специализированный модуль расширения функциональности (плагин) для Protégé. С помощью плагина выполняется выбор объектов на уровне клас-

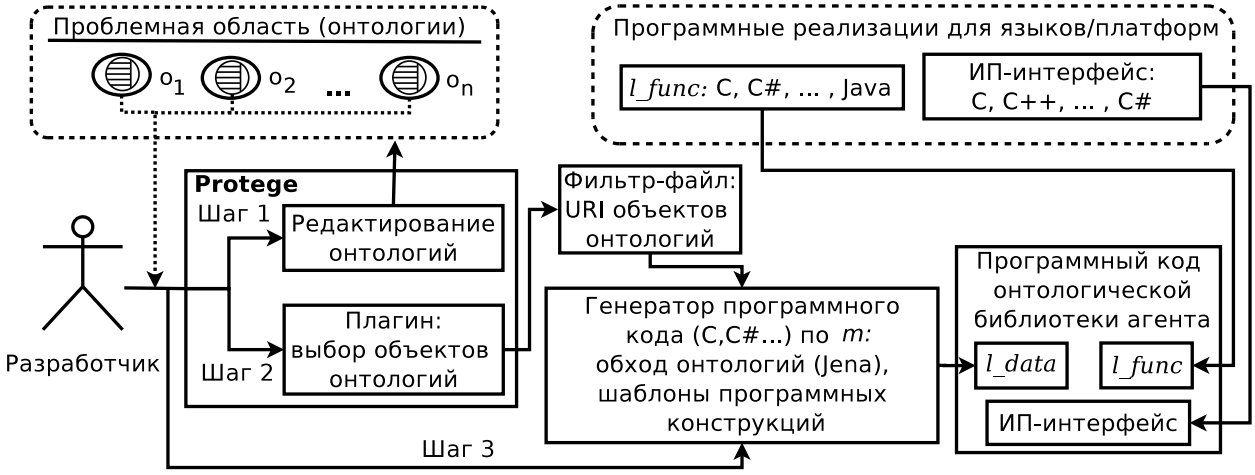


Рис. 3.1. Генерация программного кода онтологической библиотеки для агента на основе его частной онтологической модели проблемной области.

сов и свойств онтологий (включать или нет). Это позволяет уменьшить число объектов в l_{data}, снижая емкостную трудоемкость в оценках (2.1) и (2.3) из пп. 2.1 и 2.2. В предлагаемом механизме результат выбора объектов сохраняется плагином в фильтр-файле, содержащем URI объектов из онтологий. Исходные онтологии не изменяются, и разработчик может создать нескольких фильтр-файлов. Реализованный в данной работе плагин будет представлен далее в п. 4.3.

Генерация программного кода по частной онтологической модели требует специального программного средства — генератора, который выполняет трансляцию объектов из m в структуры данных и формирует вспомогательные функции на целевом языке программирования [11]. В данной работе генератор реализован на основе инструмента Jena для языков ANSI C и C#, детали реализации представлены далее в п. 4.3. Частная онтологическая модель m формируется на основе онтологий проблемной области и фильтр-файла. Генератор использует шаблоны программных конструкций для выбранного языка программирования и по m генерирует программный

код для части l_{data} онтологической библиотеки. Часть l_{func} и ИП-интерфейс выбираются из существующих программных реализаций для конкретной платформы и языка программирования.

Таким образом, разработчик логики агента применяет механизм для моделирования онтологической модели проблемной области и ее частных вариантов при помощи встроенных и расширяемых (плагин) возможностей инструмента Protégé, без необходимости использования дополнительных программных средств. Генерация программного кода выполняется автоматически. За счет применения разработчиком механизма решается одна из базовых задач программирования логики агента — представление объектов проблемной области в программном коде (см. рис. 1.4 на с. 36).

Пример 3.1 (“Умный дом: использование набора онтологий”). В примере 2.2 “Умный дом: управление климатом” (с. 54) может использоваться набор онтологий для описания:

— людей, домашних страниц и социальных сетей
(FOAF — http://foaf-project.org),
— сетевого присутствия и обмена информацией о присутствии
(Online Presence Ontology — http://online-presence.net),
— сенсоров и их параметров
(OntoSensor — http://marinemetadata.org/references/ontosensor).

Эти онтологии связаны между собой в проблемной области. Например, сенсор может определять присутствие человека в доме. В тоже время, агенты климатического оборудования не требуют для своей работы полного знания об объектах, хранящих информацию о людях в ИП (напр.,

информацию о социальных связях). Разработчик агентов климатического оборудования отбирает из FOAF-онтологии лишь минимально необходимый набор классов и свойств.

3.2. Механизм программирования взаимодействия на основе многоэлементной сессии

Для использования специализированной модели взаимодействия агентов на основе многоэлементной сессии (см. п. 2.2 на с. 45) разработан следующий механизм, который предоставляет а) способ идентификации сессии, сеанса, подписки и объектов онтологической модели и б) схему использования сессии и операций управления, которую разработчик применяет для программирования логики агента.

В модели взаимодействия сеансы, подписки и объекты онтологической должны быть объединены в одной сессии, что требует уникальных идентификаторов. На основе идентификаторов при регистрации (операция REG) выполняется ассоциация сеанса, подписки и объекта с конкретной сессией. Во время жизненного цикла сеанса, подписки и объекта возможна только одна ассоциация с одной сессией. Идентификаторы также служат для связи: а) сеанса и подписок (в модели подписки привязаны к конкретному сеансу), б) подписок и объектов (обновление объектов при получении уведомлений в локальном хранилище).

Идентификаторы для сеансов задаются разработчиком при программировании взаимодействия в логике агента. Онтологическая библиотека автоматически генерирует идентификаторы для подписок при их регистрации и связывает с сеансом, на основе которого создана подписка. Для объ-

ектов онтологической модели идентификатором служит URI. Для классов и свойств URI определяется онтологиями, на основе которых формируется частная онтологическая модель. Для индивидов URI задается разработчиком, либо генерируется онтологической библиотекой на основе URI родительского класса. Сессия также имеет уникальный идентификатор, генерируемый при ее создании онтологической библиотекой. В результате, агент использует набор уникально-идентифицируемых сессий, сеансов, подписок и объектов онтологической модели.

В механизме базовым генерируемым идентификатором для сессий, подписок и индивидов является UUID. Для индивида идентификатор формируется заменой имени родительского класса в URI на UUID. При необходимости, возможно использовать другие способы генерации уникального идентификатора, если это требуется спецификой системы, в которой работает агент, или программно-аппаратного вычислительного устройства.

Механизм определяет следующую схему, применяемую разработчиком при программировании взаимодействия на основе сессии (см. рис. 3.2).

Шаг 1. Создание сессии s и регистрация в ней необходимых наборов сеансов и подписок. Структура локального хранилища агента определяется частной онтологической моделью, для этого объекты регистрируются в сессии. Регистрация выполняется операцией $REG(s, x)$. Если в сессии нет зарегистрированных сеансов, то доступ к ИП невозможен. При отсутствии зарегистрированных классов и свойств нельзя создавать индивидов и задавать для них свойства. Классы и свойства обычно используются на протяжении всего времени работы агента, поэтому регистрируются в начале. Это не является обязательным условием — сеансы, подписки, классы

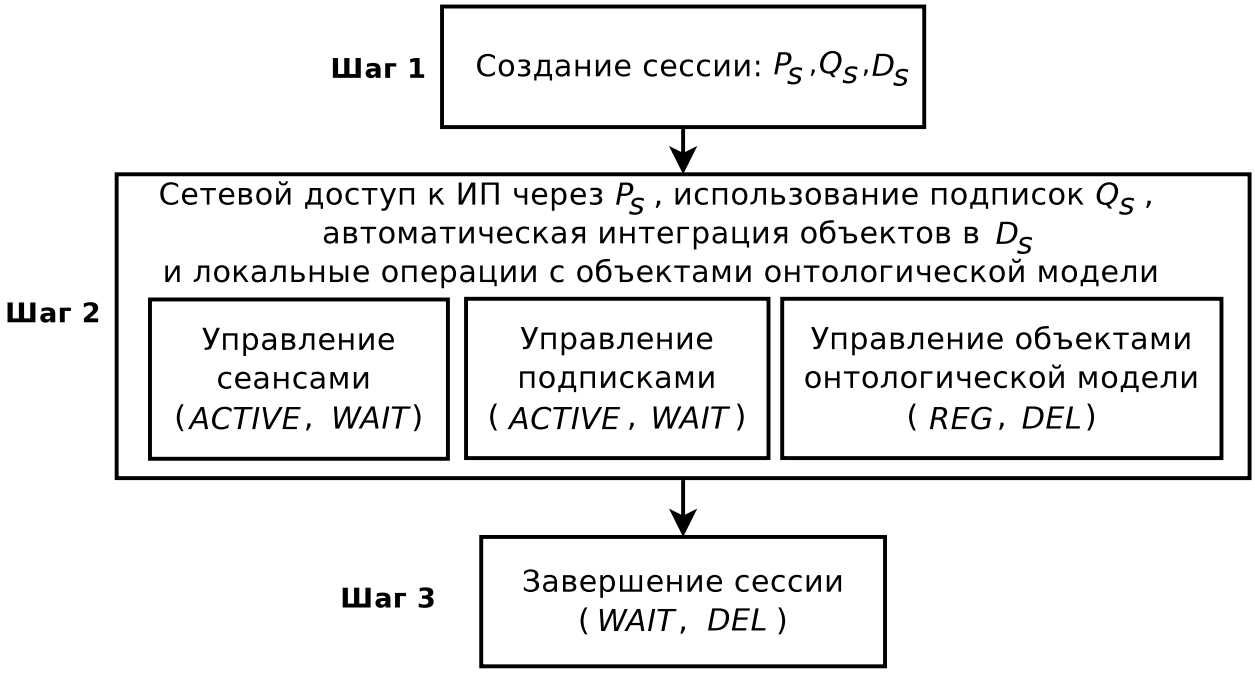


Рис. 3.2. Схема разработки агента с использованием сессии.

и свойства могут быть изменены на шаге 2.

Шаг 2. Программирование логики агента с использованием операций управления: активация $ACTIVE(s, x)$, ожидание $WAIT(s, x)$ сеансов и подписок, регистрация $REG(s, x)$ и удаление $DEL(s, x)$ объектов из локального хранилища. Запросы к ИП используют зарегистрированные в сессии объекты. Интеграция поступающих из ИП объектов выполняется автоматически в локальном хранилище на основе зарегистрированных объектов частной онтологической модели. Для этого анализируются связи между классами, свойствами, индивидами и определяются, какие связи и индивидов нужно создать/удалить/обновить в локальном хранилище.

Шаг 3. Завершение сессии: перевод в состояние ожидания всех подписок, сеансов и их удаление, включая объекты онтологической модели. Для этого используются операции для перевода сессии в состояние ожидания ($WAIT(s)$) и для удаления ($DEL(s, x)$) сеансов, подписок и объектов.

Программный код для регистрации объектов онтологической модели на шага 1 может быть получен при автоматической генерации части l_{data} онтологической библиотеки (см. п. 3.1 на с. 63). В результате, на шаге 1 разработчику нужно вызвать соответствующую функцию, которая проведет регистрацию всех объектов частной онтологической модели агента.

Предлагаемый механизм поддерживает следующие способы использования нескольких сессий:

1) активация-ожидание;
2) активация-завершение;
3) смешанный.

В способе 1 неиспользуемые сессии переводятся в состояние ожидания и активируются при необходимости. В способе 2 сессия всегда завершается после работы с ней: из сессии удаляются все сеансы, подписки и объекты онтологической модели. Такой способ подходит, если нахождение сессии в состоянии ожидания неоправданно с точки зрения затрачиваемых ресурсов вычислительного устройства. Способ 3 комбинирует два предыдущих. Например, агент может использовать часть сессий с постоянным набором сеансов и подписок и часть создавать для разовых взаимодействий с ИП.

Так как в модели взаимодействия подписки связаны с сеансом и определены операции управления, то это позволяет организовывать группы подписок. Использование нескольких сеансов с различными наборами подписок позволяет активировать и останавливать часть подписок при определенных условиях. В примере 2.1 “Умный дом: переключение между ИП” (с. 46) нужно переводить часть подписок в состояние ожидания при выходе

из дома и активировать их снова при возвращении агента в дом. Для этого в рамках одного из сеансов задаются нужные подписки и применяются операции $ACTIVE(s, p)$ и $WAIT(s, p)$.

Таким образом, разработчик агента применяет механизм для управления сессией, сеансами и подписками, определяя логику переключения между состояниями при программировании логики агента. Интеграция информации в структурах данных для представления объектов онтологической модели проводится автоматически. Это решает базовую задачу сетевого доступа к нескольким ИП с интеграцией информации и позволяет разработчику реализовать основной тип взаимодействия "концентратор информации" (см. п. 1.1 на с. 32).

Пример 3.2 ("Умный дом: управление состоянием"). В примере 1.1 "Умный дом" (с. 25) сессия может использоваться разработчиком для управления состоянием сеансов и подписок при разработке агентов на мобильных устройствах людей. Для устройства (напр., мобильного телефона) в состоянии ожидания часть подписок обрабатывать не нужно (напр., для отслеживания названия мелодии на музыкальном проигрывателе) и они также переводятся в состояние ожидания. Когда устройство активируется, тогда восстанавливаются сеансы и подписки.

3.3. Механизм программирования взаимодействия на основе операции подписки

Для использования модели взаимодействия агентов на основе операции подписки (п. 2.3 на с. 53) разработан следующий механизм, задающий формат данных, используемый при подписании и обработке уведомлений.

Механизм предоставляет разработчику два вида операции подписки для определения подписываемой информации в терминах проблемной области при программировании взаимодействия в логике агента.

Объекты онтологической модели в информационном хранилище ИП представлены в виде RDF-троек. Пример такого представления показан на рис. 3.3. Связи с другими индивидами задаются также через RDF-тройки — свойства-объекты. Подписывание на изменения RDF-троек позволяет проводить синхронизацию объекта онтологической модели, хранимого агентом локально, с информационным содержимым ИП.

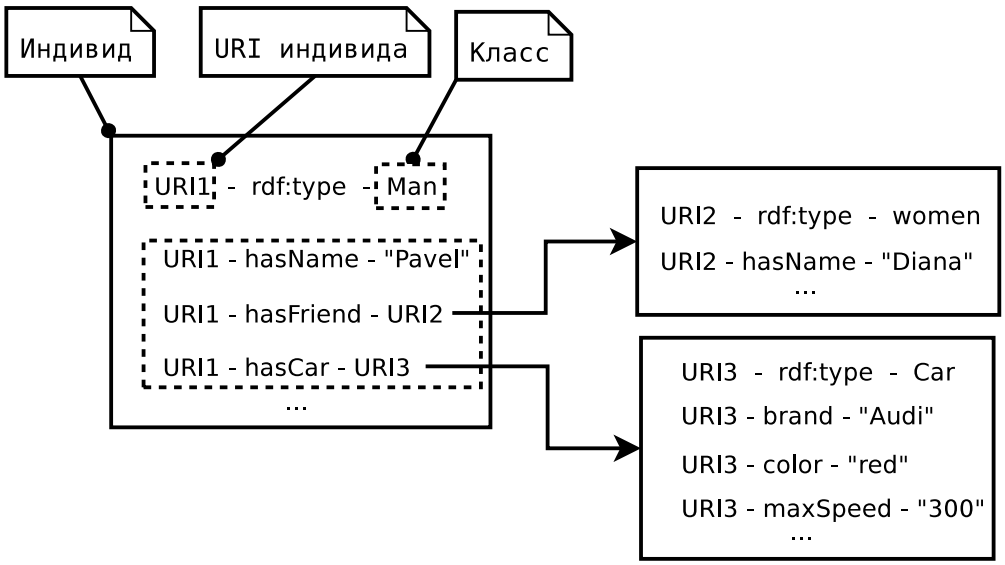


Рис. 3.3. Представление индивида в виде набора RDF-троек.

С учетом требований модели взаимодействия на основе подписки механизм предоставляет разработчику представленную на рис. 3.4 схему работы операции подписки. Таким образом, для подписывания на изменения информационного содержимого (момент времени t_1) разработчик использует объекты частной онтологической модели, а синхронизация этих объектов выполняется автоматически при получении уведомлений об измене-

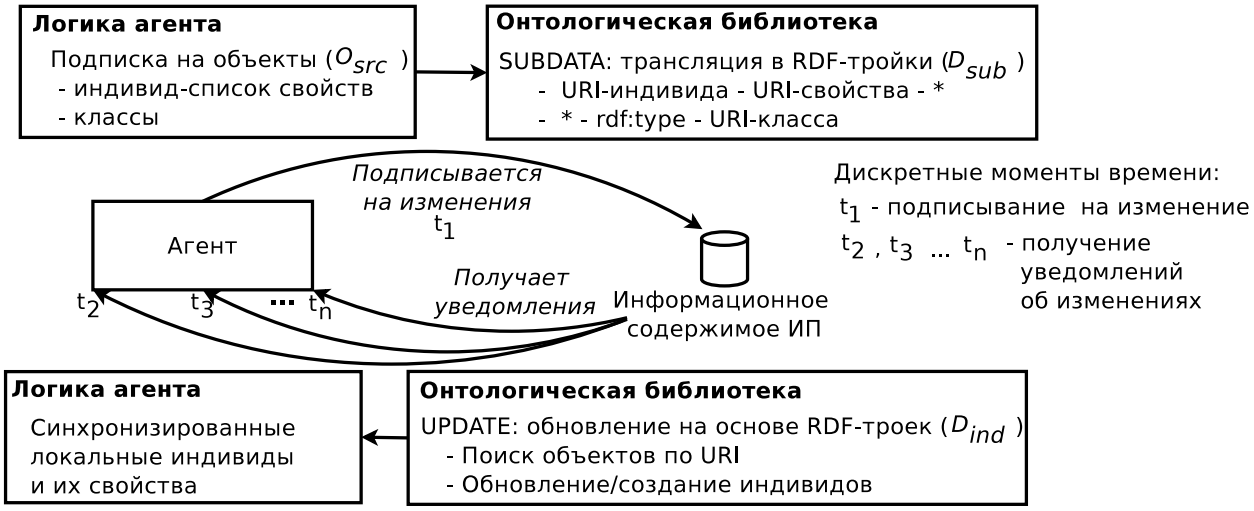


Рис. 3.4. Схема работы подписки для объектов онтологической модели.

ниях в ИП (моменты времени $t_2, t_{sd}...t_n$).

В механизме объекты O_{src} задают объекты (индивиды и их свойства) для подписания на изменения, они определяются разработчиком в логике агента. Формат RDF-представления D_{sub} задается T–шаблонами. T–шаблон — это RDF-тройка, где элементы могут быть маской. Для операций из модели взаимодействия *SUBDATA* и *UPDATE* в механизме задается формат параметров и результатов их работы. Данные операции реализуются в онтологической библиотеке. Операция *SUBDATA* преобразует объекты онтологической модели O_{src} в набор T–шаблонов (D_{sub}). Операция *UPDATE* обрабатывает полученные RDF-представления изменений D_{ind} (наборы RDF–троек). В механизме обработка RDF-троек выполняется автоматически и включает: 1) поиск индивидов по URI из RDF-троек, 2) создание локальных индивидов, если таковые не найдены и 3) обновление индивидов — установка значений для свойств.

Механизм поддерживает два вида отслеживания объектов O_{src}: 1) подписка на свойства (изменение свойств индивидов) и 2) подписка на

классы (появление/удаление индивидов). Для этого в механизме определены два вида параметров для операции *SUBDATA*. Для подписки на свойства объекта множество O_{src} задается набором пар “индивид–свойства”. Операция *SUBDATA* преобразует набор пар в список Т-шаблонов вида:

URI-индивида – URI-свойства – *

Если для подписывания на изменения в ИП используется вышеприведенный Т-шаблон, то получаемые с уведомлением об изменениях RDF-тройки, будут содержать субъект и предикат из Т-шаблонов. По субъекту определяется индивид для обновления (URI-индивида), по предикату (URI-свойства) — свойство, а объект содержит значение для свойства.

Подписка на классы отслеживает появление или удаление индивидов заданного класса в ИП. Множество O_{src} задается как набор классов. Операция *SUBDATA* преобразует набор классов в список Т-шаблонов вида:

* – rdf:type – URI-класса

Предикат rdf:type определен спецификацией RDF и описывает тип объекта. Для индивида предикат устанавливает связь с родительским классом. Каждый раз, когда индивид помещается или удаляется из ИП, агенту по подписке на класс будут приходить уведомления. По субъекту в RDF-тройке (URI-индивида) определяется соответствующий индивид.

Механизм поддерживает три способа обработки уведомлений (работа операции *UPDATE*): синхронный, асинхронный с одним потоком, асинхронный с множеством потоков. Каждый из них может быть расширен на основе функций обратного вызова (callback). Выбор способа обработки выполняется разработчиком при программировании логики агента, в

зависимости от решаемых агентом задач.

Синхронный способ не использует параллельных вычислительных потоков и применяется, когда у устройства есть ограничения на их создание. Синхронный способ требует вызова функции из части l_{func} онтологической библиотеки, которая запустит процесс проверки уведомлений об изменениях от ИП. Проверка завершается при получении изменений или по таймеру (за определенное время нет изменений).

Асинхронные способы обработки не требуют вызова функций из части l_{func}. Синхронизация локальных объектов проводится в параллельных вычислительных потоках, что требует создания дополнительных потоков (помимо потока программы агента). При асинхронном способе с одним по-

Таблица 3.1

Сравнение способов обработки уведомлений для подписок.

	Тип	Кол-во доп. потоков	Описание
1	Синхронная	Выполняется в основном потоке	Не создает дополнительных потоков. Легко обрабатывается не требовательна к ресурсам.
2	Синхронная, обратный вызов	Выполняется в основном потоке	Аналогична синхронной подписке. Для удобства, логика обработки может быть вынесена в функцию обратного вызова.
3	Асинхронная 1 поток	Один дополнительный поток	Не требовательна к ресурсам. Каждая подписка ожидает обработки других подписок, обработка подписок по кругу.
4	Асинхронная 1 поток, обратный вызов	Один дополнительный поток	Обратный вызов является синхронным - задержка обработки подписок.
5	Асинхронная множество потоков	Кол-во потоков зависит от кол-ва подписок	Для каждой подписки создается отдельный поток, нет ожидания других подписок. Требовательна к ресурсам, управление множеством потоков.
6	Асинхронная множество потоков, обратный вызов	Кол-во потоков зависит от кол-ва подписок	Обратный вызов является синхронным по отношению к потоку, в котором выполняется обработка подписки - отсутствует задержка обработки подписок.

током на стороне агента выделяется один фоновый поток для обслуживания всех активных подписок. Проверка выполняется по круговому циклу (round robin). В асинхронном способе с множеством потоков для каждой подписки на стороне агента выделяется отдельный поток выполнения. В табл. 3.1 сравниваются перечисленные способы обработки.

Отметим, что для операции подписки на свойства индивидов нужно учитывать тип свойства: свойства-данные или свойства-объекты. Свойство-данные определяет конкретное значение, например число, строку. Свойство-объект задает связь с индивидом. При обработке свойства-данных значение для свойства берется из RDF-тройки без каких-либо изменений. Для обработки свойств-объектов нужно устанавливать связь с уже хранимым локально индивидом, либо создавать нового индивида. Сравнение различных видов операции подписки представлено в табл. 3.2.

Данных видов подписки достаточно для организации действий агента при взаимодействии в ИП, аналогичных тому, что можно сделать на основе только RDF-троек. При этом упрощается программирование взаимодействия, так как для подписывания на изменения используются объекты онтологической модели, и синхронизация этих объектов выполняется автоматически.

Таким образом, разработчик агента применяет механизм для определения тех индивидов и их свойств, изменения которых необходимо отслеживать в ИП и программирует логику агента в зависимости от текущих поступающих изменений. Обработка поступающих обновлений проводится автоматически при помощи создания индивидов и изменения их свойств на основе связей, определенных в частной онтологической модели по которой

генерировался код онтологической библиотеки. Это решает базовую задачу отслеживания изменений в хранилищах информации и синхронизации изменений и позволяет разработчику реализовать основные типы взаимодействия “интеллектуальное соединение” и “централизованное хранилище” (см. п. 1.1 на с. 32).

Пример 3.3 (“Умный дом: использование подписки”). В примере 1.1 “Умный дом” (с. 25) агент может использовать подписку для отслеживания биологических параметров человека (температура, сердечный ритм, давление), представленные в ИП как свойства индивида (подписка на свойства). На основе текущих показателей и истории их изменений агент определяет для человека рекомендации (пора отдохнуть, принять лекарство).

3.4. Механизм программирования взаимодействия на основе обработки локальной группы объектов

Для использования модели взаимодействия агентов на основе локальной обработки группы объектов (п. 2.4) разработан следующий механизм,

Таблица 3.2
Виды подписок для отслеживания изменений информационного содержимого ИП.

	Вид	Описание
1	На основе RDF-троек	Манипулирует RDF-тройками и T-шаблонами.
2.1	На основе свойств-данных	Манипулирует индивидами и их свойствами.
2.2	На основе свойств-объектов	Манипулирует индивидами и их свойствами, которые ссылаются на других индивидов.
2.3	На основе свойств-данных и свойств-объектов	Манипулирует одновременно свойствами-данными и свойствами-объектами.
3	Подписка на класс	Отслеживание (появление, удаление) всех индивидов в ИП, которые соответствуют заданному классу.

реализующий дополнительное свойство, которое ассоциируется с объектом онтологической модели (только на стороне агента). Механизм предоставляет разработчику способ определения правила обработки с операциями над группой объектов, требуемыми при программировании логики агента. Для определения правила обработки механизмом предоставляется специальное свойство-обработчик. Группа объектов задается в виде индивидов и их свойств. Механизм, с учетом требований модели, поддерживает автоматическое построением запроса на множественные изменения в ИП по заданным разработчиком в правиле обработки операциям.

Разработчик использует свойство-обработчик для определения правила обработки и объявляет его в логике агента для основного объекта из группы, изменение которого приводит к изменению других объектов. Механизм определяет следующие действия, которые разработчик программирует в логике агента для группы объектов (см. рис. 3.5):

1) объявляет свойство-обработчик для основного объекта группы;
2) программирует правило обработки q как процедуру $f(e)$;
3) программирует отслеживание события e и вызов $q = f(e)$.

Свойство-обработчик определяется для объекта, онтологические свойства которого должны участвовать в правиле обработки q (объект A). Механизмом поддерживается использование в правиле обработки q других объектов и их свойства (объекты B и C). Такая поддержка необходима, так как состояние текущего объекта (его свойства) могут зависеть от состояний других объектов, аналогично может быть наоборот (состояние текущего объекта влияет на состояние других объектов).

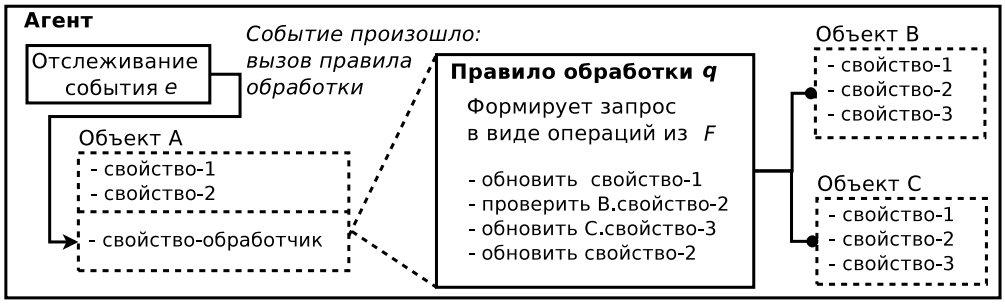


Рис. 3.5. Схема работы для обработки группы объектов при помощи свойства-обработчика.

Для формирования запроса r в правиле обработки q используются следующие операции:

$$F = \{INS, DEL, UPD, QRY\}$$

соответствующие установке, удалению, обновлению и получению свойства.

Отслеживание события e удобно реализовать на основе подписки на свойства (см. п. 3.3 с. 70), что позволит автоматически запускать обработку группы объектов. Работа со свойством-обработчиком сходна с работой с онтологическим свойствам индивида. Обновление свойства означает наступление события, значением выступает само событие. С учетом требований модели, механизмом автоматизируется формирование запроса на множественное изменение на основе правила обработки. При наступлении события, выполняются следующие шаги для обработки групп объектов на стороне агента (рис. 3.6):

1) вызов правила обработки q при помощи обновления свойства-обработчика;
2) формирование правилом q запроса r, на основе операции F;
3) преобразование запроса r в RDF-представление ИП-интерфейса;

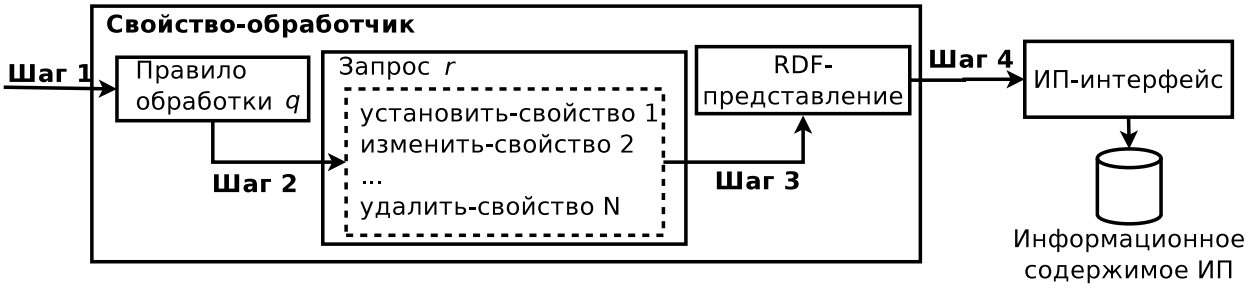


Рис. 3.6. Обработка группы объектов и формирование запроса.

4) выполнение запроса в виде одной транзакции, объединяющей все операции из r.

Таким образом, разработчик применяет механизм для определения правила обработке в свойстве-обработчике и программирует реакцию на наступление события в виде обновления свойства-обработчика. Правило обработки содержит операции над индивидами и их свойствами. На основе определенного правила автоматически формируется нужный запрос к ИП и выполняется в виде одной транзакции. Это решает базовую задачу программирования взаимодействия связанную с внесением множественных изменений в информационное содержимое и позволяет разработчику реализовать основной тип взаимодействия “глобальный посредник” (см. п. 1.1 на с. 32).

Пример 3.4 (“Умный дом: использование свойства-обработчика”). В примере 1.1 “Умный дом” (с. 25) некоторые приборы могут работать, когда дома никого нет. Агент-посредник получает информацию о числе людей — событие. Выполняет вызов правила обработки через свойство-обработчик. Правило обработки формирует запрос для добавления или удаления индивида, представляющего человека, и нового значения для количества людей. Эти операции помещаются в одну транзакцию, которая изменяет

информационное содержимое ИП.

3.5. Выводы

Представленные механизмы позволяют разработчику в логике агента программировать косвенное взаимодействие на основе предложенных специализированных моделей взаимодействия. В результате, автоматизируется разработка программного кода логики агента при решении базовых задач программирования взаимодействия.

Механизм генерации программного кода предусматривает выбор объектов из онтологий для формирования частной онтологической модели и поддерживает генерацию программного кода на целевом языке программирования для различных аппаратно-программных вычислительных устройств. Механизм программирования для логики агента на основе многоэлементной сессии включает в себя схему использования сессии и позволяет программировать логику агента в виде параллельных сеансов сетевого доступа к ИП с поддержкой автоматической интеграции информации в локальном хранилище. Механизм программирования на основе операции подписки определяет форматы данных, используемых при подписании и позволяет отслеживать изменения в информационном содержимом ИП на уровне объектов проблемной области. Механизм программирования на основе модели для локальной обработки группы объектов реализует дополнительное свойство для объекта онтологической модели, в котором определяется правило обработки группы объектов с автоматическим построением запроса на множественные изменения в ИП. Далее в работе представлена реализация разработанных механизмов в инструменте разработки агентов

для ИП развернутых на платформе Smart-M3.

4. Программное обеспечение для разработки агентов на платформе Smart-M3

В главе представлена программная реализация полученных в работе результатов, проведена апробация и сделаны выводы об их эффективности при разработке агентов. Программная реализация ориентирована на платформу Smart-M3, которая распространяется с открытым исходным кодом, поддерживает косвенное взаимодействие агентов и общее RDF-хранилище. Апробация проводится на основе, разрабатываемого при участии автора, программного инструмента SmartSlog. Последний поддерживает разработку агентов для платформы Smart-M3 на основе предложенного метода программирования взаимодействия агентов и генерации онтологических библиотек. Исследуется вопрос повышения эффективности разработки агентов для различных аппаратно-программных вычислительных платформ при использовании инструмента SmartSlog. Показана практическая эффективность на примере разработки агентов для системы интеллектуального зала SmartRoom.

Результаты, приведенные в данной главе, опубликованы в [95, 56, 62, 65, 9] и подтверждаются свидетельством о государственной регистрации программы для ЭВМ (см. прил. Б) и актами внедрения результатов диссертационной работы (см. прил. В).

4.1. Программные платформы для построения интеллектуальных пространств

Для построения интеллектуальных пространств можно использовать агентские платформы, которые реализуют программную инфраструктуру, предоставляющую возможности для работы многоагентной системы [19]. В настоящее время разработано много агентских платформ, поддерживающих различные языки программирования, способы построения многоагентной системы и модели взаимодействия агентов [19, 68]. При этом взаимодействие агентов различных агентских платформ может быть затруднено. Так, ряд известных агентских платформы (Jade, Cougaar, Jason) [29, 47, 27] основаны на технологиях Java, но только платформа Jade явно поддерживает стандарты FIPA [24]. Для других платформ (напр., платформы Cougaar) возможна разработка соответствующих плагинов. Таким образом, прямое взаимодействие не всегда может быть организовано.

Тенденции развития многоагентных систем показывают необходимость в поддержки агентами технологий Семантического веб [94, 34]. Так, для платформы Jade реализованы соответствующие плагины и библиотеки. Библиотека AgentOWL поддерживает RDF/OWL онтологии [18]. Обработка онтологий осуществляется с помощью программного инструмента Jena. Библиотека AgentOWL реализует поддержку протокола XML-RPC, что позволяет Jade-агенту возвращать информацию по запросу в RDF-формате. Поддерживается язык запросов SPARQL для взаимодействия агента со SPARQL точками доступа.

Поддержка агентами косвенного взаимодействия с использованием

технологий Семантического веб выгодна, так как позволит взаимодействовать агентам различных агентских платформ. За счет использования технологий Семантического веб возможно получать информацию из внешних RDF-хранилищ и проводить локальную интеграцию информации или публиковать её в общем хранилище для дальнейшего анализа другими участниками. Стоит отметить, что упрощается подключение к системе умных вещей характерных для IoT-сред.

Для организации косвенного взаимодействия можно использовать дополнительные платформы, которые реализуют общее хранилище информации и предоставляют соответствующие примитивы доступа к нему. Примерами таких платформ являются TripCom, SemWebSpaces, CSpaces (подробнее о платформах в [67]) и Smart-M3 [88, 75]. Программная реализация предложенных в работе механизмов выполнена для платформы Smart-M3.

Выбрана платформа Smart-M3, так как другие приведенные выше платформы имели исследовательский характер и сейчас их развитие приостановлено. Стоит отметить и то, что в разработке платформы TripCom принимала участие компания Nokia, которая позднее разработала первые версии платформы Smart-M3. Платформу Smart-M3 можно рассматривать как платформу для развертывания самостоятельных ИП в IoT-среде, так и для организации взаимодействия различных многовалентных систем или отдельных агентов через общее хранилище информации с возможностью интеграции информации из Семантического веб. Необходимо отметить, что в предлагаемом методе разработке онтологическая библиотека может использовать различные ИП-интерфейсы. Замена ИП-интерфейса позволит использовать (полностью или частично), представленные в данной главе

реализации механизмов программирования взаимодействия на других, подобных Smart-M3 платформах.

4.2. Платформа Smart-M3 и программный инструмент SmartSlog

Платформа Smart-M3 1) реализует вычислительную инфраструктуру ИП с использованием технологий Семантического веб и 2) предоставляет программный инструмент для разработки агентов. Аббревиатура M3 подчеркивает ориентацию платформы на свойства multi-device, multi-vendor и multi-domain, удовлетворяя условиям IoT-сред: независимость от вычислительного устройства, его производителя и области применения. Подробное описание, включающее принципы создания прикладных систем и развиваемые примеры таких систем, представлено в [87, 75, 23, 26].

Ключевой элемент инфраструктуры ИП — это семантический информационный брокер (далее — брокер SIB, от англ. Semantic Information Broker), являющийся точкой доступа к ИП [82]. Каждый брокер SIB управляет некоторой частью разделяемой информации (RDF-хранилищем). Брокер SIB поддерживает примитивы доступа к разделяемой информации (поместить/удалить/получить RDF-тройки), операцию подписки и семантические запросы (поддерживается язык WilburQL [58] и SPARQL [89]). Такого набора примитивов доступа и операции подписки достаточно для программирования основных типов косвенного взаимодействия (см. п. 1.4 на с. 32)

К ИП через брокера SIB динамически подключаются разнообразные устройства посредством программных агентов, называемых процессорами знаний KP (далее — агент KP, от англ. Knowledge Processor). Доступ аген-

та к информационному содержимому ИП выполняется по протоколу SSAP (Smart Space Access Protocol) [52].Последний работает поверх таких протоколов сетевого и транспортного уровня как TCP/IP. ИП-интерфейсы (см. п. 1.4), представленные KP-интерфейсами (KPI, от англ. KP interface), реализуют клиентскую часть протокола SSAP для конкретного языка программирования и вычислительной среды. Доступный на данный момент набор KP-интерфейсов представлен в Приложении в табл. А.5.

Схема развернутой инфраструктуры ИП представлена на рис. 4.1. Система создается как динамический набор агентов KP. Каждый выполняет определенный сценарий, интерпретируя некоторую, разделяемую с другими агентами KP, часть информационного содержимого ИП. Сценарий управляется наблюдаемыми изменениями в ИП, вносимыми агентами KP. Некоторые сценарии являются транзитными — их выполнение зависит от прихода и выхода агентов в ИП (напр., при изменении местоположения изменяется состав агентов и возможности для выполнения текущего сценария). Агенты могут участвовать в нескольких ИП, обмениваясь информацией, необходимой для выполнения сценария [49].

Таким образом, ИП развернутое на основе платформы Smart-M3 — это частный случай ИП, рассматриваемых в диссертационной работе. Для организации взаимодействия агентов в ИП на платформе Smart-M3 необходимо решать задачи аналогичные базовым задачам программирования логики агентов (см. рис. 1.4 на с. 36). Требуется представление объектов проблемной области на стороне агента для доступа к информационному содержимому. Для доступа агента к нескольким ИП требуется параллельный доступ к ним и поддержка интеграции информации. Для получения

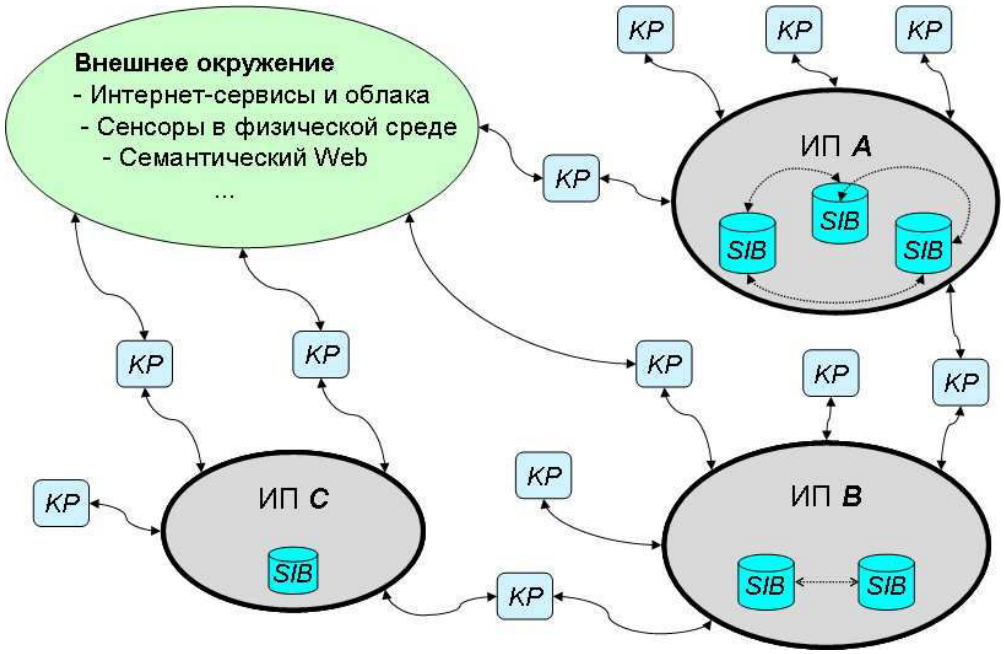


Рис. 4.1. Развернутая инфраструктура интеллектуальных пространств.

информации об изменении контекста работы системы агенту необходимо отслеживать изменения в информационном содержимом и синхронизировать локальное хранилище с изменениями в ИП. Обработка группы объектов требуется для внесения множественных изменений в информационное содержимое.

Для решения базовых задач программирования на основе OWL-представления используются высокоуровневые KP-интерфейсы, которые реализуются как промежуточное ПО для разработки агентов KP в терминах проблемной области (см. подробнее в [73]). Известны два высокоуровневых KP-интерфейса (см. строки 8, 9 в табл. А.5 Приложения). Они предоставляют только базовые функции для взаимодействия агентов в ИП и доступа к информационному содержимому. Отсутствует поддержка параллельного взаимодействия с несколькими ИП и интеграции полученной информации из распределенных источников информации на стороне аген-

та. В результате, разработчик может программировать логику агента КР в терминах проблемной области, но возможности разработки весьма ограничены из-за отсутствия поддержки специализированных моделей взаимодействия.

В диссертационной работе автором реализуются предложенные механизмы в созданном программном инструменте SmartSlog. Последний поддерживает генерацию онтологических библиотек, которые используются в качестве высокоуровневого КР-интерфейса (далее — библиотека SmartSlog). Программирование взаимодействия агентов ведется в соответствии с разработанным методом. Библиотека SmartSlog включает реализацию предложенных механизмов на основе полученных специализированных моделей взаимодействия агентов. Использование инструмента SmartSlog направлено на автоматизацию решения базовых задач программирования взаимодействия в логике агента (см. рис. 1.4 на с. 36).

В инструменте SmartSlog поддерживаются две версии онтологических библиотек в зависимости от целевой аппаратно-программной платформы: ANSI C и .NET (C#). Библиотека SmartSlog поддерживает работу различных низкоуровневых КР-интерфейсов. Для языка ANSI C используются интерфейсы KPI_low и C_KPI (см. строки 3, 4 в табл. А.5 Приложения). Они ориентированы на малопроизводительные устройства. Число используемых зависимостей от других библиотек минимизировано (необходима POSIX-библиотека вычислительных потоков и библиотека для разбора данных в XML-формате). Для поддержки Windows-устройств используется версия библиотек SmartSlog для языка C#. В инструмент SmartSlog также входит кодогенератор, преобразующий OWL-онтологии в структуры

данных для части l_{data} библиотеки SmartSlog.

В целом, инструмент SmartSlog может быть использован при разработке агентов для широкого круга аппаратно-программных вычислительных платформ, предоставляя высокоуровневые операции для программирования взаимодействия в логике агента КР. В диссертационной работе проведены эксперименты по разработке агентов КР с использованием этого инструмента для апробации и проверки эффективности полученных результатов.

4.3. Реализация механизма генерации программного кода онтологической библиотеки

Рассмотрим реализацию механизма генерации программного кода онтологической библиотеки для части, определяющей структуры данных для представления объектов частной онтологической модели агента (см. п. 3.1 на с. 63). Для генерации структур данных реализован кодогенератор CodeGen (язык Java).

Высокоуровневая архитектура кодогенератора CodeGen показана на рис. 4.2. Основным модулем является Generator, который на основе параметров (AppConfigurator) и обработчика (OntHandler) генерирует программный код. Модуль CodeModelContainer используется для хранения частей готового программного кода. Эти части используются для формирования файлов с исходным кодом. На основе обработчика OntHandler реализуются обработчики для целевых языков программирования и аппаратно-программных вычислительных платформ. Такие обработчики при помощи специальных классов TemplateResourceFile и TemplateString обрабатывают

шаблоны программного кода, на основе которых генерируется программный код.

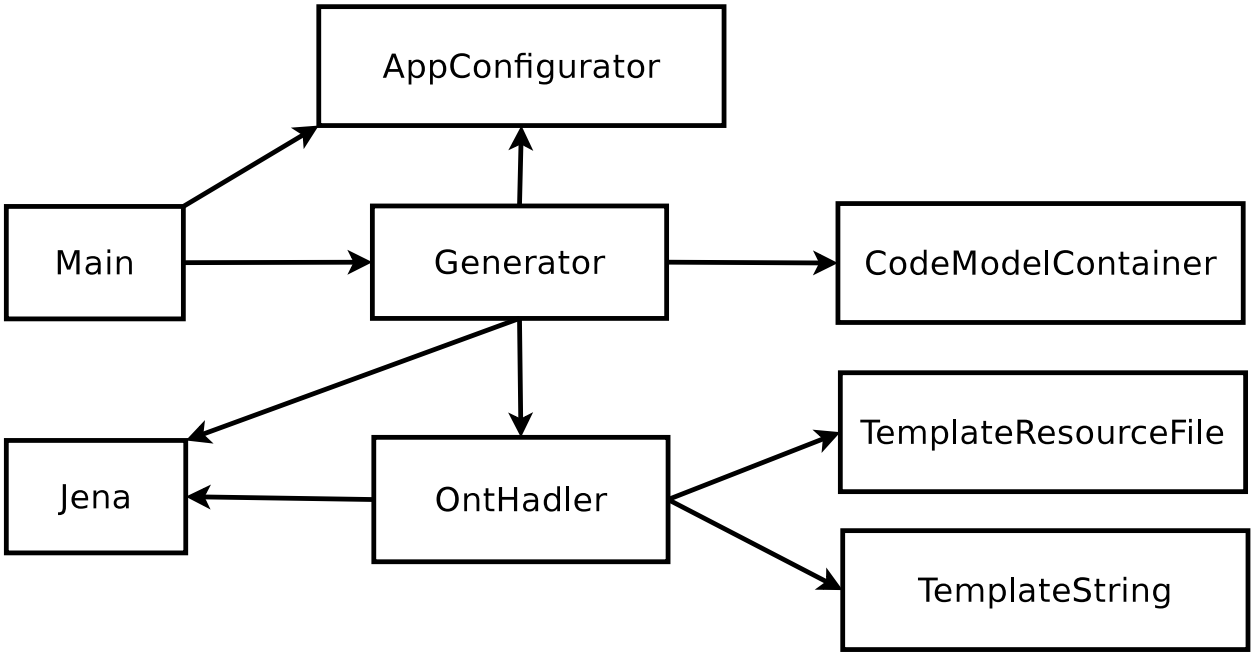


Рис. 4.2. Высокоуровневая архитектура генератора CodeGen.

Шаблоны представляют собой заготовки программного кода, представляющие OWL классы и свойства. Шаблоны определены с точностью до имени используемых элементов заданной проблемной онтологии. Шаблон содержит специальные теги ⟨name⟩ вместо реальных имен. Обработчик осуществляет преобразование одного или нескольких шаблонов в программный код, заменяя теги именами, взятыми из онтологии. Онтологии анализируются инструментом Jena [81], который представляет OWL-онтологии в виде специального графа. Генератор CodeGen выполняет обход вершин построенного графа, вызывая соответствующие вершинам обработчики. В текущей реализации генератора для языка ANSI C используется 18 шаблонов, для языка C# — 23 шаблона.

При генерации из OWL-онтологий анализируются классы, свойства,

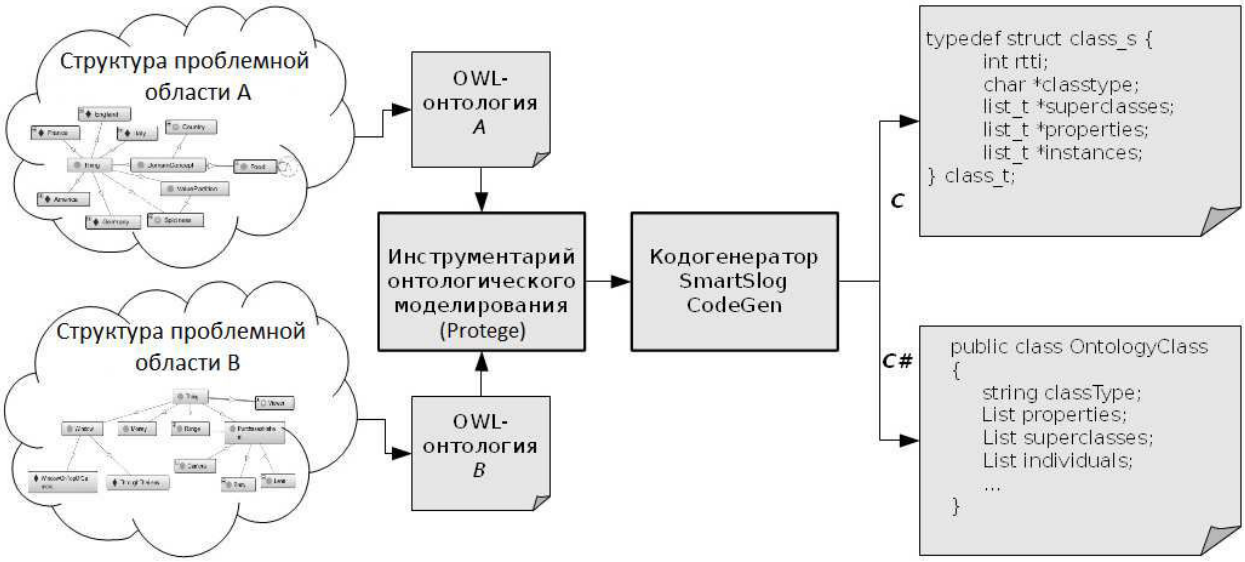


Рис. 4.3. Представление OWL-онтологии в виде структур данных и классов.

учитываются ограничения (значения свойств, принадлежность свойств к классам, кардинальность). Генерация классов и свойств в программный код основана на описательной близости онтологического и объектно-ориентированного (ОО) подходов [11]. На рис. 4.3 показаны схематичные примеры для языков ANSI C и C#. Первый является процедурным, и OWL-классы реализуются стандартными C-структурами. Второй — объектно-ориентированный, и OWL-классы реализуются в виде ОО-классов.

В разработанном механизме для выбора объектов частной онтологической модели используется авторский плагин для инструмента Protégé. Такой плагин реализован и подключается к инструменту Protégé с учетом спецификации OSGi [92]. Плагин позволяет выбрать необходимые объекты из OWL-онтологий, которые сохраняются в фильтр-файл. Интерфейс плагина показан на рис. 4.4. Разработчик выбирает нужные классы, свойства-объекты (области 1 и 2) и свойства-данные (не показаны на рис. 4.4). Та-

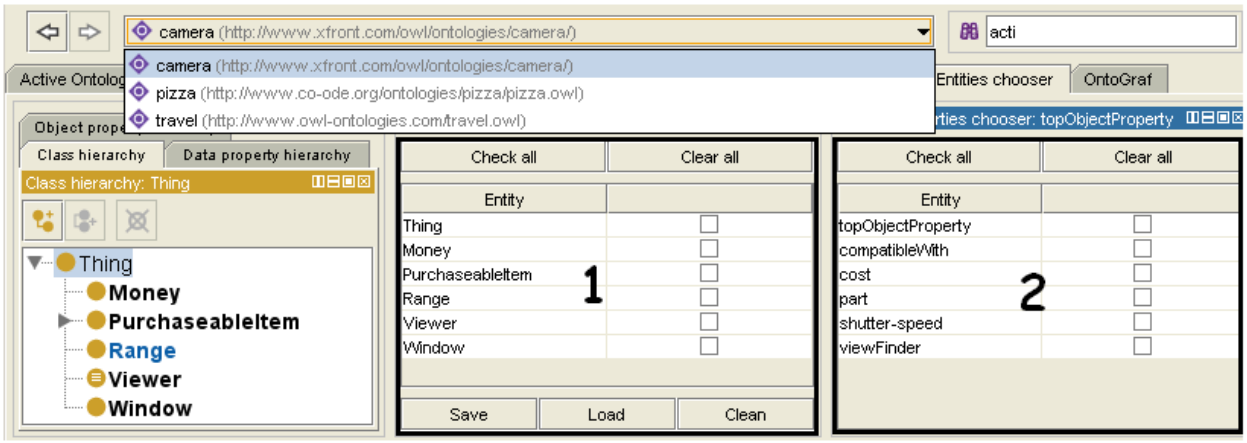


Рис. 4.4. Интерфейс плагина для формирования фильтр-файла.

ким образом, плагин реализует три элемента интерфейса Protege, которые встраиваются в общий интерфейс редактора. Для реализации таких элементов наследуются соответствующие классы:

— AbstractOWLClassViewComponent,

— AbstractOWLDataPropertyViewComponent,

— AbstractOWLObjectPropertyViewComponent.

Расположение элементов интерфейса плагина на общей форме инструмента Protege задается специальным XML-файлом. Данный файл анализируется инструментом Protege при инициализации плагина.

Фильтр-файл анализируется генератором CodeGen при обходе графа, построенного инструментом Jena. Если объект из фильтр-файла найден в графе (сравнение URI), тогда применяются необходимые шаблоны для генерации соответствующей структуры данных. Фильтр-файл имеет текстовый формат и может быть составлен вручную. Он содержит список URI объектов из OWL-онтологий и разбивается на категории в зависимости от типа объектов (см. рис. 4.5).

```
--- Classes ---
http://www.xfront.com/owl/ontologies/camera/#Range
http://www.xfront.com/owl/ontologies/camera/#Viewer
--- Object properties ---
http://www.xfront.com/owl/ontologies/camera/#viewFinder
http://www.xfront.com/owl/ontologies/camera/#part
--- Data properties ---
http://www.xfront.com/owl/ontologies/camera/#f-stop
--- Object properties ---
http://www.xfront.com/owl/ontologies/camera/#compatibleWith
...
```

Рис. 4.5. Формат фильтр-файла для хранения URI выбранных объектов.

В соответствии с предложенной структурой онтологической библиотеки инвариантная функциональная часть библиотеки SmartSlog используется для работы со структурами данных. В ней имена функций и типы аргументов не привязаны к конкретным именам из исходных OWL-онтологии. Объекты структур данных в логике агента передаются как параметры, тип которых конкретизируется лишь до общеонтологического уровня “класс”, “индивид”, “свойство”. API библиотеки SmartSlog предоставляет две основные группы функций: операции с локальными структурами данных и операции доступа к ИП. Локальные операции позволяют создавать и удалять индивидов, устанавливать и изменять их свойства без внесения изменений в информационное содержимое ИП. При выполнении операций реализована проверка на корректность (напр., кардинальность и область возможных значений свойства). Операции доступа к ИП реализуются на уровне индивидов и свойств: вставка, удаление, обновление свойств и индивидов и их получение по семантическому запросу.

4.4. Реализация механизмов программирования косвенного взаимодействия в онтологической библиотеке SmartSlog на основе специализированных моделей взаимодействия

В данном разделе приводится реализация предложенных в работе механизмов программирования для работы с сессией, операцией подписки и свойством-обработчиком. Представлены архитектурные составляющие и особенности использования реализованных механизмов.

4.4.1. Реализация механизма программирования взаимодействия на основе многоэлементной сессии

Рассмотрим программную реализацию этого механизма, представленного в п. 3.2 на с. 66, на основе объектно-ориентированной C# версии библиотеки SmartSlog. Представленная реализация механизма в библиотеке SmartSlog позволяет взаимодействовать агенту с несколькими ИП и автоматически интегрировать информацию в локальном хранилище из распределенных источников.

В библиотеке SmartSlog реализованы классы, соответствующие компонентам сессии (см. рис. 4.6), и методы в классах, соответствующие операциям управления из модели взаимодействия на основе многоэлементной сессии. Сама сессия также представлена классом (класс Session), который содержит наборы сеансов (класс Node), подписок (класс Subscription) и локальное хранилище (класс Repository), разделяемое между сеансами и подписками. Операции управления, определенные в модели (см. табл. 2.1 на с. 51), реализованы в классах: Session — операции для регистрации

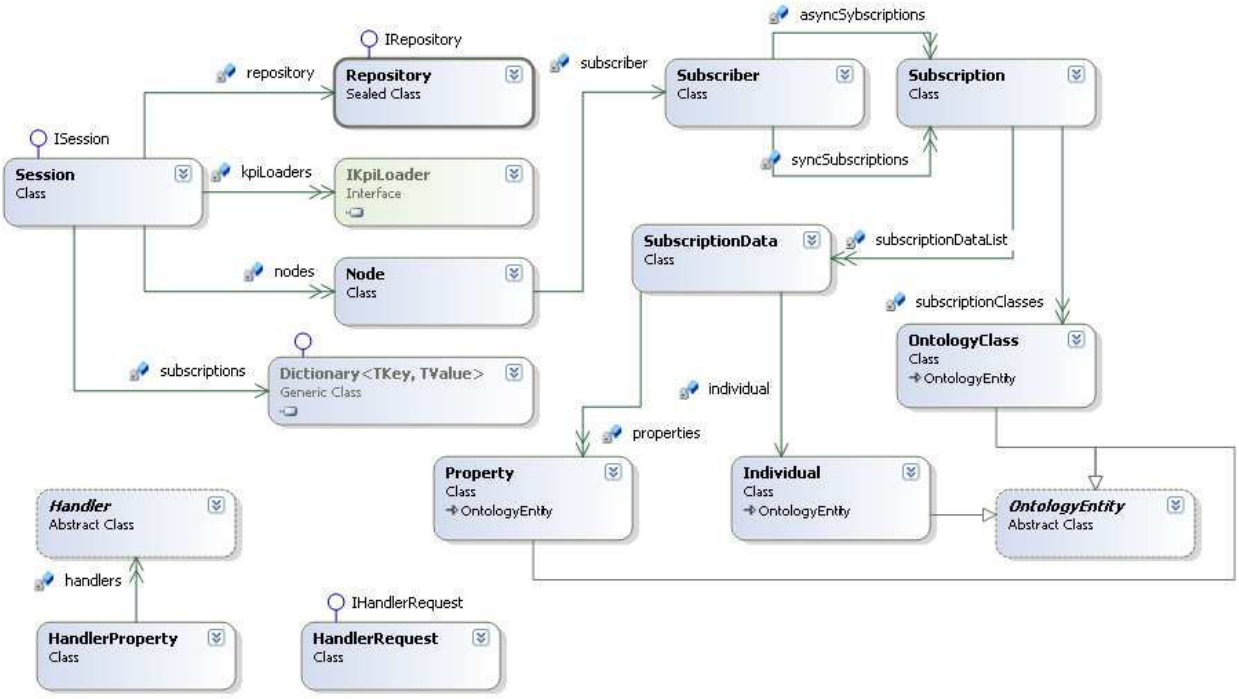


Рис. 4.6. Диаграмма основных классов для сессии, сеансов, подписок и свойства-обработчика.

и удаления (REG и DEL), Node — операции для управления состоянием ($ACTIVE$ и $WAIT$). Детали интерфейса загрузчика KP-интерфейса (IKpiLoader) будут представлены далее при рассмотрении поддержки использования различных KP-интерфейсов сессией. Классы, связанные с операцией подписки и свойством-обработчиком, будут рассмотрены далее в пп. 4.4.3 и 4.4.4. Метрики кода библиотеки SmartSlog и использованные при ее разработке инструменты приведены в табл. 4.1.

В библиотеке SmartSlog для идентификация сессии, сеансов и подписок используется UUID, как это определено в механизме программирования взаимодействия на основе многоэлементной сессии (см. п. 3.2 на с. 66). При программировании логики агента разработчик применяет схему использования сессии, так же представленную в механизме. Библио-

Таблица 4.1
Метрики кода библиотек SmartSlog и использованные инструменты для их разработки.

ANSI C	C#
18 исходный и 24 заголовочных файла	29 классов, 4 интерфейса
Строк кода: 7104 Строк комментариев: 5083 Вклад автора: 50%	Строк кода: 5068 Строк комментариев: 3678 Вклад автора: 90%
Среда разработки: Netbeans, Visual Studio 2012 Документация: Doxygen Тестирование: CUnit, Valgrind, Wireshark	Среда разработки: Visual Studio 2012 Документация: Sandcastle Help File Builder Тестирование: NUnit, Visual Studio, Network Monitor
Дополнительно: make, gcc, GNU Autotools, CCCC, CodeAnalyzer, Subversion	Дополнительно: PInvork signature toolkit, StyleCop, FxCop, CodeAnalyzer, Subversion

тека SmartSlog содержит весь код необходимый для регистрации классов и свойств из частной онтологической модели. В логике агента КР разработчику необходимо вызвать соответствующий метод и передать сессию, в которой нужно произвести регистрацию, в качестве параметра. В соответствии с определенными в модели операциями управления, классы и свойства после регистрации будут находиться в локальном хранилище сессии (класс Repository). Удаление объектов онтологической модели из сессии проводится вручную.

При удалении из сессии объектов онтологической модели необходимо учитывать их связи: классы и свойства связаны с индивидами, индивиды связаны друг с другом. Библиотека SmartSlog поддерживает следующие виды удаления: 1) класса, если в сессии не зарегистрировано индивидов данного класса, 2) свойства, если оно не задано ни для одного индивида, 3) индивида, если нет ссылающихся на него свойств-объектов, 4) индивида и ссылающиеся на него свойства-объекты.

Разработчик может создавать любое количество сессий. Сессии не связаны между собой и их работа не влияет друг на друга. Онтологические

объекты, сеансы и подписки могут быть зарегистрированы только в одной сессии, повторная регистрация запрещена.

4.4.2. Поддержка ИП-интерфейсов сессией при помощи адаптеров

Предложенная в рамках метода программирования косвенного взаимодействия схема использования онтологической библиотеки (см. п. 2.1 на с. 40) позволяет использование различных ИП-интерфейсы. В библиотеке SmartSlog такая поддержка реализована на уровне сессии на основе KPI-адаптера (далее — адаптер). Последний спроектирован и реализован автором на основе известного шаблона проектирования “Адаптер” [3]. Адаптер скрывает внутреннюю логику и предоставляет единый способ работы с KP-интерфейсами. Основные классы и интерфейсы адаптера представлены на рис. 4.7.

Интерфейсы адаптера описывают методы для работы с RDF-тройками и операциями протокола SSAP:

— создание/обработка RDF-троек, списков RDF-троек;
— вход/выход из ИП;
— добавление/удаление/обновление RDF-троек в ИП;
— выполнение SPARQL-запросов;
— подписка на изменения в ИП.

Для подключения KP-интерфейса к библиотеке SmartSlog разработчику нужно реализовать интерфейсы адаптера. На уровне адаптера используются RDF-тройки. Все необходимые преобразования между объекта-

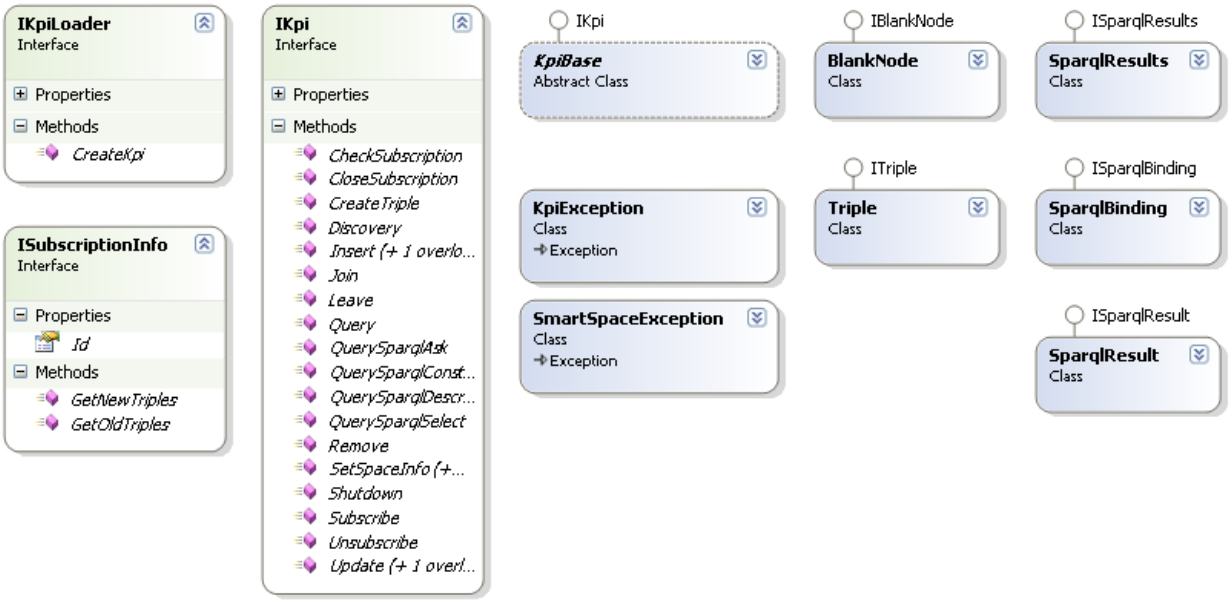


Рис. 4.7. Основные классы и интерфейсы KPI-адаптера.

ми онтологической модели и RDF-тройками выполняются непосредственно в библиотеке SmartSlog.

Поиск KP-интерфейсов происходит при создании сессии. Выполняются следующие шаги (рис. 4.8):

1) обращение к модулю поиска;
2) выполнение поиска;
3) создание экземпляров загрузчиков KP-интерфейсов;
4) предоставление KP-интерфейсов для сеансов.

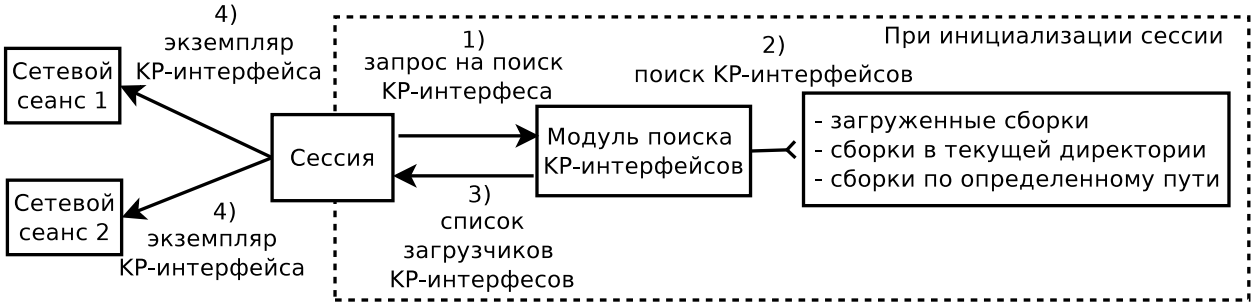


Рис. 4.8. Поиск и инициализация KP-интерфейсов при создании сессии.

Загрузчик КР-интерфейса выполняет все необходимые действия для правильной инициализации КР-интерфейса. Сеансы запрашивают у сессии КР-интерфейсы и используют их, но все детали реализации остаются скрытыми за интерфейсами адаптера.

4.4.3. Реализация механизма программирования взаимодействия на основе операции подписки

Рассмотрим программную реализацию этого механизма, представленного в п. 3.3 на с. 70. Представленная реализация механизма в библиотеке SmartSlog позволяет агенту КР получать уведомления об изменениях объектов в ИП и автоматически синхронизировать локальные объекты онтологической модели с информационным содержимым ИП.

Для представления подписки в библиотеке SmartSlog предусмотрен класс Subscription (см. рис. 4.6). Разработчик задает множество объектов онтологической модели для отслеживания их изменений в ИП (задает множество O_{src}). Механизм поддерживает два вида операции подписки, с учетом этого, разработчик может задать O_{src} как пары “индивид-свойства” (хранятся в классах SubscriptionData), как классы (хранятся как список классов OntologyClass) или как их комбинацию (пары “индивид-свойства” и классы).

Из поддерживаемых механизмом способов обработки уведомлений (см. табл. 3.2 на с. 76) в библиотеке SmartSlog реализованы асинхронный с одним потоком и синхронный способы обработки. Реализован именно асинхронный способ с одним потоком, так как является более экономичным — используется один дополнительный поток для обработки уведомлений по

подпискам. Способ обработки задается через класс Subscription.

Операции для трансляции объектов в RDF-представление ($SUBDATA$) и обновления объектов ($UPDATE$) реализованы в классе Subscriber (см. рис. 4.6). Данный класс не доступен для разработчика KP и вызывается только сеансами (класс Node). Класс Subscriber обрабатывает все подписки в текущем сеансе. При получении уведомления происходит синхронизация локального хранилища — выполняется поиск и обновление индивидов по существующим URI в соответствии с тем, как это определено в механизме программирования взаимодействия на основе операции подписки.

Необходимо пояснить, как происходит выбор действия (установить, обновить, удалить) для свойства при синхронизации индивида. В протоколе SSAP сообщения с уведомлениями содержат два набора RDF-троек: добавленные и удаленные. Если RDF-тройка для свойства локального индивида, находится в “удаленном” наборе RDF-троек, то свойство удаляется у индивида. Если в “добавленном” наборе RDF-троек, то свойство устанавливается. Если RDF-тройки с одинаковыми субъектами и предикатами, но различными объектами (значением для свойства), находятся в двух наборах, то свойство индивида обновляется.

4.4.4. Реализация механизма программирования взаимодействия на основе обработки локальной группы объектов

Рассмотрим программную реализацию этого механизма, представленного в п. 3.4 на с. 76. Представленная реализация механизма в библиотеке

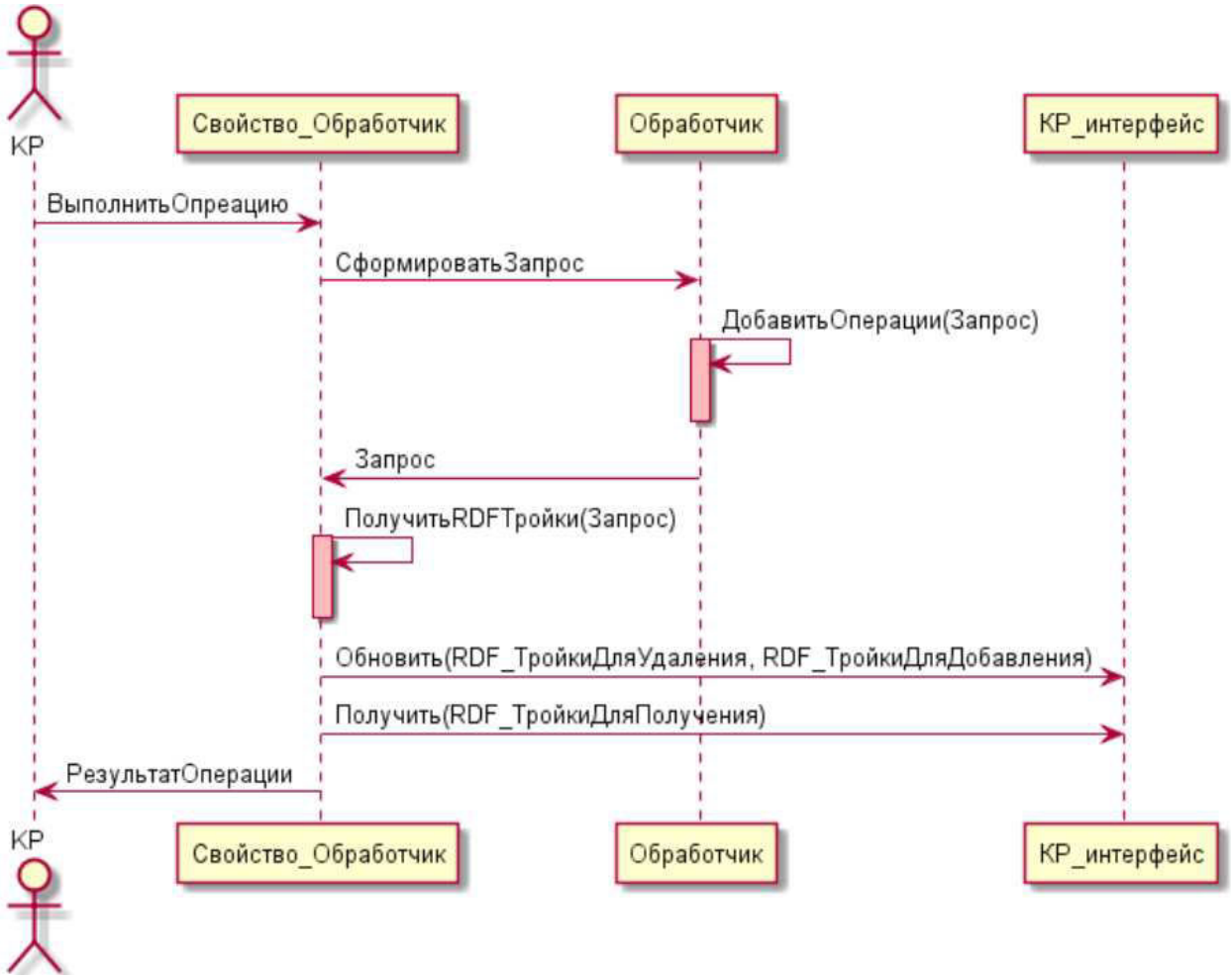


Рис. 4.9. UML-диаграмма последовательности для свойства-обработчика.

SmartSlog основана свойстве-обработчике, которое используется аналогично свойству онтологической модели.

Свойство-обработчик (HandlerProperty) может быть локально установлено, обновлено или удалено для индивида (см. рис. 4.6). Разработчик программирует правило обработки на основе класса Handler (далее — обработчик) с использованием операций (установить, удалить, обновить) над значениями свойств индивидов. UML-диаграмма на рис. 4.9 показывает последовательность работы свойства-обработчика. В логике КР выполняется операция над свойством-обработчиком (напр., обновление) и передается параметр (индивид), описывающий произошедшее событие, на которое нуж-

но отреагировать. Далее обработчик формирует запрос (HandleRequest), добавляя в него необходимые операции над индивидами их свойствами. Запрос транслируется в наборы RDF-троек для добавления, удаления или получения. Тройки для добавления и удаления объединяются в одной операции “обновление” и нужные изменения выполняются при помощи одного SSAP-запроса. После проведенных изменений выполняется SSAP-запрос на получение троек. Полученные тройки автоматически преобразуются в соответствующие объекты онтологической модели аналогично тому, как это происходит при получении уведомлений по подписке (см. п. 4.4.3).

4.5. Применение реализованных механизмов программирования взаимодействия при разработке системы интеллектуального зала

В данном разделе представлено практическое применение программного инструмента SmartSlog и реализованных в нем механизмов программирования взаимодействия. Показана апробация инструмента SmartSlog на примере системы интеллектуального зала SmartRoom [55, 41, 83] для решения задач, возникающих при взаимодействии агентов в интеллектуальном зале (ИЗ). Приводятся полученные результаты экспериментального исследования инструмента SmartSlog.

Система SmartRoom разрабатывается в ПетрГУ и предназначена для проведения научных и учебных конференций, семинаров, лекций, тренингов и других мероприятий. Она реализуется на платформе Smart-M3. Для программирования взаимодействия используется инструмент SmartSlog. Результаты использования основных возможностей инструмента SmartSlog

Таблица 4.2
Возможности инструмента SmartSlog и результаты их применения для разработки агентов интеллектуального зала.

Возможности	Результат применения
Разработка в терминах проблемной области.	Меньше программируемого кода в логики агента для взаимодействия в ИП (напр., подписка для отслеживания слайда, выступающего).
Частные онтологические модели, плагин Protege для выбора объектов.	Генерация библиотек SmartSlog из набора OWL-онтологий для описания сервисов, людей-участников, IoT-объектов.
Использование сессии параллельных сеансов и подписок.	Переключение состояния сеансов/подписок при вкл./выкл. устройства (напр., мобильного телефона), отслеживание ошибок соединения сеанса, подписки.
Повторная подписка.	Отслеживание сетевых ошибок и автоматическое восстановление подписок.
Обработка группы объектов.	Объединение запросов к ИП для уменьшения сетевой нагрузки (напр., пересчет времени выступления для участников).
Генерация библиотек SmartSlog, поддержка адаптеров.	Агенты работают на телефонах, планшетах, одноплатных компьютерах с сенсорами (Raspberry Pi).

сведены в табл. 4.2.

Схема взаимодействия агентов в ИЗ представлена на рис. 4.10. В ИЗ есть сервисы (напр., презентации, расписания), агенты на клиентских устройствах участников (напр., мобильные телефоны посетителей конференции), агент председателя и сенсорное оборудование. Используется набор онтологий для представления участника, сервиса, председателя, расписания и сенсоров. Для корректной работы агенту нужно работать с несколькими онтологиями или их частями. Разработчик использует инструмент Protégé и разработанный автором плагин. При помощи них осуществляется выбор объектов из онтологий и формирование фильтр-файлов. Фильтр-файл используются в генераторе CodeGen инструмента SmartSlog для формирования частной онтологической модели и генерации библиотек SmartSlog для агентов на различных платформах (Windows,

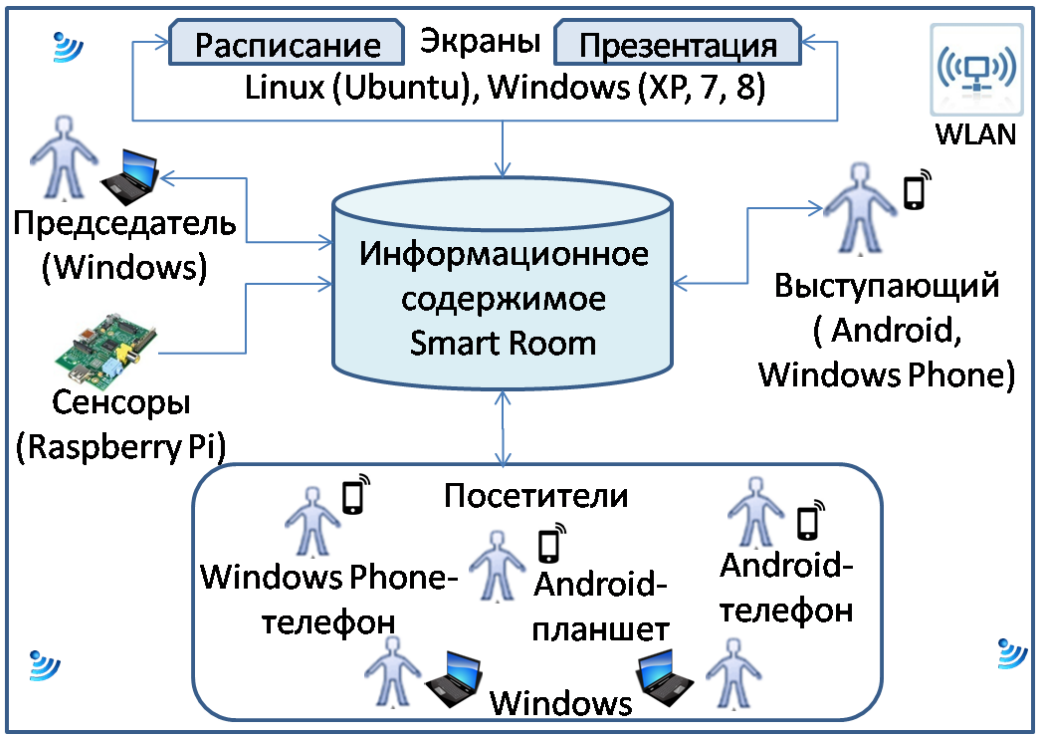


Рис. 4.10. Схема взаимодействия агентов в интеллектуальном зале.

Linux, Android).

Агенты отслеживают изменения в информационного содержимого при помощи подписок на уровне объектов проблемной области. Например, подписка на класс используется сервисом расписания для отслеживания подключения участников конференции. Клиентское устройство публикует информацию об участнике (индивид класса “Участник”), обновляя информационное содержимое. Другие агенты получают уведомления и реагируют на это, например, добавлением данных о выступающем в расписание. Отслеживание различных событий (начало выступления, завершение секции, закончилось время выступления и др.) выполняется при помощи подписки на класс “Уведомление”. Когда индивид такого класса публикуется, то подписанные агенты получают соответствующих индивидов и по их свойствам определяют детали события.

Операция подписки на свойства индивида используется, например, для отслеживания изменения слайда презентации. Агент выступающего изменяет номер слайда у индивида в информационном содержимом. Другие агенты (посетителей) получают уведомления и реагируют на них, меняя слайд.

В ИЗ используется одно ИП и агентам не нужны параллельные сеансы, но каждый агент имеет набор подписок. За счет использования сессии, разработчик может управлять группами подписок и синхронизировать нужную часть объектов в локальном хранилище в зависимости от текущего контекста работы.

Использование сессии также возможно для вре́менного отключения агента от ИЗ. Это актуально для мобильных устройств (напр., мобильных телефонов), когда такое устройство не используется, то оно может перейти в энергосберегающий режим. В этом случае сессия переходит в состояние ожидания: сетевое взаимодействие сеансов и подписок прекращается. Когда участник включает телефон, тогда сессия активируется (сеансы и подписки восстановятся) и участник продолжает взаимодействие в ИЗ. В результате такого использования экономится энергопотребление, при этом от разработчика требуется только вызывать соответствующие функции при событиях (включение/выключение телефона). Подобный вариант работы может быть использован при проблемах сетевого доступа. Например, если сеть недоступна — сессия находится в ожидании, когда сетевой доступ восстанавливается, тогда сессия активируется.

Свойства-обработчики используется для пересчета времени выступления. Такая ситуация возникает, если выступающий опаздывает. В

свойстве-обработчике разработчиком программирует пересчет времени (правило обработки), событием является отсутствие выступающего. Если событие наступает — человек не пришел, то в правиле обработки для каждого участника устанавливается новое время выступления. Выполняется один запрос на множественные изменения в информационном содержимом.

4.6. Экспериментальные исследования библиотек SmartSlog для программирования взаимодействия агентов

Использование библиотек SmartSlog позволяет уменьшить количество необходимого программного кода в логике агента. Уменьшение объема программного кода логики агента показано на рис. 4.11. Сравнивается количество операций, требуемых для реализации разработчиком действий в рамках взаимодействия с помощью библиотеки SmartSlog и на уровне RDF-представления через ИП-интерфейс (в экспериментах используется интерфейс C KPI).

В данном случае, операция — это некоторое законченное действие, например, создание индивида (библиотека SmartSlog) или троек (интерфейс C KPI). В случае библиотеки SmartSlog операция совпадает с вызовом функции (одна строчка кода). Для интерфейса C KPI операция может включать несколько строк кода (напр., операция “сохранение полученных троек” может включать в себя цикл для прохождения по всем полученным тройкам, проверку дублирования и сохранение троек). Подсчет количества операций выбран, т.к. на интерфейсе C KPI реализовать конкретную операцию можно различными способами и сравнение количества строк кода давало бы результаты в различных пределах. Важно и то, что количе-

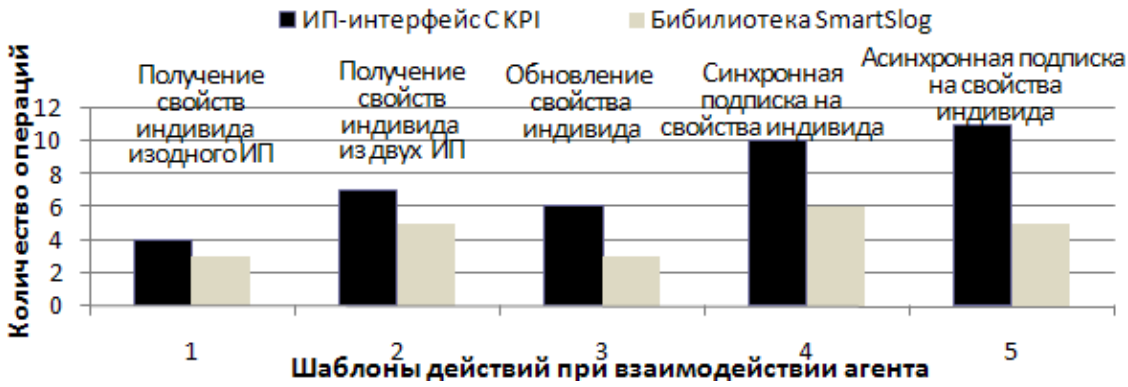


Рис. 4.11. Диаграмма количества операций для реализации действий при взаимодействии агента в ИП.

ство строк кода будет различаться от используемого языка программирования. Количество же операций остается неизменным. Например, для получения и сохранения свойств индивида, независимо от низкоуровневого ИП-интерфейса, надо: 1) создать тройки для запроса, 2) выполнить запрос, 3) проверить дублирование и 4) сохранить полученные тройки.

Диаграмма, показывающая сравнение количества операций для основных действий при программировании взаимодействия агента в ИП представлена на рис. 4.11 Выигрыш по количеству операций в логике агента в среднем равен 39%. Он достигается из-за того, что при использовании KP-интерфейса разработчику необходимо самостоятельно программировать обработку наборов RDF-троек (провести их поиск, проверку и обновление). Уменьшение объема программного кода логики агента приводит к меньшему использованию условных операторов (проверки выполнения функций, параметров для функций и др.), циклов (обходы списков RDF-троек), точек выхода из функций при ошибках. В результате, снижается цикломатическая сложность [96], что приводит к упрощению кода и меньшему число тестов для покрытия кода.

Рассмотрим результаты экспериментального исследования, связанные с производительностью библиотеки SmartSlog в сравнении с низкоуровневым интерфейсом C KPI. Необходимо проверить, на сколько снижается производительность при трансляции между RDF-тройками и структурами данных для объектов онтологической модели. При преобразовании библиотекой SmartSlog выполняется поиск нужных структур в локальном хранилище, установка связей, проверки для свойств объектов.

Исследование проводилось для обработки уведомлений по подписке с RDF-тройками, описывающими одно свойство-объект и одно свойство-данные. При использовании библиотеки SmartSlog происходило обновление структуры данных локального индивида, при этом проверялось дублирование свойств и их корректность. Для свойств-объектов создавался новый индивид, если таковой отсутствовал в локальном хранилище. В случае с интерфейсом C KPI все получаемые RDF-тройки хранились в виде списка. При добавлении новой RDF-тройки, проводилась проверка на дублирование. Исследование выполнялось на устройстве со следующими характеристиками:

— платформа x86;
— процессор Intel Core i5-2520M, 2.5 GHz;
— оперативная память 256 МБ;
— операционная система Debian.

График на рис. 4.12 показывает экспериментальную зависимость между временем и количеством локальных RDF-троек (для библиотеки SmartSlog представление такого количества троек в виде индивидов и их

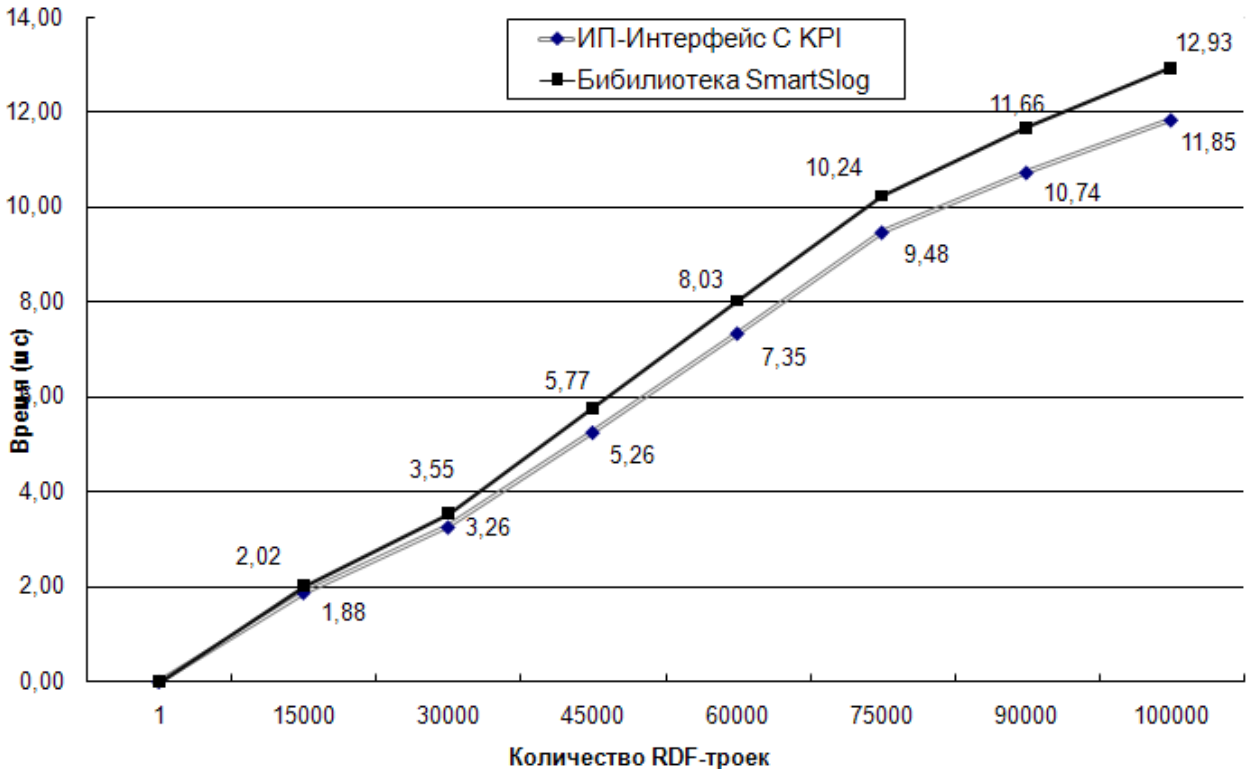


Рис. 4.12. Зависимость между временем и количеством локальных RDF-троек при обработке уведомлений по подписке.

свойств). Можно сделать вывод, что использование библиотека SmartSlog не приводит к значительным потерям производительности. В среднем снижение достигает 0.15 мс, (около 7%). В приложениях агентам единовременно необходимо меньше информации из ИП, т.к. отдельный агент отвечает только за часть определенных для него действий в системе и ему не нужно анализировать весь текущий контекст. Например, в системе SmartRoom агенты получают информацию обо всех выступающих проводимой секции на конференции (до нескольких десятков человек), текущее расписание, различные события (показать слайд, завершить презентацию и пр.). В подобных сценариях разработка на уровне RDF-троек не приведет к заметному повышению производительности, но увеличит объем программного кода в логике агента.

Таблица 4.3

Дополнительное потребление памяти в зависимости от количества локальных объектов.

Класс агентов	Пример устройства	Кол-во объектов	Дополнительный объем памяти (Кб)	
			Индивиды	Свойства
Агенты сенсоров	Одноплатные компьютеры с сенсорами.	до 50	2,344	0,977
Агенты оборудования (встроенными ЭВМ)	Холодильники, телевизоры, медиа-доски и т.д.	до 100	4,688	1,953
Персональные мобильные устройства	Телефон, нетбук, планшет.	до 1000	46,875	19,531
Сервисы, сервис-посредники для организации работы агентов	Настольные, серверные ЭВМ.	от 10000	от 468,75	от 195,312

На стороне агента во время его выполнения основное потребление памяти — это хранение структур данных для индивидов и их свойств. При использовании библиотеки SmartSlog дополнительно требуемый объем памяти также незначителен. Для хранения индивида и свойства требуется 48 и 20 байт, соответственно. Структуры данных для индивидов и свойств заполняются на основе полученных от KP-интерфейса структур для представления RDF-троек. Структуры для RDF-троек не хранятся и после извлечения из них необходимой информации (субъекта, предиката или объекта) они удаляются. Дополнительный объем памяти, требуемый различным классами устройств для хранения свойств и индивидов при взаимодействии в ИП, показан в табл. 4.3. Стоит отметить, что стоимость памяти снижается, а ее объем даже для малопроизводительных устройств растет, также агент может работать только с частью объектов, динамически загружая необходимую информацию из ИП. Маломощные сенсоры редко можно ис-

пользовать как самостоятельного агента, тогда они подключаются к ИП через посредника. В ИЗ сенсоры подключаются к одноплатным компьютерам, которые обладают достаточными ресурсами для выполнения агентов KP на основе библиотек SmartSlog.

4.7. Возможности разработки агентов для различных аппаратно-программных вычислительных устройств

Инструмент SmartSlog поддерживает разработку для разнообразных аппаратно-программных вычислительных платформ за счет генерации библиотек SmartSlog на различных языках программирования. При этом ANSI C версия библиотеки SmartSlog имеет малое число зависимостей от сторонних библиотек и поддерживает возможность отключения определенного функционала (напр., асинхронных подписок) для уменьшения объема программного кода библиотеки SmartSlog.

Экспериментально проверено использование библиотек SmartSlog для платформ, представленных в табл. 4.4 (C KPI и C# KPI — это низкоуровневые KP–интерфейсы). Для C# версии реализованы адаптеры для существующего интерфейса C KPI. Под платформу Android разработчик реализует взаимодействие между Java и ANSI C кодом. Поддерживается работа на одноплатном компьютере Raspberry Pi с ОС Wheezy, основанной на ОС Linux.

Библиотеки SmartSlog поддерживают подключение низкоуровневых KP-интерфейсов через программные адаптеры. Такая поддержка позволяет: 1) подключать различные KP-интерфейсы без модификации библиотеки SmartSlog или KP-интерфейса, 2) скрыть от разработчика детали се-

Таблица 4.4
Поддерживаемые инструментарием SmartSlog платформы

Интерфейс	Windows	Windows Phone	На основе Linux	Mac OS	Android
ANSI C (C KPI)	+	-	+	+	+ (взаимодействие нативного и Java кода)
C# (C KPI)	+(C KPI адаптер)	+(C KPI адаптер)	На базе Mono		–
C# (C# KPI)	+	+	На базе Mono		–

тевого взаимодействия агента с ИП (напр., протокол передачи данных), 3) разрабатывать агентов КР для других платформ, сходных по функциональным возможностям с платформой Smart-M3.

Разработаны четыре адаптера для КР-интерфейсов. Два из них разработаны автором (C_KPI для Windows и Windows Phone) и являются сейчас основными. Два другие адаптера поддерживают ранее существующие интерфейсы KPI_Low и C# KPI.

4.8. Выводы

На основе платформы Smart-M3 разворачиваются ИП с косвенным взаимодействием агентов и поддержкой технологий Семантического веб, что позволяет использовать ее для апробации предложенных метода и специализированных моделей взаимодействия при разработки агентов. Платформа Smart-M3 позволяет интегрировать технологии, используемые в многоагентных системах (агентских платформах) и Семантическом веб. В результате, существует поддержка подключения различных устройств, представленных агентами или умными объектами к многоагентной системе через ИП, что способствует реализации концепции IoT. Реализованный инструмент SmartSlog поддерживает предложенный метод программиро-

вания косвенного взаимодействия и позволяет генерировать онтологические библиотеки SmartSlog. В результате, разработчик генерирует необходимые ему библиотеки SmartSlog и программирует взаимодействие агентов в терминах проблемной области. Проведенные эксперименты и тесты показывают незначительное снижение производительности при использовании библиотек SmartSlog и повышение эффективности разработки агента за счет использования реализованных механизмов программирования для логики агента. Инструмент SmartSlog применяется как основное средство разработки агентов для системы интеллектуального зала SmartRoom. Библиотеки SmartSlog используются на различных устройствах (стационарных компьютерах, ноутбуках, мобильных телефонах), что подтверждает возможность использования библиотек SmartSlog для разработки агентов на различных аппаратно-программных вычислительных устройствах.

Заключение

Полученные в диссертационном исследовании результаты представляют собой решение актуальной задачи повышения эффективности разработки программных агентов интеллектуальных пространств. Внедрение результатов вносит вклад в развитие методов разработки программного обеспечения для интеллектуальных пространств. В ходе исследования получены следующие основные результаты:

1) Разработан метод программирования косвенного взаимодействия на основе специализированных моделей взаимодействия с использованием технологий Семантического веб. Метод позволяет повысить эффективность разработки программных агентов в условиях разнообразия аппаратно-программных платформ в IoT-средах.

2) Разработана модель взаимодействия агентов на основе многоэлементной сессии для поддержки параллельных сеансов сетевого доступа и интеграции информационных объектов из нескольких ИП. Модель позволяет организовать группы сеансов в сессии и управлять их сетевым взаимодействием с ИП.

3) Разработана модель взаимодействия агентов на основе операции подписки для автоматизации отслеживания агентом происходящих в ИП изменений на уровне объектов онтологической модели. Модель позволяет автоматически синхронизировать подписанные объекты с их изменениями в ИП.

4) Разработана модель взаимодействия агентов на основе локальной обработки группы объектов для программирования в логике агента операций над объектами онтологической модели как реакции на отслеживаемое событие. Модель позволяет агенту вносить множественные изменения в ИП в виде одной транзакцию, что уменьшает количество сетевых обращений агента к ИП и поддерживает семантическую целостность ИП.

5) Реализованы механизмы программирования косвенного взаимодействия и апробированы в программном инструменте SmartSlog платформы Smart-M3. Механизмы позволяют уменьшить объем создаваемого вручную разработчиком программного кода агента, поддерживают различные языки программирования, обеспечивают возможность программирования агентов для маломощных устройств.

Литература

1. *Александров, В. В.* Методы построения информационно-логистических систем / В. В. Александров, Н. А. Андреева, С. В. Кулешов. — СПб.: Политехнический университет, 2006. — 95 с.

2. *Гаврилов, А. В.* Искусственный домовой. / А. В. Гаврилов // *Искусственный интеллект и принятие решений.* — 2012. — № 2. — С. 77–89.

3. *Гамма, Э.* Приемы объектно-ориентированного проектирования. Паттерны проектирования. / Э. Гамма, Р. Хельм, Р. Джонсон. — СПб.: Питер, 2011. — 368 с.

4. *Городецкий, В. И.* Агенты и извлечение знаний из данных в интеллектуальном пространстве. / В. И. Городецкий // *Сборник трудов первой международной конференции "Автоматизация управления и интеллектуальные системы и среды".* — 2010. — С. 24–36.

5. *Городецкий, В. И.* Методология разработки прикладных многоагентных систем в среде MASDK. / В. И. Городецкий, О. В. Карсаев, В. Г. Конюший и др. // *Труды СПИИРАН.* — 2006. — Т. 3. — С. 11–32.

6. *Калиниченко, Л.* Методология организации решения задач над множественными распределенными неоднородными источниками информации / Л. Калиниченко // *Сб. тр. межд. конф. "Современные инфор-*

мационные технологии и ИТ-образование". М.: МГУ. — 2005. — С. 20–37.

7. *Корзун, Д. Ж.* Автоматизированная модельно-ориентированная разработка программных агентов для интеллектуальных пространств на платформе Smart-M3 / Д. Ж. Корзун, А. А. Ломов, П. И. Ванаг // *Теоретический и прикладной научно-технический журнал "Программная инженерия".* — 2012. — № 5. — С. 6–14.

8. *Курдюков, А. А.* Интеллектуальные агенты и их применения в инженерном проектировании / А. А. Курдюков // *Доклады конференции CAD/CAM/PDM.* — 2001. — Режим доступа: `http://lab18.ipu.ru/projects/conference2004.htm`, свободный. — Загл. с экрана. Дата доступа: 2011-05-20.

9. *Ломов, А. А.* Операция подписки для приложений в интеллектуальных пространствах платформы Smart-M3 / А. А. Ломов, Д. Ж. Корзун // *Труды СПИИРАН.* — 2012. — Т. 23. — С. 439–458.

10. *Разумовский, А.* Использование онтологий в разработке программного обеспечения / А. Разумовский, М. Пантелеев // Сб. тр. конф. "Инженерия знаний и технологии семантического веба - 2011". — СПб.: НИУ ИТМО, 2011. — С. 88–95.

11. *Разумовский, А. Г.* Валидация объектно-ориентированных программ с использованием онтологии / А. Г. Разумовский, М. Г. Пантелеев // *Программная инженерия.* — 2012. — № 7. — С. 7–13.

12. *Смирнов, А. В.* Онтолого-ориентированный многоагентный подход к

построению систем интеграции знаний из распределённых источников / А. В. Смирнов, Т. В. Левашова, М. П. Пашкин и др. // *Информационные технологии и вычислительные системы.* — 2002. — № 1. — С. 62–68.

13. *Таненбаум, Э.* Компьютерные сети. / Э. Таненбаум, Д. Уэзеролл. — СПб.: Питер, 2012. — 955 с.

14. *Тарасов, В.* От многоагентных систем к интеллектуальным организациям: философия, психология, информатика. / В. Тарасов. — М.: Едиториал УРСС, 2002. — 352 с.

15. *Тарасов, В. Б.* Агенты, многоагентные системы, виртуальные сообщества: стратегическое направление в информатике и искусственном интеллекте / В. Б. Тарасов // *Новости искусственного интеллекта.* — 1998. — № 2. — Режим доступа: `http://www.raai.org/library/ainews/1998/2/TARASOV.ZIP`, свободный. — Загл. с экрана. Дата доступа: 2013-03-14.

16. *Хорошевский, В. Ф.* Пространства знаний в сети интернет и semantic web / В. Ф. Хорошевский // *Искусственный интеллект и принятие решений.* — 2008. — № 1. — С. 80–97.

17. *Юсупов, Р. М.* От умных приборов к интеллектуальному пространству / Р. М. Юсупов, А. Л. Ронжин // *Вестник РАН: научный и общественно-политический журнал.* — 2010. — Т. 80, № 1. — С. 45–51.

18. AgentOWL: Semantic Knowledge Model and Agent Architecture. /

L. Hluchy, M. Laclavik, Z. Balogh, M. Babik // *Computers and Artificial Intelligence.* — 2006. — T. 25, № 5. — C. 421–439.

19. *Allan, R. J.* DL Technical Reports: Survey of Agent Based Modelling and Simulation Tools / R. J. Allan. — 2010. — October. — 42 c.

20. *Allemang, D.* Semantic Web for the Working Ontologist: Effective Modeling in RDFS and OWL / D. Allemang, J. Hendler. Morgan Kaufmann. — Elsevier Science, 2011. — 384 c.

21. An RDF-Based Publish/Subscribe System / D. Shi, J. Yin, Y. Li et al. // *Semantics, Knowledge and Grid, International Conference on.* — 2007. — Vol. 0. — Pp. 342–345.

22. Architectural implications for context-adaptive smart spaces / P. Nixon, S. Dobson, S. Terzis, F. Wang // Proceedings of the International Workshop on Networked Applicances. — IEEE Press, 2003. — C. 156–161.

23. *Balandin, S.* Key properties in the development of smart spaces / S. Balandin, H. Waris // *Proc. 5th Int'l Conf. Universal Access in Human-Computer Interaction (UAHCI'09). Part II: Intelligent and Ubiquitous Interaction Environments. Springer-Verlag.* — 2009. — C. 3–12.

24. *Bellifemine, F. L.* Developing Multi-Agent Systems with JADE / F. L. Bellifemine, G. Caire, D. Greenwood. — Wiley, 2007. — 300 c.

25. *Berners-Lee, T.* The semantic web / T. Berners-Lee, J. Hendler, O. Lassila // *Scientific American.* — 2001. — T. 284, № 5. — C. 34–43.

26. *Boldyrev, S.* A Mechanism for Managing and Distributing Information

and Queries in a Smart Space Environment / S. Boldyrev, I. Oliver, J. Honkola // *Proc. of MDMD 2009.* — 2009. — May. — 10 c.

27. *Bordini, R. H.* Programming Multi-Agent Systems in AgentSpeak Using Jason (Wiley Series in Agent Technology) / R. H. Bordini, J. F. Hübner, M. Wooldridge. — John Wiley & Sons, 2007.

28. *Brickley, D.* RDF Vocabulary Description Language 1.0: RDF Schema: Tech. rep. / D. Brickley, R. V. Guha: 2004. — 2. — Режим доступа: `http://www.w3.org/TR/2004/REC-rdf-schema-20040210/`, свободный. — Загл. с экрана. Дата доступа: 2011-01-30.

29. *Caire, G.* JADE Tutorial: Application-Defined Content Languages and Ontologies / G. Caire. — 2002. — C. 26.

30. *Cook, D. J.* How smart are our environments? an updated look at the state of the art / D. J. Cook, S. K. Das // *Pervasive and Mobile Computing.* — 2007. — T. 3, № 2. — C. 53–73.

31. *Cook, D. J.* Designing smart homes / D. J. Cook, M. Youngblood, S. K. Das / Ed. by J. C. Augusto, C. D. Nugent. — Berlin, Heidelberg: Springer-Verlag, 2006. — Pp. 165–182.

32. *Deniélou, P.-M.* Multiparty session types meet communicating automata / P.-M. Deniélou, N. Yoshida // Proceedings of the 21st European conference on Programming Languages and Systems. — ESOP'12. — Berlin, Heidelberg: Springer-Verlag, 2012. — C. 194–213.

33. The description logic handbook: theory, implementation, and applications /

Под ред. F. Baader, D. Calvanese, D. L. McGuinness и др. — New York, NY, USA: Cambridge University Press, 2003. — 574 с.

34. Developing multi agent systems on semantic web environment using seagent platform. / O. Dikenelli, R. C. Erdur, G. Kardas et al. // ESAW / Ed. by O. Dikenelli, M. P. Gleizes, A. Ricci. — Vol. 3963 of *Lecture Notes in Computer Science*. — Springer, 2005. — Pp. 1–13.

35. Development of a future intelligent sweet home for the disabled / L. Ju-Jang, S. Kap-Ho, O. Changmok, B. Zenn // *Artificial Life and Robotics*. — 2007. — № 11. — С. 8–12.

36. *DuCharme, B.* Learning SPARQL / B. DuCharme; Под ред. S. St. Laurent, J. Perez. — 2011. — 258 с.

37. *Euzenat, J.* Ontology matching / J. Euzenat, P. Shvaiko // *Springer-Verlag*. — 2007. — С. 341.

38. *Fensel, D.* Triple-space computing: Semantic web services based on persistent publication of information / D. Fensel // *Proc. IFIP Int'l Conf. Intelligence in Communication Systems (INTELLCOMM 2004)*. — 2004. — № 1. — С. 43–53.

39. Foundations of semantic web databases / C. Gutierrez, C. A. Hurtado, A. O. Mendelzon, J. Perez // *J. Comput. Syst. Sci.* — 2011. — Т. 77, № 3. — С. 520–541.

40. Foundations of semantic web databases / Gutierrez C., Hurtado C. A., Mendelzon A. O., Perez J. // *Journal of Computer and System Sciences*. — 2011. — № 3. — С. 520–541.

41. *Galov, I.* The smartroom infrastructure: Service runtime reliability / I. Galov, D. Korzun // *Proc. 14th Conf. Open Innovations Framework Program FRUCT.* — 2013. — November. — С. 188–189.

42. *Garlik, S. H.* Sparql 1.1 query language. world wide web consortium. / S. H. Garlik, A. Seaborne, E. Prud'hommeaux. — 2013. — Режим доступа: `http://www.w3.org/TR/sparql11-query/`, свободный. — Загл. с экрана. Дата доступа: 2013-06-30.

43. *Gelernter, D.* Generative communication in Linda / D. Gelernter // *ACM Transactions on Programming Languages and Systems.* — 1985. — № 1. — С. 80–112.

44. Generating Modest High-Level Ontology Libraries for Smart-M3 / D. G. Korzun, A. A. Lomov, P. I. Vanag и др. // *Proc. 4th Int'l Conf. on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM 2010).* — 2010. — October. — С. 103–109.

45. *Gomez-Goiri, A.* A Triple Space-Based Semantic Distributed Middleware for Internet of Things / A. Gomez-Goiri, D. Lopez-de Ipina // *Proc. 10th Int'l Conf. Current trends in web engineering (ICWE 2010). LNCS 6385. Springer-Verlag.* — 2010. — С. 447–458.

46. *Guarino N.* Formal ontology, conceptual analysis and knowledge representation / Guarino N. // *International Journal of Human-Computer Studies. Special issue: the role of formal ontology in the information technology.* — 1995. — Т. 43, № 5-6. — С. 625–640.

47. *Helsinger, A.* Cougaar: a scalable, distributed multi-agent architecture. /

A. Helsinger, M. Thome, T. Wright // SMC (2). — IEEE, 2004. — Pp. 1910–1917.

48. *Hideki, H.* Present state and future of intelligent space-discussion on the implementation of rt in our environment / H. Hideki // *Artificial Life and Robotics.* — 2007. — № 11. — С. 1–7.

49. Integration of Smart-M3 applications: Blogging in smart conference / D. Korzun, I. Galov, A. Kashevnik и др. // *Proc. 11th Int'l Conf. Next generation wired/wireless networking and 4th Int'l Conf. Smart spaces (NEW2AN'11/ruSMART'11). LNCS 6869. Springer.* — 2011. — С. 51–62.

50. The internet of things / R. Van Kranenburg, E. Anzelmo, A. Bassi и др. // *Draft paper Prepared for the 1st Berlin Symposium on Internet and Society.* — 2011. — October.

51. *Khushraj, D.* sTuples: Semantic Tuple Spaces / D. Khushraj, O. Lassila, T. W. Finin. — IEEE Computer Society, 2004. — С. 268–277.

52. *Kiljander, J.* Knowledge Sharing Protocol for Smart / J. Kiljander, F. Morandi, J.-P. Soininen // *International Journal of Advanced Computer Science and Applications (IJACSA).* — 2010. — Т. 3, № 9. — С. 14.

53. *Knublauch, H.* Ontology-driven software development in the context of the semantic web: An example scenario with protege/owl / H. Knublauch // 1st International Workshop on the Model-Driven Semantic Web (MDSW2004) / Под ред. D. S. Frankel, E. F. Kendall, D. L. McGuinness. — 2004. — С. 56–84.

54. *Korzun, D.* Deployment of smart spaces in internet of things: Overview of the design challenges / D. Korzun, S. Balandin, A. Gurtov. — 2013. — Vol. 8121. — Pp. 48–59.

55. *Korzun, D.* Development of smart room services on top of smart-m3 / D. Korzun, I. Galov, S. Balandin // *Proc. 14th Conf. Open Innovations Association FRUCT.* — 2013. — November. — C. 83–94.

56. *Korzun, D. G.* SmartSlog 0.3x: New Ontology Library API and Optimization / D. G. Korzun, A. A. Lomov, P. I. Vanag // *Proc. 8th Conf. Open Innovations Framework Program FRUCT.* — 2010. — November. — C. 85–91.

57. *Kusznir, J.* Designing lightweight software architectures for smart environments / J. Kusznir, D. J. Cook // Proceedings of the Sixth International Conference on Intelligent Environments (IE 2010), Kuala Lumpur, Malaysia. — 2010. — Pp. 220–224.

58. *Lassila, O.* Generating rewrite rules by browsing rdf data. / O. Lassila // RuleML / Ed. by T. Eiter, E. Franconi, R. Hodgson, S. Stephens. — IEEE Computer Society, 2006. — Pp. 51–57.

59. *Lomov, A. A.* Smartslog Session Scheme for Smart-M3 applications / A. A. Lomov // *Proc. 12th Conf. Open Innovations Association FRUCT.* — 2012. — November. — C. 66–71.

60. *Lomov, A. A.* Ontology-based KP Development for Smart-M3 applications / A. A. Lomov // *Proc. 13th Conf. Open Innovations Association FRUCT.* — 2013. — April. — C. 94–100.

61. *Lomov, A. A.* Subscription Operation in Smart-M3 / A. A. Lomov, D. G. Korzun // *Proc. 10th Conf. Open Innovations Association FRUCT and 2nd Finnish-Russian Mobile Linux Summit.* — 2011. — Nov. — С. 83–94.

62. *Lomov, A. A.* Multilingual Ontology Library Generator for Smart-M3 Application Development / A. A. Lomov, P. I. Vanag, D. G. Korzun // *Proc. 9th Conf. Open Innovations Framework Program FRUCT.* — 2011. — November. — С. 6–14.

63. *Lynch, D.* A Proactive approach to Semantically Oriented Service Discovery / D. Lynch, et al. — 2006. — 93 с.

64. The many faces of publish/subscribe / P. Eugster, P. Felber, A. Kenmarrec, G. R. // *ACM Computing Surveys (CSUR).* — 2003. — June. — Т. 35. — С. 114–131.

65. Multilingual Ontology Library Generator for Smart-M3 Information Sharing Platform / D. G. Korzun, A. A. Lomov, P. I. Vanag и др. // *International Journal On Advances in Intelligent Systems.* — 2011. — November. — Т. 4, № 3–4. — С. 68–81.

66. Multilingual ontology library generator for Smart-M3 Information Sharing Platform / D. G. Korzun, A. A. Lomov, P. I. Vanag и др. // *International Journal On Advances in Intelligent Systems.* — 2012. — Т. 4, № 3. — С. 68–81.

67. *Murth, M.* Knowledge-based interaction patterns for semantic spaces /

M. Murth, eva Kuhn // CISIS / Под ред. L. Barolli, F. Xhafa, S. Vitabile, H.-H. Hsu. — IEEE Computer Society, 2010. — С. 1036–1043.

68. *Nikolai, C.* Tools of the trade: A survey of various agent based modeling platforms / C. Nikolai, G. Madey // *Journal of Artificial Societies and Social Simulation.* — 2009. — Т. 12, № 2. — С. 2.

69. *Noy, N. F.* Ontology development 101: A guide to creating your first ontology. [Электронный ресурс] / N. F. Noy, D. L. McGuinness. — Электрон. дан. — Режим доступа: `http://www.ksl.stanford.edu/people/dlm/papers/ontology101/ontology101-noy-mcguinness.html`, свободный. — Загл. с экрана. Дата доступа: 2011-06-30.

70. *Oliver, I.* Personal Semantic Web Through A Space Based Computing Environment / I. Oliver, J. Honkola. — 2008. — August. — 14 с.

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

72. Ontologies for enterprise knowledge management / A. Maedche, B. Motik, L. Stojanovic и др. // *IEEE Intelligent Systems.* — 2003. — Mart. — Т. 18. — С. 26–33.

73. Ontology Driven Smart Space Application Development / M. M. Saleemi, N. Diaz Rodriguez, E. Suenson et al. // Semantic Interoperability: Issues, Solutions, and Challenges, 101-125. / Ed. by S. F. Pileggi, C. Fernandez-Llatas. — River Publishers, 2012. — Pp. 1–25.

74. Ontology-Driven Software Development / Ed. by J. Z. Pan, S. Staab, U. Aßmann et al. — Berlin: Springer, 2013.

75. Overview of Smart-M3 principles for application development / D. G. Korzun, S. I. Balandin, V. Luukkala и др. // Proc. Congress on Information Systems and Technologies (IS&IT'11), Conf. Artificial Intelligence and Systems (AIS'11). — Т. 4. — Moscow: Physmathlit, 2011. — September. — С. 64–71.

76. Owl 2: The next step for owl / B. C. Grau, I. Horrocks, B. Motik et al. // *Web Semant.* — 2008. — November. — Vol. 6, no. 4. — Pp. 309–322.

77. *Bechhofer, S.* OWL Web Ontology Language Reference / S. Bechhofer, F. van Harmelen, J. Hendler et al. — Электрон. дан. — 2004. — Режим доступа: `http://www.w3.org/TR/owl-ref/`, свободный. — Загл. с экрана. Дата доступа: 2012-03-20.

78. *Poole, J. D.* Model-driven architecture: Vision, standards and emerging technologies / J. D. Poole // *Workshop on Metamodeling and Adaptive Object Models.* — 2001. — Т. 4, № 4. — С. 337–361.

79. *Poslad, S.* Specifying protocols for multi-agent systems interaction. / S. Poslad // *TAAS.* — 2007. — Vol. 2, no. 4.

80. *Poslad, S.* Ubiquitous Computing: Smart Devices, Environments and Interactions / S. Poslad. — Wiley, 2009. — 502 с.

81. *Powers, S.* Practical RDF / S. Powers. — Beijing, Cambridge: O'Reilly, 2003. — С. 352.

82. RedSib: a Smart-M3 Semantic Information Broker Implementation / F. Morandi, L. Roffia, A. D'Elia и др. // *Proc. 12th Conf. Open Innovations Association FRUCT.* — 2012. — November. — С. 86–98.

83. *Ronzhin, A. L.* A comparative analysis of smart space prototypes functionality / A. L. Ronzhin, A. A. Karpov // *Trudy SPIIRAN – SPIIRAS Proceedings.* — 2013. — Т. 24. — С. 277–290.

84. *Schmidt, D. C.* Guest editor's introduction: Model-driven engineering / D. C. Schmidt // *Computer.* — 2006. — Vol. 39, no. 2. — Pp. 25–31.

85. *Segaran, T.* Programming the Semantic Web / T. Segaran, J. Taylor, C. Evans. — Cambridge, MA: O'Reilly, 2009. — С. 302.

86. *Singh, R.* State of the Art Smart Spaces: Application Models and Software Infrastructure / R. Singh, P. Bhargava, S. Kain // *Ubiquity.* — 2006. — September. — Т. 2006. — С. 7:2–7:9.

87. Smart-M3 Information Sharing Platform / J. Honkola, H. Laine, R. Brown, O. Tyrkkö // Proc. IEEE Symp. Computers and Communications. — ISCC '10. — IEEE Computer Society, 2010. — June. — С. 1041–1046.

88. Smart-M3 is a functional platform that provides a cross domain search extent for triple based information. [Электронный ресурс]. — Электрон. дан. — Режим доступа: `http://smart-m3.sourceforge.net/`, свободный. — Загл. с экрана.

89. SPARQL query language for RDF. — Электрон. дан. — Режим доступа: `http://www.w3.org/TR/rdf-sparql-query/`, свободный. — Загл. с экрана. Дата доступа: 2013-10-01.

90. *Staab, S.* Handbook on Ontologies / S. Staab, R. Studer. — 2nd изд. — Springer Publishing Company, Incorporated, 2009. — 811 с.

91. *Terekhov, A. N.* Basic Technology for Creating Mobile Distributed Systems / A. N. Terekhov, A. M. Kudinov, S. I. Makarov // *Proceedings of the International Workshop School of Management New Models of Business. Managerial Aspects and Enabling Technology.* — 2002. — June. — С. 187–195.

92. *The OSGi Alliance.* OSGi Alliance Specifications / The OSGi Alliance. — 2012. — Режим доступа: `http://www.osgi.org/Specifications/`, свободный. — Загл. с экрана. Дата доступа: 2012-02-30.

93. Tuplespace-based computing for the semantic web: A survey of the state-of-the-art / L. Nixon, Simperl E., Krummenacher R., Martinrecuerda F. // *The Knowledge Engineering Review.* — 2008. — № 2. — С. 181–212.

94. Using Semantic web technology in Multi-Agent systems: a case study in the TAGA Trading agent environment / Y. Zou, T. Finin, L. Ding et al. // Proceeding of the 5th International Conference on Electronic Commerce. — 2003. — September. — Pp. 95–101.

95. *Vanag, P. I.* SmartSlog: An ANSI C Ontology Library Generator for the Smart-M3 Platform / P. I. Vanag, A. A. Lomov, D. G. Korzun // *The Annual Int'l Workshop on Advances in Methods of Information and Communication Technology (AMICT'2010).* — 2010. — May. — С. 45–50.

96. *Watson, A. H.* Structured testing: A testing methodology using the cyclomatic complexity metric: Tech. Rep. NIST Special Publication 500-235 /

A. H. Watson, T. J. Mccabe: National Institute of Standards and Technology (NIST), 1996.

97. *Weiser, M.* Hot topics—ubiquitous computing / M. Weiser // *Computer.* — 1993. — October. — T. 26, № 10. — C. 71–72.

98. *Yu, L.* A Developer's Guide to the Semantic Web. / L. Yu. — Springer, 2011. — 608 c.

Приложение

A. Сводка доступных KP-интерфейсов для платформы Smart-M3

В таблице сведены известные низкоуровневые и высокоуровневые KP-интерфейсы доступные для свободного доступа и использования при разработке агентов интеллектуальных пространств на платформе Smart-M3.

Таблица A.5
KP-интерфейсы для разработки агентов на платформе Smart-M3.

№	KP-интерфейс	Средства программирования KP	Автор	Публичный репозитарий
Низкоуровневые KP-интерфейсы (операции с RDF-тройками)				
1	Whiteboard, Whiteboard-Qt	C/Glib, C/Dbus, C++/Qt	Исследовательский центр Nokia (Хельсинки, Финляндия)	`http://sourceforge.net/projects/smart-m3/`
2	M3-Python KPI	Python	Исследовательский центр Nokia (Хельсинки, Финляндия)	`http://sourceforge.net/projects/smart-m3/`
3	KPI_Low	ANSI C (ограниченный набор для программирования малопроизводительных устройств)	Научно-технический Центр Финляндии VTT (Оулу, Финляндия)	`http://sourceforge.net/projects/kpilow/`
4	C_KPI	ANSI C (основан на KPI_Low, поддержка SPARQL)	Петрозаводский государственный университет (Петрозаводск, Россия)	`http://sourceforge.net/projects/smartslog/`
5	Smart-M3 Java KPI library	Java (KP работает через веб-обозреватель)	Болонский университет (Болонья, Италия) и Научно-технический Центр Финляндии VTT (Оулу, Финляндия)	`http://sourceforge.net/projects/smartm3-javakpi/`

№	KP-интерфейс	Средства программирования KP	Автор	Публичный репозитарий
6	Smart-M3 PHP KPI library	PHP (KP работает через веб-обозреватель)	Болонский университет (Болонья, Италия)	http://sourceforge.net/projects/sm3-php-kpi-lib/
7	C# KPI for Smart-M3	.NET, C#	Болонский университет (Болонья, Италия)	http://sourceforge.net/projects/m3-csharp-kpi/
Высокоуровневые KP-интерфейсы (операции в терминах предметной области)				
8	Smart-M3 OWL-DL to C(glib) API Generator	C/Glib, C/Dbus	Университет Åbo Akademi (Турку, Финляндия)	http://sourceforge.net/projects/smart-m3/
9	Smart-M3 Ontology to Python API Generator	Python	Исследовательский центр Nokia (Хельсинки, Финляндия)	http://sourceforge.net/projects/smart-m3/
10	SmartSlog	ANSI C .NET (C#)	Петрозаводский государственный университет (Петрозаводск, Россия)	http://sourceforge.net/projects/smartslog/

Б. Регистрация программы для ЭВМ

Свидетельство о государственной регистрации программы для ЭВМ.

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2014615902

Программный каркас «SmartSlog»

Правообладатель: ***Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования «Петрозаводский государственный университет» (RU)***

Авторы: ***Ломов Александр Андреевич (RU), Ванаг Павел Имантович (RU), Марченков Сергей Александрович (RU), Корзун Дмитрий Жоржевич (RU)***

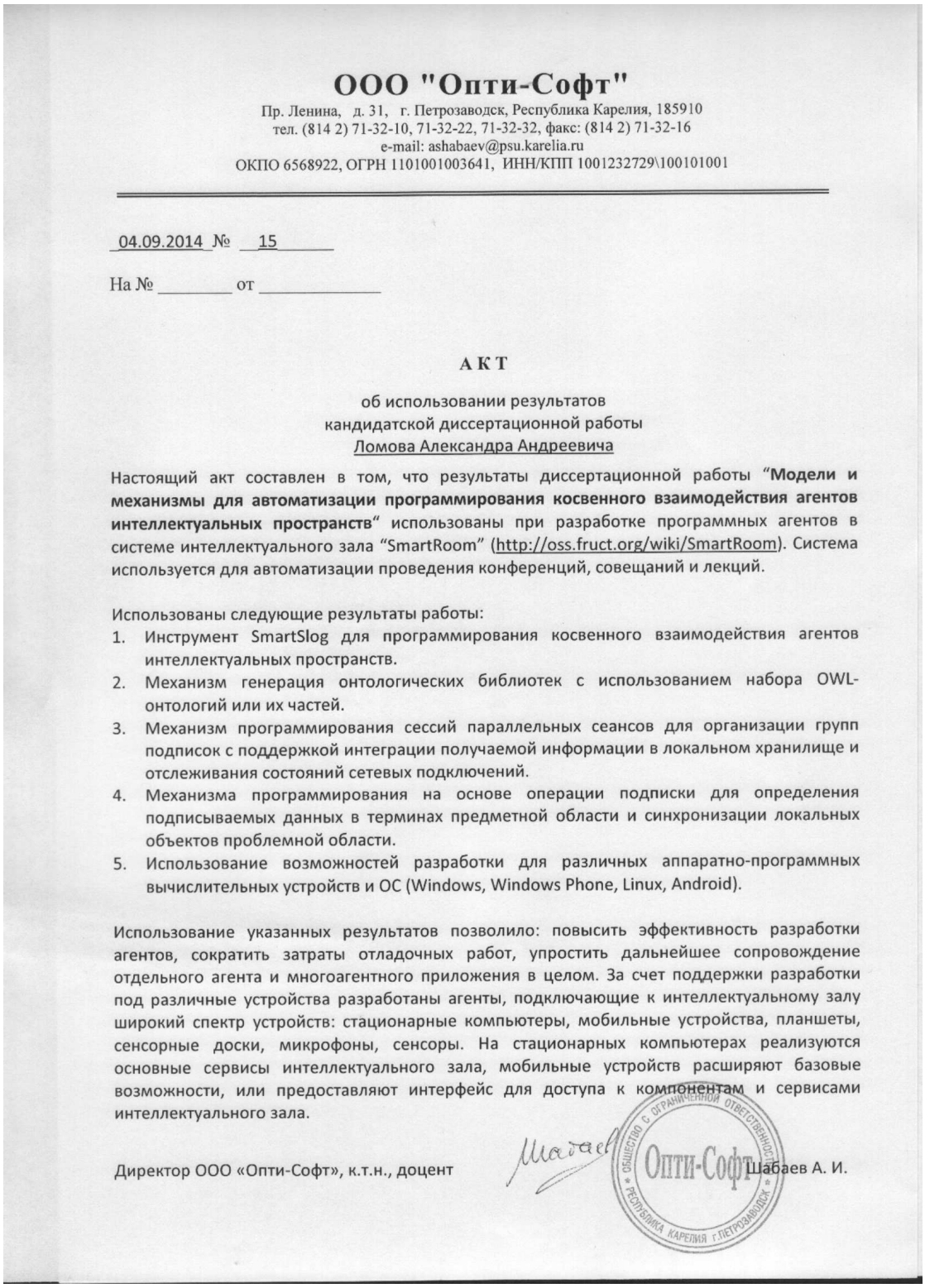
Заявка № **2014613192**
Дата поступления **11 апреля 2014 г.**
Дата государственной регистрации
в Реестре программ для ЭВМ ***05 июня 2014 г.***

Руководитель Федеральной службы по интеллектуальной собственности

Б.П. Симонов

В. Акты внедрения

1) Акт об использовании результатов кандидатской диссертационной работы для разработки агентов в системе интеллектуального зала SmartSroom, используемой ООО “Опти-Софт”.



ООО "Опти-Софт"

Пр. Ленина, д. 31, г. Петрозаводск, Республика Карелия, 185910
тел. (814 2) 71-32-10, 71-32-22, 71-32-32, факс: (814 2) 71-32-16
e-mail: ashabaev@psu.karelia.ru
ОКПО 6568922, ОГРН 1101001003641, ИНН/КПП 1001232729\100101001

04.09.2014 № 15

На № _____ от _____

АКТ

об использовании результатов
кандидатской диссертационной работы
Ломова Александра Андреевича

Настоящий акт составлен в том, что результаты диссертационной работы “**Модели и механизмы для автоматизации программирования косвенного взаимодействия агентов интеллектуальных пространств**“ использованы при разработке программных агентов в системе интеллектуального зала “SmartRoom” (http://oss.fruct.org/wiki/SmartRoom). Система используется для автоматизации проведения конференций, совещаний и лекций.

Использованы следующие результаты работы:

1. Инструмент SmartSlog для программирования косвенного взаимодействия агентов интеллектуальных пространств.
2. Механизм генерация онтологических библиотек с использованием набора OWL-онтологий или их частей.
3. Механизм программирования сессий параллельных сеансов для организации групп подписок с поддержкой интеграции получаемой информации в локальном хранилище и отслеживания состояний сетевых подключений.
4. Механизма программирования на основе операции подписки для определения подписываемых данных в терминах предметной области и синхронизации локальных объектов проблемной области.
5. Использование возможностей разработки для различных аппаратно-программных вычислительных устройств и ОС (Windows, Windows Phone, Linux, Android).

Использование указанных результатов позволило: повысить эффективность разработки агентов, сократить затраты отладочных работ, упростить дальнейшее сопровождение отдельного агента и многоагентного приложения в целом. За счет поддержки разработки под различные устройства разработаны агенты, подключающие к интеллектуальному залу широкий спектр устройств: стационарные компьютеры, мобильные устройства, планшеты, сенсорные доски, микрофоны, сенсоры. На стационарных компьютерах реализуются основные сервисы интеллектуального зала, мобильные устройств расширяют базовые возможности, или предоставляют интерфейс для доступа к компонентам и сервисами интеллектуального зала.

Директор ООО «Опти-Софт», к.т.н., доцент Шабаев А. И.

Общество с ограниченной ответственностью * Опти-Софт * Республика Карелия г. Петрозаводск

2) Акт об использовании результатов кандидатской диссертационной работы для проведения секций конференций Ассоциации открытых инноваций FRUCT.

Open Innovations Association FRUCT

CERTIFICATE OF DEPLOYMENT

for Ph.D. research application of

Lomov Aleksandr Andreevich

By this certificate of deployment we confirm that results of the Ph.D. thesis research on "**Models and mechanisms for automation of programming agents' indirect interaction in smart spaces**" by Lomov A. A. have been used in the software development process of a portable version of the SmartRoom system (http://oss.fruct.org/wiki/SmartRoom). The system was used as an automatic sections' chair at the following International conferences of the FRUCT Association:

- The 12th International Conference of Open Innovations Association FRUCT (5-9 November 2012, Oulu, Finland), Sections "Internet of Things and Smart Spaces" and "Smart Space Technologies";
- The 13th International Conference of Open Innovations Association FRUCT (22-26 April 2013, Petrozavodsk, Russia), Section "Internet of Things and Smart Spaces I";
- The 14th International Conference of Open Innovations Association FRUCT (11-15 November 2013, Helsinki, Finland), at the Demo Section;
- The 15th International Conference of Open Innovations Association FRUCT (21-25 April 2014, Saint-Petersburg, Russia), Section "Smart-M3 Applications Room".

The software implementation of SmartRoom system is essentially based on the following results of the Ph.D. research of Lomov A. A.

1. Description of software agent development in the domain terms based on ontological libraries.
2. Method of ontology library generation based on a set of ontologies.
3. Subscription operation definition for synchronization of information sources of distributed devices in a computing environment.
4. Automatic integration of information at the development of services.
5. Ontological libraries for cross-platform development targeted for widespread mobile platforms.

These results increased the effectiveness of the software development process, reduced the cost of code debugging and simplified the software maintenance.

15.09.2014

/ Sergey I. Balandin /

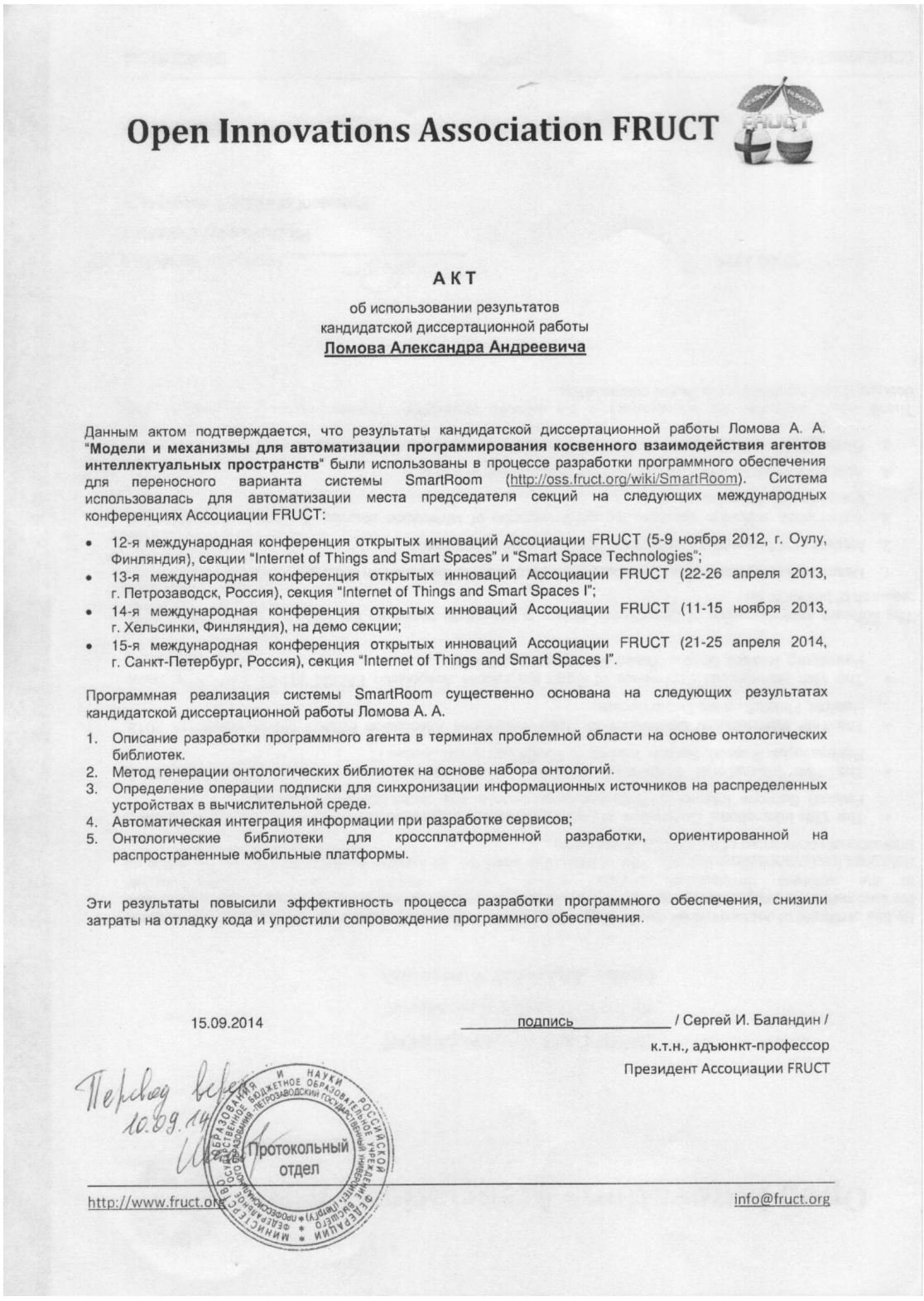
Ph.D., Adjunct Professor

President of FRUCT Association

http://www.fruct.org

info@fruct.org

Перевод акта об использовании результатов кандидатской диссертационной работы для проведения секций конференций Ассоциации открытых инноваций FRUCT.



Open Innovations Association FRUCT

АКТ

об использовании результатов
кандидатской диссертационной работы
Ломова Александра Андреевича

Данным актом подтверждается, что результаты кандидатской диссертационной работы Ломова А. А. **"Модели и механизмы для автоматизации программирования косвенного взаимодействия агентов интеллектуальных пространств"** были использованы в процессе разработки программного обеспечения для переносного варианта системы SmartRoom (http://oss.fruct.org/wiki/SmartRoom). Система использовалась для автоматизации места председателя секций на следующих международных конференциях Ассоциации FRUCT:

- 12-я международная конференция открытых инноваций Ассоциации FRUCT (5-9 ноября 2012, г. Оулу, Финляндия), секции "Internet of Things and Smart Spaces" и "Smart Space Technologies";
- 13-я международная конференция открытых инноваций Ассоциации FRUCT (22-26 апреля 2013, г. Петрозаводск, Россия), секция "Internet of Things and Smart Spaces I";
- 14-я международная конференция открытых инноваций Ассоциации FRUCT (11-15 ноября 2013, г. Хельсинки, Финляндия), на демо секции;
- 15-я международная конференция открытых инноваций Ассоциации FRUCT (21-25 апреля 2014, г. Санкт-Петербург, Россия), секция "Internet of Things and Smart Spaces I".

Программная реализация системы SmartRoom существенно основана на следующих результатах кандидатской диссертационной работы Ломова А. А.

1. Описание разработки программного агента в терминах проблемной области на основе онтологических библиотек.
2. Метод генерации онтологических библиотек на основе набора онтологий.
3. Определение операции подписки для синхронизации информационных источников на распределенных устройствах в вычислительной среде.
4. Автоматическая интеграция информации при разработке сервисов;
5. Онтологические библиотеки для кроссплатформенной разработки, ориентированной на распространенные мобильные платформы.

Эти результаты повысили эффективность процесса разработки программного обеспечения, снизили затраты на отладку кода и упростили сопровождение программного обеспечения.

15.09.2014 подпись / Сергей И. Баландин /

к.т.н., адъюнкт-профессор
Президент Ассоциации FRUCT

Протокольный отдел

http://www.fruct.org info@fruct.org

3) Акт об использовании результатов кандидатской диссертационной работы в рамках курса “Интеллектуальные сетевые пространства (Smart Spaces)”.

Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Петрозаводский государственный университет» (ПетрГУ)

"УТВЕРЖДАЮ"
декан математического факультета,
доцент, к.ф.-м.н.
Варфоломеев Алексей Геннадьевич
“ 09 2014 г.

АКТ

об использовании результатов
кандидатской диссертационной работы
Ломова Александра Андреевича

Комиссия в составе: председателя Семеновой Е. Е., заведующего кафедрой информатики и математического обеспечения Богоявленского Ю. А. и Корзуна Д. Ж. составили настоящий акт о том, что результаты диссертационной работы “**Модели и механизмы для автоматизации программирования косвенного взаимодействия агентов интеллектуальных пространств**“ использованы при проведении курса “Интеллектуальные сетевые пространства (Smart Spaces)”.

Использованы следующие результаты работы:
1. Моделирование проблемной области при помощи редактора онтологий Protegé. Отображение объектов OWL-онтологий в программный код.
2. Разработка агентов на уровне RDF-троек при помощи ИП-интерфейса C KPI для платформы Smart-M3.
3. Разработка агентов на основе онтологических библиотек при помощи инструмента SmartSlog для платформы Smart-M3.
4. Сравнение подходов к разработке агентов на уровне RDF-троек и в терминах проблемной области.

Использование указанных результатов дает студентам базовые знания для разработки программных агентов в интеллектуальных пространствах с косвенным взаимодействием. Дается обзор платформы Smart-M3 для развертывания интеллектуальных пространств. Демонстрируются особенности онтолого-ориентированной разработки и программирование логики агента на уровне RDF-троек и в терминах проблемной области с использованием структур данных, представляющих объекты, определяемые OWL-онтологиями. Практически показано сокращение и упрощение программного кода логики агента при использовании онтологических библиотек SmartSlog в сравнении с ИП-интерфейсом C KPI.

Председатель комиссии
Семенова Е. Е.

Члены комиссии:
Богоявленский Ю.А.
Корзун Д.Ж.