УДК 004.4, 004.8

И.В. Галов, Д.Ж. Корзун

МОДЕЛЬ УВЕДОМЛЕНИЙ ДЛЯ РАЗРАБОТКИ ПРОГРАММНЫХ ПРИЛОЖЕНИЙ ИНТЕЛЛЕКТУАЛЬНЫХ ПРОСТРАНСТВ

Галов И.В., Корзун Д.Ж. **Модель уведомлений для разработки программных приложений интеллектуальных пространств.**

Аннотация. Рассматривается модель уведомлений для организации взаимодействия множества программных агентов в интеллектуальном пространстве. Модель предназначена для проблемно-ориентированной разработки программных приложений на платформе Smart-M3, используя такие возможности, как операция подписки и онтологическое представление информации. Набор требуемых вариантов взаимодействия описывается на основе онтологии уведомлений, расширяющей исходную онтологию предметной области приложения. Реализация взаимодействия сводится к выполнению каждым агентом подписки на нужные варианты взаимодействия из онтологии уведомлений. Применимость модели показана на примере программной системы SmartScribo, реализующей интеллектуальное пространство для мультиблоггинга. Выполненный экспериментальный анализ показывает приемлемую для таких приложений производительность.

Ключевые слова: интеллектуальные пространства, Smart-M3, операция подписки, взаимодействие агентов, онтологии, RDF.

Galov I.V., Korzun D.G. **Notification model for smart space applications development.**

Abstract. We consider a notification model for constructing interactions of multiple software agents in a smart space. The model supports Smart-M3 application development and exploits such features as subscription operation and ontological information representation. The set of required variants for interaction is described using a notification ontology, which extends the application domain ontology. Programming the interaction is reduced to subscription that each agent implements based on the notification ontology. The applicability of model is demonstrated on the case study of SmartScribo system for smart space multi-blogging. Our experiments show reasonable performance for this class of applications.

Keywords: smart spaces, Smart-M3, subscription operation, mutli-agent interaction, ontology, RDF.

1. Введение. Интеллектуальное пространство (ИП) формирует сервисно-ориентированную информационную систему (среду), развернутое в которой программное приложение способно выполнять рассуждения для определения потребностей своих пользователей, адаптации к текущей ситуации, построения и доставки пользователю нужных услуг [1-6]. Платформа Smart-M3 [7] является исследовательской программной реализацией с открытым кодом для создания ИП в разнообразных вычислительных средах, включая локализованные среды Интернета физических устройств (Internet of Things, далее — IoT) [6]. Аббревиатура M3 акцентирует свойства multi-device (множество устройств), multi-vendor (множество производителей аппаратуры)

и multi-domain (множество предметных областей), характерные для возникающих сейчас IoT-сред.

Платформа Smart-M3 определяет построение ИП на основе набора взаимодействующих программных агентов — процессоров знаний (knowledge processor — агент *KP*), работающих на устройствах вычислительной среды. Взаимодействие идет через разделение информационного содержимого, представленного с помощью модели RDF (resource description framework) из Семантического веб [8]. Доступ выполняется через семантический информационный брокер (semantic information broker — SIB). Помимо разовых операций и поисковых запросов (чтение, вставка, изменение, удаление) поддерживается операция подписки — постоянный запрос к брокеру SIB для отслеживания изменений в информационном содержимом [9].

Рост числа участвующих в приложении агентов *KP* и вовлечение ими в обработку разнообразных и объемных данных приводят к трудностям при программировании взаимодействия на уровне каждого агента *KP*. В данной статье предлагается модель уведомлений, определяющая проблемно-ориентированный способ для программирования взаимодействия агентов ИП на платформе Smart-M3. Взаимодействие организуется с помощью публикации агентами *KP* специального вида уведомлений, на которые подписываются другие агенты заданного программного приложения. Прикладному разработчику предоставляется шаблонная схема проектирования и программирования взаимодействия, которая, с одной стороны, упрощает разработку, а, с другой стороны, обеспечивает приемлемую производительность реализуемого взаимодействия.

2. Организация взаимодействия агентов в интеллектуальном пространстве. Базовое взаимодействие агентов *KP* на платформе Smart-M3 основано на известной модели классной доски (доски объявлений) [10]. Брокер SIB управляет доступом агентов к общему информационному содержимому I, поддерживая операции чтения/записи. Взаимодействие является косвенным — каждый агент наблюдает текущее информационное содержимое ИП, формируемое всеми участвующими агентами, на основе чего агент автономно определяет свои действия, включая внесение собственных изменений в информационное содержимое ИП.

Косвенное взаимодействие агентов *KP* расширяется с помощью модели публикации/подписки [11], что позволяет использовать механизмы событийно-управляемого программирования [9]. Агент может подписаться на интересующую его часть информационного содержимого и затем регулярно получать уведомления о происходящих в ней

изменениях. Известно, что операция подписки является ресурсоемкой и критичной для обеспечения надежности по сравнению с разовыми операциями доступа к ИП. В силу этого свойства, при программировании агента следует использовать небольшое число подписок.

В условиях платформы Smart-M3, работа [12] рассматривает организацию взаимодействия агентов ИП, соответствующих потоков управления, передачу информации между агентами и координацию их работы в рамках заданного программного приложения. В то же время, остается открытым вопрос разработки проблемно-ориентированного способа для программирования требуемого взаимодействия на уровне каждого агента (его индивидуальной стратегии участия и координации в ИП) с учетом предметной области приложения.

Помимо платформы Smart-M3, известны и другие, ориентированные на построение многоагентных систем и предоставляющие различные механизмы для организации взаимодействия агентов. В работе [13] рассматривается сервис уведомления о событиях Siena. За доставку уведомлений отвечает распределенная сеть серверов (брокеров), реализующая специализированный протокол доставки. Такой подход распространен в современных системах публикации/подписки в силу масштабируемости при росте числа агентов. В то же время, это платформенное решение, которое обеспечивает эффективность доставки уведомлений, но не предоставляет проблемно-ориентированного способа для программирования взаимодействия в терминах уведомлений на уровне отдельного агента. Отметим, что этот требуемый проблемно-ориентированный способ слабо зависит от реализации доставки — с помощью централизованного брокера или сети брокеров.

В работе [14] исследуется организация взаимодействия агентов ИП на основе концепции оверлейных P2P сетей. Каждый агент ИП становится равноправным участником такой сети. Взаимодействие происходит через обмен агентами информацией друг с другом и сводится к задаче маршрутизации в P2P сети агентов. Агенты выступают не только отправителями и получателями, но и посредниками при передаче информации. Такой подход определяет примитивы, которые платформа предоставляет разработчикам для организации базового взаимодействия агентов, но не определяет способа для программирования взаимодействия в каждом отдельном агенте с учетом заданной предметной области приложения.

Известно достаточно много систем публикации/подписки, в которых информационное содержимое представлено с помощью модели RDF. В системе JTangPS [15] операция подписки использует специализированный язык для описания интересующей агента части информа-

ционного содержимого. Платформа Smart-M3 позволяет использовать в операции подписки шаблоны RDF троек и язык SPARQL (Protocol and RDF Query Language). Соответствующие механизмы упрощают программирование взаимодействия агентов, но прикладному разработчику для заданного приложения все равно приходится проектировать частный вариант взаимодействия с последующей его реализацией на уровне каждого агента *KP*.

Таким образом, можно сделать вывод, что существующие платформенные решения для организации взаимодействия агентов ИП на основе операции подписки требуют от прикладного разработчика проектирования собственного варианта для организации взаимодействия агентов в каждом конкретном приложении. В качестве возможного общего решения нами предлагается модель уведомлений. Она определяет проблемно-ориентированный способ для организации взаимодействия между агентами *KP* на платформе Smart-M3. Данная модель может быть использована при разработке программных приложений и на других платформах, в которых поддерживается операция подписки и представление информационного содержимого по модели RDF.

3. Взаимодействие агентов *KP* заданного программного приложения. Элементарным взаимодействием будем называть бинарное отношение на множестве агентов *KP*:

$$\mathrm{KP}_{\mathrm{snd}} \rightarrow \mathrm{KP}_{\mathrm{rcv}}, \qquad (1)$$

где $\mathrm{KP}_{\mathrm{rcv}}$ читает (разовая операция или по подписке) данные из I, опубликованные $\mathrm{KP}_{\mathrm{snd}}$. Отношение (1) позволяет формализовать понятие программного приложения ИП, как набора всех взаимодействующих агентов *KP*. Агенты KP_1 и KP_2 входят в состав одного приложения, если $\mathrm{KP}_1 \rightarrow \mathrm{KP}_2$ или $\mathrm{KP}_2 \rightarrow \mathrm{KP}_1$. Рефлексивно-транзитивное замыкание отношения состава определяет классы эквивалентности, каждый соответствует некоторому приложению. Отношение (1) обладает следующими полезными свойствами для организации взаимодействия:

– один-ко-многим: один агент $\mathrm{KP}_{\mathrm{snd}}$ может взаимодействовать с несколькими агентами $\mathrm{KP}_{\mathrm{rcv}}$ одновременно;

– асинхронность: отправка и получение данных не блокируют работу агентов $\mathrm{KP}_{\mathrm{snd}}$ и $\mathrm{KP}_{\mathrm{rcv}}$;

– анонимность: агентам $\mathrm{KP}_{\mathrm{snd}}$ и $\mathrm{KP}_{\mathrm{rcv}}$ не требуется непосредственно знать друг друга для выполнения взаимодействия.

Представление содержимого I следует модели RDF из Семантического веб [8, 16]. Базовым элементом является тройка вида «субъект–предикат–объект». Множество хранимых троек образует ориенти-

рованный граф, в котором узлами являются субъекты и объекты, а дуги — предикаты. Брокер SIB реализует RDF-хранилище (базу знаний) с поддержкой точки доступа SPARQL для выполнения семантических поисковых запросов.

Каждый агент *KP* работает лишь с некоторой частью *I* (определяется разработчиком). Для выполнения (1) необходимо, чтобы соответствующие части для KP_{snd} и KP_{rcv} пересекались. Для разработчика описание этих частей удобно представить с помощью онтологий [17, 18, 5]. Известно, что онтология позволяет получить концептуальную модель предметной области [19]. В Семантическом веб используется язык веб онтологий (web ontology language, OWL) [20] для описания фактических данных (ресурсов) на основе структуры онтологических классов (концепций), отношений и ограничений в виде индивидов (экземпляры классов), их свойств данных (атрибуты индивида) и семантических связей (объектные свойства) между индивидами.

Такое проблемно-ориентированное OWL-описание фактических данных может быть автоматически преобразовано в машинно-ориентированное RDF-представление [17, 21]. Тем самым, разработчик может программировать операции доступа агента *KP* и обработки им информационного содержимого ИП в терминах OWL-индивидов, их свойств данных и объектных свойств. При переводе в RDF-представление индивид преобразуется в набор троек с одинаковым значением субъекта (подграф RDF). Такой субъект называется идентификатором индивида. Далее в работе будем использовать понятие OWL-индивидов, но детали предлагаемых решений также приводить в терминах RDF-троек, на уровне которых непосредственно оперирует брокер SIB.

Рассмотрим организацию взаимодействия (1) на основе операции подписки. Результатом взаимодействия агентов KP_{snd} и KP_{rcv} является передача от KP_{snd} факта о наступлении события для вызова определенной реакции на стороне KP_{rcv}. Такой вариант может включать и передачу необходимых данных для выполнения действий (напр., для построения сервиса). При реализации агентов KP_{snd} и KP_{rcv} разработчику необходимо выделить в онтологии приложения свойства, отвечающие за каждый используемый вариант взаимодействия (1). На эти свойства подписывается агент KP_{rcv}. При их изменении в *I* агент KP_{snd} получает обновленные значения и соответствующим образом реагирует. В предметной области подходящие свойства не всегда присутствуют явно, и требуется вводить дополнительные свойства в онтологию приложения.

В настоящее время нет общего подхода к выбору и добавлению таких свойств. Кроме того, при проектировании необходимо решать задачу о передаче обновленных значений свойств, которые определяют исходные данные для выполнения действий агентом KP_{rcv}. Рассмотрим пример на рисунке 1. Пусть агент KP_{snd} изменяет часть свойств (публикует) некоторого индивида. В базовой операции подписки (платформа Smart-M3) информирование KP_{rcv} об изменениях происходит отдельно для каждого обновленного свойства. Однако, агенту KP_{rcv} необходимо получить индивида целиком со всеми его обновленными свойствами. Определение момента завершения множественного изменения индивида в ИП является нетривиальной задачей. Дополнительные сложности появляются, когда требуется организовать взаимодействие с подпиской на свойства сразу нескольких индивидов.

Рис. 1. Пример подписки агента KP_{rcv} на несколько онтологических свойств одного или более индивидов

В итоге, требуемое решение задачи организации взаимодействия агентов *KP* на основе операции подписки должно обладать следующими свойствами: 1) инициирование выполнения действий для каждого требуемого в приложении варианта взаимодействия, 2) возможность передачи всех исходных данных для выполнения действий, 3) общий подход к описанию в онтологии приложения свойств для организации взаимодействия, 4) расширяемость при добавлении вариантов взаимодействия в приложение.

4. Модель уведомлений. Предлагаемая далее модель уведомлений является концептуальной. Известно, что концептуальная модель предоставляет систематизированное содержательное описание предметной области или проблемной ситуации. В случае модели уведомлений проблемной ситуацией выступает взаимодействие между агентами. Такая модель определяет: (а) способ описания (представления) дополнительных онтологических классов и свойств для организации взаимодействия между двумя или более агентами *KP* на основе операции подписки и (б) схему использования этого онтологического описания при программировании взаимодействия в агентах *KP*. Как будет

показано далее, за счет онтологического описания становится возможным проблемно-ориентированные проектирование и программирование взаимодействия на уровне каждого отдельного агента приложения.

Модель уведомлений предназначена для организации следующих двух типов взаимодействия:

– запрос: получатель KP_{rcv} осуществляет требуемые действия, используя исходные данные от агента KP_{snd};

– событие: получатель KP_{rcv} реагирует на определенное событие (информацию), о котором информирует отправитель KP_{snd}.

Они определяют две функциональные точки зрения на агента KP_{rcv}: обработчик данных или реагирующий модуль. Первая точка зрения ближе к процедурному программированию, тогда как вторая соответствует событийно-ориентированному программированию.

Определим уведомление x как информационное сообщение для организации взаимодействия, отправляемое агентом KP_{snd} при выполнении определенного условия e и получаемое агентом KP_{rcv}. В частности, уведомление содержит исходные данные — параметры уведомления, если взаимодействие является вариантом запроса. Каждое уведомление соответствует отдельному варианту взаимодействия. Прикладной разработчик при проектировании приложения должен определит список уведомлений, отражающих все требуемые варианты взаимодействия агентов *KP*.

Рассмотрим двунаправленное расширение взаимодействия (1):

$$KP_{snd} \leftrightarrow KP_{rcv}\ , \qquad (2)$$

Во взаимодействии (2) могут участвовать несколько агентов KP_{snd} и несколько агентов KP_{rcv}. Схема использования уведомлений при организации взаимодействия (2) состоит из нижеперечисленных шагов (рисунок 2), программируемых в агентах KP_{snd} и KP_{rcv}. Перед выполнением этих шагов агенты должны подписаться на уведомления.

1. При выполнении определенного условия e агенту KP_{snd} необходимо инициировать выполнение действий некоторым агентом KP_{rcv}. Тогда KP_{snd} формирует уведомление x с n параметрами и публикует его в I.

2. Агент KP_{rcv} обнаруживает факт публикации уведомления по подписке и получает уведомление x из I.

3. Агент KP_{rcv} обрабатывает уведомление x:

1) получает значения параметров уведомления;
2) выполняет действия, требуемые данным вариантом взаимодействия, используя исходные данные из полученного уведомления;
3) удаляет уведомление x (если тип взаимодействия — запрос).

4. Агент KP_{rcv} формирует и публикует в *I* ответное уведомление *y* с результатами выполнения своих действий.

5. Агент KP_{snd} получает ответное уведомление *y* с помощью операции подписки, реагирует и удаляет уведомление *y*.

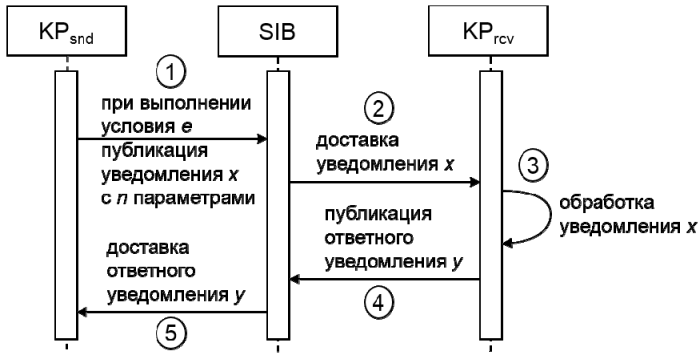

Рис. 2. Схема использования уведомлений агентами KP_{snd} и KP_{rcv}

Шаги 1-3 являются обязательными для взаимодействия (2). Шаги 4-5 реализуют обратную связь и могут быть пропущены, если это не требуется заданным вариантом взаимодействия в приложении.

Таким образом, при разработке каждого агента *KP* разработчик должен учитывать следующие свойства уведомлений: представление (форма уведомления), момент инициации (отправки), тип взаимодействия (запрос или событие), наличие ответного уведомления и момент удаления. Отдельного внимания требует и вопрос производительности из-за использования ресурсоемкой операции подписки.

Представление: простое уведомление (индивид уведомления) содержит только один параметр и состоит из двух RDF-троек:

$$<id_x>, rdf:type, <class_x>$$
$$<id_x>, <name_x>, <value>$$

Здесь $<id_x>$ — идентификатор индивида уведомления *x*, rdf:type — стандартный предикат RDF, описывающий тип индивида, $<class_x>$ — класс из онтологии, описывающий уведомление, $<name_x>$ — свойство из онтологии, описывающее название уведомления (или его параметра), <value> — значение параметра (текстовые данные или идентификатор индивида из онтологии приложения). Параметр содержит данные, которые необходимо передать получателю уведомления. Онтологический класс может содержать несколько свойств, т.е. описывать несколько различных уведомлений.

Составное уведомление состоит из набора RDF-троек для представления передаваемых параметров:

$<id_x>$, rdf:type, $<class_x>$
$<id_x>$, $<name_x>$, $<id_{ind}>$
$<id_{ind}>$, rdf:type, $<class_{ind}>$
$<id_{ind}>$, $<p_1>$, $<value_1>$
…
$<id_{ind}>$, $<p_n>$, $<value_n>$

Здесь $<name_x>$ — название уведомления (вид запроса или события), $<id_{ind}>$ — идентификатор дополнительного индивида, хранящего все параметры уведомления, $<class_{ind}>$ — онтологический класс, описывающий дополнительного индивида, $<p_i>$ — название *i*-го параметра уведомления ($i = 1, \ldots, n$), $<value_i>$ — значение *i*-го параметра.

Такое представление уведомлений задает расширение онтологии приложения (рисунок 3). Каждое уведомление имеет один или более параметров, значение каждого хранится как объект RDF-тройки уведомления. Значением параметра могут быть строковые данные (свойство-значение) или идентификатор индивида из ИП (объектное свойство). Таким образом, уведомление может быть связано с индивидом (его идентификатором), как это схематично показано на рисунке 4. Тем самым, взаимодействие между агентами *KP* осуществляется за счет публикации и изменения данных из онтологии уведомлений, тогда как данные из онтологии приложения остаются неизменны.

Для выполнения взаимодействия между агентами *KP* на основе операции подписки требуется изменение определенных данных в *I*. Без модели уведомлений для передачи информации от KP_{rcv} к KP_{snd} потребовалось бы непосредственное изменение свойств индивидов из онтологии приложения. Однако не во всех предметных областях заданы свойства, изменение которых инициирует взаимодействие между агентами *KP*. Например, для передачи текстового сообщения от KP_{rcv} к KP_{snd} не требуется изменения данных из предметной области приложения, поэтому онтология приложения не содержит в себе классы и свойства для описания таких сообщений. В этом случае разработчику необходимо расширить онтологию приложения. Модель уведомлений предлагает общий способ для описания всех таких свойств, которые отвечают за инициирование взаимодействия. Более того, предлагаемые уведомления описывают и формат обмена информацией между агентами *KP* в случае, если они используют разные онтологии.

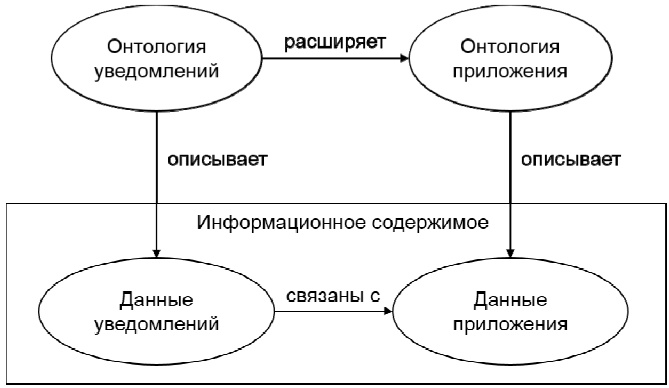

Рис. 3. Связь между уведомлениями и онтологией приложения

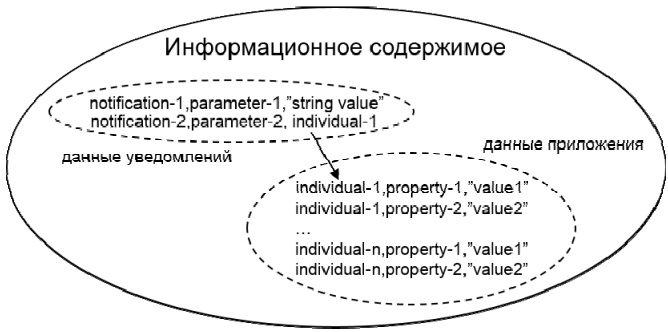

Рис. 4. Пример связи между уведомлением и данными приложения

Инициация взаимодействия: в зависимости от момента отправки на стороне агента KP_{snd} уведомления делятся на реактивные и проактивные (упреждающие). Момент отправки наступает при выполнении условия *e*, которое определяется разработчиком агента *KP*.

Реактивные уведомления соответствуют явной инициации пользователем (команда) на отправку уведомления от агента KP_{snd} для выполнения требуемых действий агентом KP_{rcv}. В этом случае агент KP_{snd}, как правило, ожидает ответного уведомления с результатом выполненных действий.

Проактивное уведомление агент KP_{snd} посылает без явной команды от пользователя и, как правило, не ожидает ответного уведомления. Контекст пользователя формирует неявную зависимость от активности пользователя: агент KP_{snd} в фоновом режиме анализирует контекст (параллельно с основной активностью пользователя) и затем осуществляет отправку уведомления. Проактивные уведомления

обычно используются при событийно-ориентированном программировании взаимодействия между агентами *KP*.

Пример реактивного уведомления: пользователь запускает агента KP_{snd} (клиентская программа) для получения информационного сервиса. Сервисный агент KP_{rcv} отвечает за построение сервиса. По команде пользователя клиентский агент KP_{snd} отправляет реактивное уведомление и ожидает ответа. Пользователь явно вовлечен в процесс инициации взаимодействия.

Пример проактивного уведомления: сервисный агент KP_{rcv} выполняет рекомендательную функцию (напр., анализирует данные, с которыми работает пользователь, и предлагает сходные данные из других источников). На клиентском агенте KP_{snd} пользователь просматривает информацию в рамках своей основной деятельности. Параллельно, агент KP_{snd} автоматически посылает уведомления, описывающие контекст работы пользователя. Сервисный агент KP_{rcv} реагирует и формирует рекомендации, которые уже явно получает пользователь. Отметим, что сервисный агент KP_{rcv} может быть отключен, а приложение продолжит работу (с ограниченным набором сервисов).

Тип взаимодействия: по запросу или по событию. Уведомление-запрос используется для выполнения определенного действия другим агентом *KP* (один получатель). В предыдущих рассмотренных примерах использовалось уведомление-запрос: клиентский агент KP_{snd} запрашивал (реактивно или проактивно) сервисный агент KP_{rcv}. Уведомление-событие позволяет информировать других агентов *KP* (получателей может быть несколько) о наступлении события. Например, уведомление посылается, когда пользователь читает, а факт чтения может быть использован для приостановки формирования рекомендаций сервисным агентом KP_{rcv} (т.е. реализуется, что чтение — это непрерываемая деятельность).

Ответное уведомление: в зависимости от цели взаимодействия, агент KP_{snd} ожидает или нет ответного уведомления с результатом выполнения запрошенных действий.

Удаление: уведомление-запрос удаляется из *I* получателем KP_{rcv}, когда он выполнит необходимые действия. Уведомление-событие удаляется отправителем KP_{snd}, который сам определяет длительность нахождения уведомления в ИП.

Производительность: операция подписки является ресурсоемкой операцией, поэтому каждый агент *KP* должен ограничивать число одновременно работающих подписок. В предлагаемой модели обработка всех уведомлений для заданного варианта взаимодействия мо-

жет быть реализована с помощью одной подписки на следующие RDF тройки-шаблоны:

$$<id_x>, *, *; \quad (3)$$

или

$$*, \text{rdf:type}, <class_x>, \quad (4)$$

где маска $*$ представляет любое значение, $<id_x>$ — идентификатор уведомления, $<class_x>$ — класс онтологии, описывающий уведомление. Подписка (3) происходит на индивида, агент *KP* имеет свой уникальный идентификатор уведомления и подписка срабатывает при изменении свойств соответствующего индивида уведомления. Подписка (4) выполняется на онтологический класс и срабатывает при появлении или удалении индивида уведомления в *I*, а не при изменении свойств этого индивида. Такой вариант не привязан к идентификатору определенного уведомления.

Подписка (3) может использоваться в небольших системах, где достаточно завести один онтологический класс для уведомлений. Подписка (4) предназначена для систем, в которых каждый класс онтологии соответствует отдельному типу агента *KP*. Также, в (3), из-за подписки на фиксированный идентификатор уведомления (субъект тройки), невозможна отправка двух одинаковых уведомлений (т.к. предикат и объект тройки будут теми же самыми). Таким образом, требуется обработка и удаление из *I* уведомления, чтобы можно было отправить следующее. В (4) такого ограничения нет, поскольку индивиды уведомления имеют различные идентификаторы (субъекты тройки).

После получения основной тройки уведомления по подписке агент должен получить параметры. Их число влияет на производительность агента KP_{rcv} при обработке уведомления, как это будет рассмотрено далее в п. 5.

5. Применение модели. Предложенная модель уведомлений дает проблемно-ориентированный способ для организации взаимодействия агентов *KP*, применимый для широкого круга приложений ИП на платформе Smart-M3. Рассмотрим систему SmartScribo [22] в качестве примера. Система предназначена для мобильного мультиблоггинга: пользователи с персональных мобильных компьютеров взаимодействуют одновременно с несколькими блогами на внешних блог-сервисах. В ИП разделяется персональная информация пользователя и данные о его блогах. Архитектура системы представлена на рисунке 5. В системе выделяется три типа агентов *KP*: KP-клиент, блог-процессор и KP-посредник. KP-клиент реализует интерфейс пользователя. Блог-

процессор отвечает за доступ к заданному блог-сервису. Для каждого подключаемого к системе блог-сервиса нужен соответствующий блог-процессор. KP-посредник выполняет дополнительный анализ и обработку текущего содержимого ИП (напр., персонализированная рекомендация блогов для просмотра).

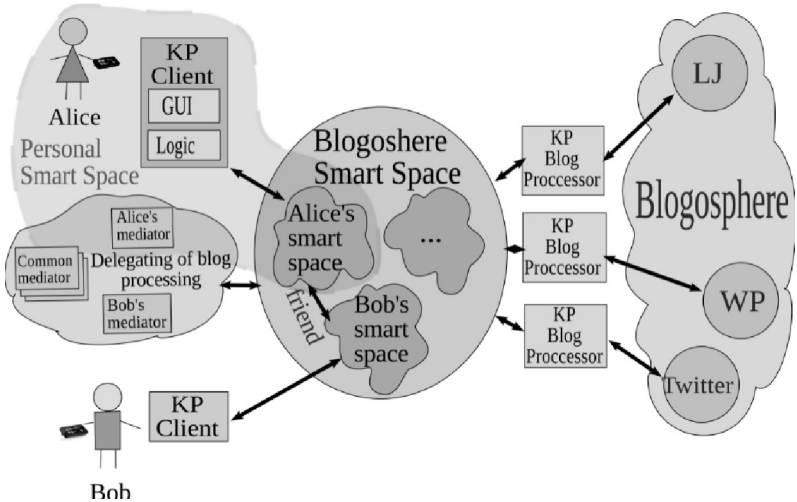

Рис. 5. Архитектура системы SmartScribo

Работа пользователя организована следующим образом. На клиентском агенте *KP* пользователь выбирает блоги для подключения. Блог-процессоры загружают посты с выбранных блогов. Затем пользователь может читать/изменять существующие посты или создавать новые. При создании нового поста или изменении существующего, блог-процессоры отражают изменения на блог-сервисах.

В ИП может храниться информация о многих постах и блогах различных пользователей. При использовании подписки на отдельные свойства индивидов постов, блог-процессор должен определить, какие посты были обновлены, а также когда обновленный индивид поста должен быть отправлен на блог-сервис. Например, у индивида поста есть такие свойства, как заголовок (title), текст, тэги. Блог-процессор может подписаться на тройку-шаблон: "<post_id>, title, ∗" и, аналогично, для других свойств поста. Он даже может подписаться на тройку "<post_id>, ∗, ∗". Однако, в обоих случаях агент *KP* будет получать обновления по подписке отдельно для каждого свойства и момент завершения изменения поста не определен явным образом.

Кроме того, посты динамически создаются и удаляются в ИП. Каждый раз устанавливать и снимать подписку — неэффективно. Хотя блог-процессор может подписаться на свойства, относящиеся к любому индивиду поста, например, "∗, title, ∗". Однако, подписка все равно будет информировать об изменении каждого свойства отдельно. В этом случае обновления будут относиться к различным индивидам поста. Для решения указанных проблем используют предложенную модель уведомлений. Она позволит информировать блог-процессор о необходимости выполнить требуемое действие с определенным индивидом поста.

Список уведомлений, используемых для осуществления взаимодействия между KP-клиентом и блог-процессором в системе SmartScribo, представлен в таблице 1. Все уведомления являются реактивными уведомлениями-запросами, т.к. отправляются после действий пользователя и требуют выполнения операций от блог-процессора. Каждое уведомление требует ответного уведомления, которое информирует пользователя о результате выполнения операции. В качестве параметров уведомлений используются идентификаторы индивидов: субъект тройки, описывающий определенную учетную запись, пост или комментарий.

Таблица 1. Список уведомлений в системе SmartScribo

Уведомление (вариант взаимодействия)	Параметры	Описание
refreshAccount	учетная запись	получить информацию об учетной записи блога
refreshPosts	учетная запись	получить посты заданной учетной записи
sendPost	учетная запись, пост	отправить пост
editPost	прошлый пост, новый пост	заменить прошлый пост на новый
delPost	учетная запись, пост	удалить пост
refreshComments	учетная запись	получить комментарии всех постов учетной записи
sendComment	учетная запись, комментарий, родительский объект	отправить комментарий на родительский объект (пост или комментарий)
delComment	учетная запись, комментарий, родительский объект	удалить комментарий у родительского объекта (поста или комментария)

Рассмотрим простое уведомление "refreshPosts". Оно имеет вид:

Notification-<service>, rdf:type, Notification
Notification-<service>, refreshPosts, <account_id>

Здесь <service> — тип блог сервиса (например "LJ" для LiveJournal), <account_id> — идентификатор индивида учетной записи. Различные идентификаторы сервисов используются для распределения уведомлений между различными блог-процессорами. Согласно (3) каждый блог-процессор подписывается на шаблон тройки, субъект которой — идентификатор сервиса, а предикат и объект – любые значения. Для блог-процессора, работающего с сервисом LiveJournal, шаблон тройки имеет вид "Notification-LJ, ∗, ∗".

Пользователь SmartScribo указывает учетные записи для блогов с помощью KP-клиента. Затем он может обновлять список постов для каждой учетной записи. KP-клиент публикует индивида учетной записи со всеми необходимыми для доступа к блог-сервису свойствами и публикует уведомление "refreshPosts". Блог-процессор получает уведомление по подписке, извлекает данные об учетной записи из ИП, получает список постов с блог-сервиса и удаляет уведомление. Затем блог-процессор публикует список постов в ИП и отправляет ответное уведомление для KP-клиента об удачном завершении операции.

Рассмотрим сложное уведомление "sendPost". Оно имеет вид:

Notification-<service>, rdf:type, Notification
Notification-<service>, sendPost, <notif_ind>
<notif_ind>, rdf:type, NotificationParameter
<notif_ind>, postAcc, <account_id>
<notif_ind>, postId, <post_id>

Здесь <service> — тип блог-сервиса, <notif_ind> — идентификатор дополнительного индивида уведомления, хранящего параметры уведомления, <account_id> — идентификатор индивида учетной записи, <post_id> — идентификатор индивида поста. Пользователь (на KP-клиенте) создает новый пост и публикует его и уведомление в ИП. Блог-процессор получает уведомление по подписке, извлекает идентификатор индивидов учетной записи и поста, получает свойства поста по идентификатору поста и отправляет пост на внешний блог-сервис. После этого блог-процессор удаляет обработанное уведомление и публикует ответное уведомление для KP-клиента.

6. Экспериментальная оценка производительности. Число параметров в уведомлении влияет на производительность приложения при обработке уведомления из-за особенностей операции подпис-

ки [9]. Рассмотрим экспериментальную оценку зависимости времени обработки уведомления от числа параметров.

Пусть *t(n)* — это время, прошедшее с момента отправки уведомления агентом KP_{snd} до получения результата агентом KP_{rcv}. Нами проведено два типа экспериментов: 1) все n параметров уведомления извлекаются одним запросом, 2) каждый параметр $i = 1,2, \ldots, n$ получается отдельным запросом (запрос из n итераций) тройки-шаблона:

$$<id_{ind}>, <p_i>, *.$$

С точки зрения производительности, первый тип соответствует наилучшему варианту реализации подписки агентом KP_{rcv} на уведомление с n параметрами. Второй тип демонстрирует производительность наихудшего случая, когда агент KP_{rcv} запускает запросы в цикле для получения всех n параметров. Пусть $t_{bst}(n)$ и $t_{wst}(n)$ — измеряемые оценки затрачиваемого времени. Таким образом, они будут показывать нижнюю и верхнюю границы производительности.

В экспериментах брокер SIB запущен на серверной ЭВМ. На персональном компьютере были запущены агенты KP_{snd} и KP_{rcv} (реализованы на языке Python). Выбранные серверная ЭВМ и персональный компьютер находятся в различных локальных сетях, в среднем RTT = 3 мс. В реальной ситуации агенты KP_{snd} и KP_{rcv} обычно запущены на разных компьютерах. Использование общего компьютера для агентов упростило измерение времени, исключив необходимость синхронизации времени. В то же время, каждый из этих агентов *KP* взаимодействует по сети непосредственно лишь с брокером SIB, а следовательно запуск агентов на одном компьютере достаточно близок к ситуации, когда эти агенты работают на двух одинаковых равноудаленных от серверной ЭВМ компьютерах.

Каждая итерация измерений состоит из шагов: 1) KP_{snd} отправляет уведомление с n-параметрами в SIB, 2) KP_{rcv} получает уведомление от брокера SIB, получает значения всех n параметров и удаляет уведомление. Выполнено по 100 измерений для каждого значения n, варьируемого от 1 до 100, с вычислением средних значений для $t_{bst}(n)$ и $t_{wst}(n)$. Результат представлен на рисунке 6 (а) для n усеченного до 50, чтобы растущая разность между $t_{bst}(n)$ и $t_{wst}(n)$ не скрыла детали.

Поведение $t_{bst}(n)$ и $t_{wst}(n)$ хорошо описывается линейной регрессией (построена для n от 1 до 100 с вычислением угловых коэффициентов k). Разница угловых коэффициентов показывает более быстрый рост $t_{wst}(n)$ по сравнению с $t_{bst}(n)$. Этот факт является следствием организации запроса в виде цикла. Такой циклический вариант также при-

водит к большей вариабельности для $t_{wst}(n)$ (для каждого n на графике показано среднеквадратическое отклонение).

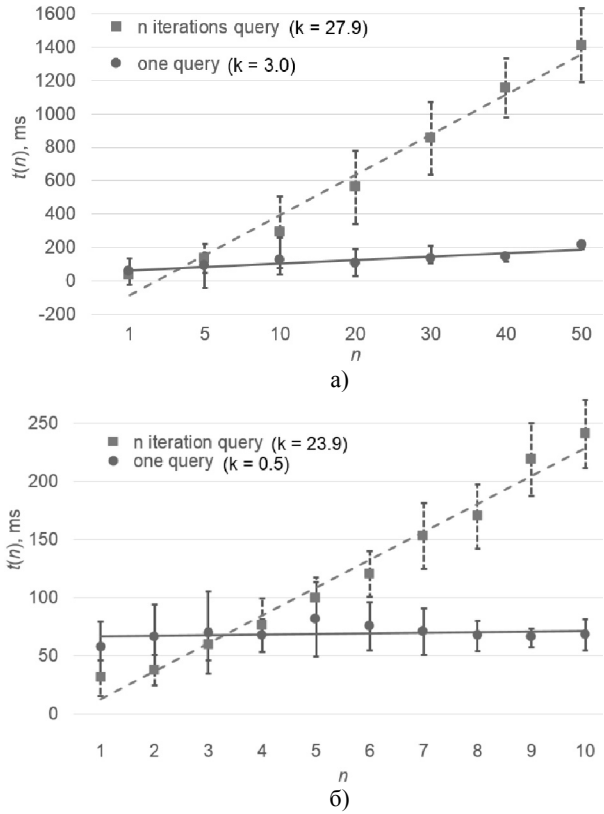

Рис. 1. Экспериментальное поведение: а) $t_{bst}(n)$; б) $t_{wst}(n)$

Наблюдаемый линейный рост $t_{bst}(n)$ имеет малый угловой коэффициент k. Наличие такого роста вызвано увеличением времени обработки данных на стороне SIB и необходимостью передачи большего объема данных по сети при увеличении n. В целом, эксперименты позволяют сделать вывод, что предложенная модель уведомлений сохраняет приемлемую производительность даже для больших значений n.

Разумно предполагать, что для большинства Smart-M3 приложений значение n ограничено, т.е. события, инициирующие взаимодействие, имеют компактное описание в силу особенностей предметной области. График на рисунке 6 (б) показывает измерения для всех

n = 1,2, … ,10 с построением линейной регрессии только для данного отрезка. Как и ранее, наблюдается линейное поведение, но угловые коэффициенты k меньше по сравнению со случаем для $n \leq 100$. При небольших n угловой коэффициент k линейной регрессии для $t_{bst}(n)$ практически равен нулю, т.е. временны*е* затраты близки к константе.

Отметим, что изменение угловых коэффициентов линейной регрессии при смене интервалов для n свидетельствует о наличии нелинейного эффекта в производительности. Так, для $t_{wst}(n)$ коэффициент k равен 27.9 и 23.9 при $n \leq 100$ и $n \leq 10$, соответственно. Причиной такого эффекта является нелинейность по n затрат времени на выполнение а) поисковых запросов (3) и (4) на стороне SIB и б) сетевой передачи данных между брокером SIB и агентом *KP*.

7. Заключение. В статье предложена модель уведомлений, предоставляющая общий проблемно-ориентированный способ для организации взаимодействия между агентами *KP* в рамках заданного программного приложения ИП на платформе Smart-M3. Модель уведомлений позволяет одним агентам *KP* информировать других агентов о наступлении событий или отправлять параметризованные запросы на выполнение действий. Все возможные варианты взаимодействия описываются в виде онтологии, расширяющей исходную онтологию приложения. Проведенные эксперименты подтверждают применимость модели при разработке Smart-M3 приложений.

Литература

1. *Cook D.J., Das S.K.* How smart are our environments? An updated look at the state of the art // Pervasive and mobile computing. 2007. vol. 3. no. 2. pp. 53–73.
2. *Chen H. et al.* Intelligent agents meet the semantic web in smart spaces // Internet Computing, IEEE. 2004. vol. 8. no. 6. pp. 69–79.
3. *Balandin S., Waris H.* Key properties in the development of smart spaces // Universal Access in Human-Computer Interaction. Intelligent and Ubiquitous Interaction Environments. Springer Berlin Heidelberg. 2009. pp. 3–12.
4. *Kiljander J. et al.* Enabling semantic technology empowered smart spaces // Journal of Computer Networks and Communications. 2012. vol. 2012. pp 1–14.
5. *Ovaska E., Cinotti T.S., Toninelli A.* The design principles and practices of interoperable smart spaces // Advanced Design Approaches to Emerging Software Systems: Principles, Methodology and Tools. 2012. pp. 18–47.
6. *Korzun D.G., Balandin S.I., Gurtov A.V.* Deployment of Smart Spaces in Internet of Things: Overview of the design challenges // Internet of Things, Smart Spaces, and Next Generation Networking. Springer Berlin Heidelberg. 2013. pp. 48–59.
7. *Honkola J. et al.* Smart-M3 information sharing platform // ISCC. 2010. pp. 1041–1046.
8. *Klyne G., Carroll J.J.* Resource description framework (RDF): Concepts and abstract syntax. 2006.

9. *Ломов А.А., Корзун Д.Ж.* Операция подписки для приложений в интеллектуальных пространствах платформы Smart-M3 // Труды СПИИРАН. 2012. Т. 4. № 23. С. 439–458.
10. *Corkill D.D.* Collaborating software: Blackboard and multi-agent systems & the future // Proceedings of the International Lisp Conference. 2003. vol. 10.
11. *Eugster P.T. et al.* The many faces of publish/subscribe // ACM Computing Surveys (CSUR). 2003. vol. 35. no. 2. pp. 114–131.
12. *Smirnov A. et al.* Anonymous agent coordination in smart spaces: State-of-the-art // Smart Spaces and Next Generation Wired/Wireless Networking. Springer Berlin Heidelberg. 2009. pp. 42–51.
13. *Carzaniga A., Rosenblum D.S., Wolf A.L.* Design and evaluation of a wide-area event notification service // ACM Transactions on Computer Systems (TOCS). 2001. vol. 19. no. 3. pp. 332–383.
14. *Городецкий В.И.* Агенты и извлечение знаний из данных в интеллектуальном пространстве. // Сборник трудов первой международной конференции «Автоматизация управления и интеллектуальные системы и среды». Терскол. 2010. С. 24–36.
15. *Shi D. et al.* An RDF-Based Publish/Subscribe System // Third International Conference on Semantics, Knowledge and Grid (SKG 2007). IEEE. 2007. pp. 342–345.
16. *Gutierrez C., Hurtado C., Mendelzon A.O.* Foundations of semantic web databases // Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems. ACM. 2004. pp. 95–106.
17. *Корзун Д.Ж., Ломов А.А., Ванаг П.И.* Автоматизированная модельно-ориентированная разработка программных агентов для интеллектуальных пространств на платформе Smart-M3 // Программная инженерия. 2012. № 5. С. 6–14.
18. *Palviainen M., Katasonov A.* Model and ontology-based development of smart space applications // Pervasive Computing and Communications Design and Deployment: Technologies, Trends, and Applications. 2011. pp. 126–149.
19. *Соловьев В.Д. и др.* Онтологии и тезаурусы. Учебное пособие // Казань, Москва. 2006. С. 75–90.
20. *Bechhofer S.* OWL: Web ontology language // Encyclopedia of Database Systems. Springer US. 2009. pp. 2008–2009.
21. *Ломов А.А.* Взаимодействие программного агента на уровне сессии с интеллектуальными пространствами // Ученые записки Петрозаводского государственного университета. 2013. № 8 (137). С. 118–121.
22. *Korzun D.G., Galov I.V., Balandin S.I.* Proactive personalized mobile multi-blogging service on Smart-M3 // CIT. Journal of Computing and Information Technology. 2012. vol. 20. no. 3. pp. 175–182.

References

1. Cook D.J., Das S.K. How smart are our environments? An updated look at the state of the art. Pervasive and mobile computing. 2007. vol. 3. no. 2. pp. 53–73.
2. Chen H. et al. Intelligent agents meet the semantic web in smart spaces. Internet Computing, IEEE. 2004. vol. 8. no. 6. pp. 69–79.
3. Balandin S., Waris H. Key properties in the development of smart spaces. Universal Access in Human-Computer Interaction. Intelligent and Ubiquitous Interaction Environments. Springer Berlin Heidelberg. 2009. pp. 3–12.
4. Kiljander J. et al. Enabling semantic technology empowered smart spaces. Journal of Computer Networks and Communications. 2012. vol. 2012. pp 1–14.
5. Ovaska E., Cinotti T.S., Toninelli A. The design principles and practices of interoperable smart spaces. Advanced Design Approaches to Emerging Software Systems: Principles, Methodology and Tools. 2012. pp. 18–47.

6. Korzun D.G., Balandin S.I., Gurtov A.V. Deployment of Smart Spaces in Internet of Things: Overview of the design challenges. Internet of Things, Smart Spaces, and Next Generation Networking. Springer Berlin Heidelberg. 2013. pp. 48–59.
7. Honkola J. et al. Smart-M3 information sharing platform. ISCC. 2010. pp. 1041–1046.
8. Klyne G., Carroll J.J. Resource description framework (RDF): Concepts and abstract syntax. 2006.
9. Lomov A.A., Korzun D.G. [Subscription operation for smart space applications on Smart-M3 platform]. *Trudy SPIIRAN – SPIIRAS Proceedings*. 2012. vol. 4. no. 23. pp. 439–458. (In Russ.).
10. Corkill D.D. Collaborating software: Blackboard and multi-agent systems & the future. Proceedings of the International Lisp Conference. 2003. vol. 10.
11. Eugster P.T. et al. The many faces of publish/subscribe. ACM Computing Surveys (CSUR). 2003. vol. 35. no. 2. pp. 114–131.
12. Smirnov A. et al. Anonymous agent coordination in smart spaces: State-of-the-art. Smart Spaces and Next Generation Wired/Wireless Networking. Springer Berlin Heidelberg. 2009. pp. 42–51.
13. Carzaniga A., Rosenblum D.S., Wolf A.L. Design and evaluation of a wide-area event notification service. ACM Transactions on Computer Systems (TOCS). 2001. vol. 19. no. 3. pp. 332–383.
14. Gorodetskiy V.I. [Agents and knowledge extracting from data in the smart space]. *Sbornik trudov pervoy mezhdunarodnoy konferentsii «Avtomatizatsiya upravleniya i intellektual'nye sistemy i sredy»* [Proceedings of the First International Conference «Control Automatization and Smart Systems and Environments»]. Terskol. 2010. pp. 24–36. (In Russ.).
15. Shi D. et al. An RDF-Based Publish/Subscribe System. Third International Conference on Semantics, Knowledge and Grid (SKG 2007). IEEE. 2007. pp. 342–345.
16. Gutierrez C., Hurtado C., Mendelzon A.O. Foundations of semantic web databases. Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems. ACM. 2004. pp. 95–106.
17. Korzun D.G., Lomov A.A., Vanag P.I. [Automated model-oriented development of software agents for smart spaces on Smart-M3 platform]. *Programmnaya inzheneriya – Software engineering*. 2012. no. 5. pp. 6–14. (In Russ.).
18. Palviainen M., Katasonov A. Model and ontology-based development of smart space applications. Pervasive Computing and Communications Design and Deployment: Technologies, Trends, and Applications. 2011. pp. 126–149.
19. Solovyev V.D. et al. *Ontologii i tezaurusy. Uchebnoe posobie* [Ontologies and thesauruses. Textbook]. Kazan', Moscow. 2006. pp. 75–90. (In Russ.).
20. Bechhofer S. OWL: Web ontology language. Encyclopedia of Database Systems. Springer US, 2009. pp. 2008–2009.
21. Lomov A.A. [Interaction of software agent on session level with smart spaces]. *Uchenye zapiski Petrozavodskogo gosudarstvennogo universiteta – Scientific notes of Petrozavodsk state university*. 2013. no. 8(137). pp. 118–121. (In Russ.).
22. Korzun D.G., Galov I.V., Balandin S.I. Proactive personalized mobile multi-blogging service on Smart-M3. CIT. Journal of Computing and Information Technology. 2012. vol. 20. no. 3. pp. 175–182.

Галов Иван Викторович — аспирант, младший научный сотрудник, кафедра информатики и математического обеспечения, математический факультет, Петрозаводский государственный университет (ПетрГУ). Область научных интересов: интеллектуальные пространства, семантический веб, онтологическое моделирование. Число научных публикаций — 23. galov@cs.karelia.ru; пр. Ленина, д. 33, г. Петрозаводск, 185910, РФ; р.т. +7(8142)711015.

Galov Ivan Viktorovich — Ph.D. student, researcher, Department of Computer Science, Faculty of Mathematics, Petrozavodsk State University (PetrSU). Research interests: smart spaces, semantic web, ontological modeling. The number of publications — 23. galov@cs.karelia.ru; Lenin St. 33, Petrozavodsk, 185910, Russia; phone +7(8142)711015.

Корзун Дмитрий Жоржевич — к-т физ.-мат. наук, доцент, ведущий научный сотрудник, кафедра информатики и математического обеспечения, математический факультет, Петрозаводский государственный университет (ПетрГУ). Область научных интересов: анализ распределенных систем, дискретное моделирование, повсеместные вычисления и интеллектуальные пространства, Интернет вещей, технологии разработки ПО, проектирование алгоритмов и вычислительная сложность, линейный диофантов анализ и его приложения, теория формальных языков и методы трансляции. Число научных публикаций — 129. dkorzun@cs.karelia.ru; пр. Ленина, д. 33, г. Петрозаводск, 185910, РФ; р.т. +7(8142)711084, факс +7(8142)711000.

Korzun Dmitry Georzhevich — Ph.D, associate professor, leader researcher, Department of Computer Science, Faculty of Mathematics, Petrozavodsk State University (PetrSU). Research interests: analysis and evaluation of distributed systems, discrete modeling, ubiquitous computing in smart spaces, Internet of Things, software engineering, algorithm design and complexity, linear Diophantine analysis and its applications, theory of formal languages and parsing. The number of publications — 129. dkorzun@cs.karelia.ru; Lenin St. 33, Petrozavodsk, 185910, Russia; office phone +7(8142)711084, fax +7(8142)711000.

Поддержка исследований. Исследование выполнено при финансовой поддержке Минобрнауки России по заданию № 2014/154 на выполнение государственных работ в сфере научной деятельности в рамках базовой части государственного задания, НИР № 1481. Работа выполнена при поддержке Программы стратегического развития ПетрГУ на 2012–2016 годы в рамках реализации комплекса мероприятий по развитию научно-исследовательской деятельности.

Acknowledgements. This research is financially supported by project #1481 from the basic part of state research assignment # 2014/154 of the Ministry of Education and Science of the Russian Federation. The article was published with financial support from the Strategic Development Program of Petrozavodsk State University.

РЕФЕРАТ

Галов И.В., Корзун Д.Ж. **Модель уведомлений для разработки программных приложений интеллектуальных пространств.**

Платформа Smart-M3 позволяет создавать программные приложения интеллектуальных пространств на основе набора взаимодействующих друг с другом агентов. Такое взаимодействие выполняется через общее разделяемое информационное содержимое. Для реализации некоторых видов взаимодействия удобно использовать операцию подписки. В этом случае агент выполняет подписку на определенные свойства классов из онтологии приложения. Однако не во всех предметных областях заданы свойства, изменение которых инициирует взаимодействие между агентами, и разработчику требуется вводить дополнительные свойства в онтологию приложения. В настоящее время нет общего проблемно-ориентированного подхода к выбору и добавлению таких свойств.

В работе предлагается модель уведомлений, которая определяет: (а) способ описания дополнительных онтологических классов и свойств для организации взаимодействия между двумя или более агентами на основе операции подписки и (б) схему использования этого онтологического описания при программировании агентов. Схема использования уведомлений состоит из следующих шагов: 1) отправка уведомления, 2) получение уведомления, 3) обработка уведомления, 4) отправка ответного уведомления, 5) получение ответного уведомления. В проведенном исследовании рассмотрены основные свойства уведомления: его представление, момент инициации, тип взаимодействия, наличие ответного уведомления. Авторами приведен пример применения модели уведомлений для разработки программной системы мобильного мультиблоггинга SmartScribo.

В статье также представлена экспериментальная оценка производительности взаимодействия, организуемого с помощью предложенной модели уведомлений. Показано, что скорость обработки уведомления при увеличении числа его параметров описывается линейной регрессией. Результаты экспериментов подтверждают применимость модели при разработке Smart-M3 приложений для развертывания в условиях реальных вычислительных сред.

SUMMARY

Galov I.V., Korzun D.G. **Notification model for smart space applications development.**

The Smart-M3 platform allows creating software applications of smart spaces based on a set of interacting agents. Interaction is organized via a shared informational content. For some types of interaction the subscription operation can be used. In this case, an agent establishes a subscription to certain properties from the application ontology. If the problem domain does not contain priory the properties the subscription requires, then the application developer has to introduce additional properties to the application ontology. Currently, there is no generic problem-oriented approach to selection and addition of such interaction-aware ontological properties.

This work proposes a notification model, which defines: (a) a way to describe additional ontological classes and properties needed for organizing the interaction between two or more agents based on the subscription operation and (b) a usage scheme of such ontological description in application agents programming. The scheme of notification usage consists of the following steps: 1) notification sending, 2) notification receiving, 3) notification processing, 4) reply notification sending, 5) reply notification reconvening. The study considers basic properties of a notification: its representation, initiation, type of interaction, reply notification. The authors discuss an example of applying the notification model in the development of practical software system SmartScribo for mobile multi-blogging.

The paper also introduces experimental performance evaluation of interactions organized using the proposed notification model. The processing time follows linear regression when increasing the number of parameters per notification. The experiment confirms applicability of the model for Smart-M3 application development for the case of real-life computing environments.