Лабораторная работа №3. Разработка простейшего приложения для взаимодействия с платформой Smart-M3

РАССМАТРИВАЕМЫЕ ВОПРОСЫ

1. Архитектура Smart-M3.
2. Доступ к интеллектуальному пространству.
3. Разработка агента KP.

1. АРХИТЕКТУРНОЕ ВИДИНИЕ SMART-M3

Платформа Smart-M3 является одним из вариантов реализации концепции интеллектуальных пространств. Она позволяет создать инфраструктуру, на основе которой распределенные информационные системы строят свои интеллектуальные пространства. Аббревиатура M3 объединяет три англоязычных термина: Multivendor, Multi-device и Multi-domain, что подчеркивает ориентацию платформы на положения большой индустриальной конвергенции, парадигмы повсеместных вычислений и возможности эффективного взаимодействия в IoT-средах. Постулируется независимость от конкретных производителей аппаратуры, типов участвующих устройств и предметных областей участвующих агентов. Потенциально допускается участие в распределенной системе произвольных IoT-устройств, независимо от производителя (напр., Nokia или Samsung), типа (напр., настольный компьютер или мобильный сенсор), назначения (напр., съемка видео или определение географических координат). Одно и то же устройство может рассматриваться как составленное из индивидуальных элементов, каждый из которых является полноправным участником (напр., экран, клавиатура и сенсорные элементы мобильного телефона). Далее интеллектуальную систему, разрабатываемую на базе платформы Smart-M3, будем называть Smart-M3 системой. Обмен информацией между агентами такой системы должен выполняться через интеллектуальное пространство — посредством общедоступного семантического информационного брокера с простым интерфейсом взаимодействия. Платформу можно разделить на две части: 1) инфраструктура и 2) инструменты разработки. Инфраструктура реализует семантических информационных брокеров, которые управляют доступом к информационному

содержимому интеллектуального пространства. Инструменты предназначены для разработчиков программного кода агентов.

В терминологии платформы Smart-M3 семантический информационный брокер обозначается аббревиатурой SIB (от англ. «Semantic Information Broker»), далее — брокер SIB, а информационный агент именуется KP (от англ. «Knowledge Processor»), далее — агент KP. Схема развертывания Smart-M3 системы представлена на рис. 1.1. Брокер SIB предоставляет доступ агентам KP к информационному содержимому общего пространства, обеспечивая их функциями обработки информации, такие как вставка, извлечение, редактирование, удаление и подписка. Брокер также управляет правами доступа. Информационное содержимое реализовано в виде отдельного хранилища, в котором вся информация сохраняется как RDF-граф, описывающий семантическую сеть с унифицированными идентификаторы ресурсов для узлов и дуг. Агенты KP — это программные модули, реализующие свои части прикладной логики распределенной системы. Взаимодействие KP и SIB следует специализированному прикладному протоколу SSAP (от англ. «Smart Space Access Protocol»).

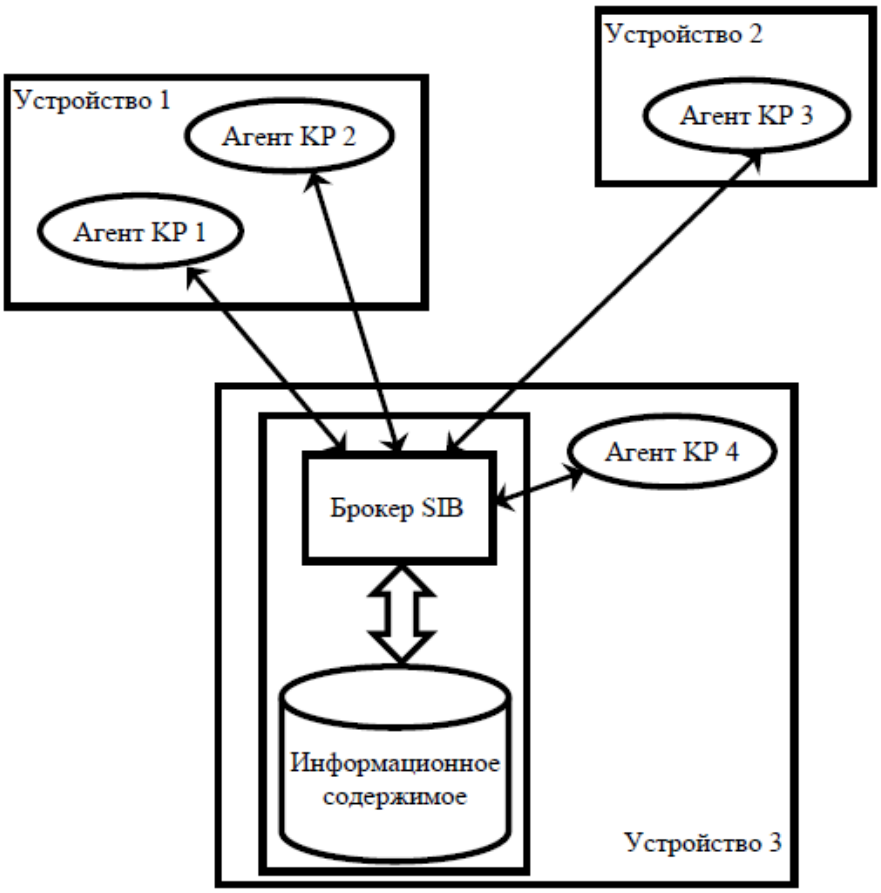

Рис. 1.1 – Схема развертывания Smart-M3

2. ДОСТУП К ИНТЕЛЛЕКТУАЛЬНОМУ ПРОСТРАНСТВУ

Протокол SSAP является XML-ориентированным, как и в большинстве современных web-сервисов, что унифицирует клиентскую часть, позволяя на основе единообразного формата взаимодействовать любому агенту KP с пространством. Формат XML является текстовым, сочетая читабельность и возможность машинной обработки. В то же время, для очень низкопроизводительных устройств текстовый формат может оказаться чрезмерно объемным. В настоящее время идут работы по разработке компактного бинарного формата для протокола SSAP.

Таблица 2.1 – Операции протокола SSAP

Операция	**Описание**
Join	Подключение к пространству по заданному имени и установка сессии взаимодействия с пространством. В типичном для IoT случае для идентификации также используется IP-адрес и порт сетевой ЭВМ, на которой работает брокер SIB.
Leave	Завершение сессии взаимодействия с пространством, т.е. агент KP явно выходит из пространства.
Insert	Публикация набора RDF-триплетов (RDF-графа) в информационном хранилище пространства. Выполняется атомарно, независимо от числа триплетов в наборе.
Remove	Удаление заданного набора RDF-триплетов (RDF-графа) из информационного хранилища пространства. Выполняется атомарно, независимо от числа триплетов в наборе.
Update	Обновление заданного набора RDF-триплетов (RDF-графа) в информационном хранилище пространства. Является атомарной комбинацией последовательного выполнения операций remove и insert.
Query	Поисковый запрос на выборку из информационного хранилища пространства. Возможно использование простейших запросов на основе шаблонов триплетов (используется маска типа «*») или языка семантических запросов SPARQL. Возвращается набор найденных RDF-триплетов.
Subscribe	Установка сеанса подписки на изменения заданного подмножества информационного содержимого. Интересуемое подмножество специфицируется с помощью шаблонно триплетов или языка SPARQL.

Операция	Описание
Unsubscribe	Завершение сеанса подписки.

Гарантируется, что при выполнении этих операций порядок их выполнения соответствует порядку их отправки заданным агентом KP. Операции доступа к SIB поддерживают семантические запросы на основе языка SPARQL. Тем самым, в интеллектуальном пространстве реализуется функциональность базы знаний на основе RDF-хранилища. Из содержимого по запросу могут извлекаться не только непосредственно хранимые данные и информация, но и выводные знания. Использование унифицированных идентификаторов ресурсов позволяет Smart-M3 системе рассматривать общее пространство как «именованное поиско-ориентированное расширение информационного содержимого» (от англ. «named search extend of information»). В отличие от принципа гигантского глобального графа, используемого в семантическом Веб, платформа Smart-M3 реализует в пространстве локализованную и динамически меняющуюся семантическую сеть.

Рекомендуется строить ее как некий информационный центр (от англ. «hub»), который определяет логические связи между всеми используемыми в данный момент системой знаниями. При этом само пространство, как правило, не хранит исходные объемные данные, а дает ссылки на их источники. Тем самым, реализуется эффективный способ интеграции данных от множественных неоднородных динамических источников.

3. ПРИМЕР АГЕНТА KP

```
public class SS_Agent implements iKPIC_subscribeHandler {
   private final KPICore kpi;
   private List<String> subscriptionIdList = new ArrayList<String>();
   private static Logger log = Logger.getLogger(SS_Agent.class.getName());

   public SS_Agent(String HOST, int PORT, String SMART_SPACE_NAME) {
      kpi = new KPICore(HOST, PORT, SMART_SPACE_NAME);
      if (log.isLoggable(Level.FINEST))
         kpi.enable_debug_message();
      if (log.isLoggable(Level.FINER))
         kpi.enable_error_message();
   }
```

```
   public String insert(String subject, String predicate, String object, String s_type, String objType) throws SmartSpaceException {
      String retXml;
      synchronized (kpi) {
         retXml = kpi.insert(subject,    // subject
               predicate,  // predicate
               object,     // object
               "uri",      // subject type
               objType);   // object type
         if (retXml != null && kpi.xmlTools.isInsertConfirmed(retXml)) {
            if (log.isLoggable(Level.FINE)) {
               log.fine(String.format("Insert of <%s, %s, %s> succeeded", subject, predicate, object));
            }
         } else {
            log.warning(String.format("Insert of <%s, %s, %s> failed", subject, predicate, object));
            throw new SmartSpaceException("Insert triple failed", retXml);
         }
      }
      return retXml;
   }

   public String remove(String subject, String predicate, String object, String objType) throws SmartSpaceException {
      String retXml;
      synchronized (kpi) {
         retXml = kpi.remove(subject,    // subject
               predicate,  // predicate
               object,     // object
               "uri",      // subject type
               objType);   // object type
         if (retXml != null && kpi.xmlTools.isRemoveConfirmed(retXml)) {
            if (log.isLoggable(Level.FINE)) {
               log.fine(String.format("Remove of <%s, %s, %s> succeeded", subject, predicate, object));
            }
         } else {
            log.warning(String.format("Remove of <%s, %s, %s> failed", subject, predicate, object));
            throw new SmartSpaceException("Insert triple failed", retXml);
         }
```

```
        }
        return retXml;
    }

    @Override
    public void kpic_SIBEventHandler(String s) {
        synchronized (kpi) {
            Vector<Vector<String>> deleted = kpi.xmlTools.getObsoleteResultEventTriple(s);
            Vector<Vector<String>> inserted = kpi.xmlTools.getNewResultEventTriple(s);
            String id = kpi.xmlTools.getSubscriptionID(s);
            if (!inserted.isEmpty()) {
                System.out.println("subject: " + inserted.get(0).get(0) + " predicate: " + inserted.get(0).get(1) + " object: "
+ inserted.get(0).get(2));
            }
        }
    }

    public void subscribe(String subject, String predicate, String object, String objectType) {
        synchronized (kpi) {

            String retXml = kpi.subscribeRDF(subject, predicate, object, objectType);
            if (retXml != null && kpi.xmlTools.isSubscriptionConfirmed(retXml)) {
// ОБРАТИТЕ ВНИМАНИЕ, КАК ПРАВИЛЬНО ОПРЕДЕЛЯЕТСЯ SUBSCRIPTION_ID
                String subscriptionId = kpi.xmlTools.getSubscriptionID(retXml);
                if (subscriptionId != null && !subscriptionId.isEmpty()) {
                    subscriptionIdList.add(kpi.xmlTools.getSubscriptionID(retXml));
                }
                if (log.isLoggable(Level.FINE)) {
                    log.fine(String.format("%s : Subscription of <%s> confirmed ", new Date(), predicate));
                }
            } else {
                if (log.isLoggable(Level.FINE)) {
                    log.fine(String.format("%s : Subscription of <%s> was aborted ", new Date(), predicate));
                }
            }

        }
    }
```

```
public void unSubscribeAll() {
    String retXml;
    synchronized (kpi) {
        for (String id : subscriptionIdList) {
            retXml = kpi.unsubscribe(id);
            if ((retXml != null && retXml.equals(""))) {
                if (log.isLoggable(Level.FINE)) {
                    log.fine(String.format("%s : Unsubscription is confirmed.", new Date()));
                }
            } else {
                if (log.isLoggable(Level.FINE)) {
                    log.fine(String.format("%s : Unsubscription isn't confirmed", new Date()));
                }
            }
        }
    }
    subscriptionIdList = new ArrayList<String>();
}

public void start() {
    String retXml = kpi.join();
    if (retXml != null && kpi.xmlTools.isJoinConfirmed(retXml)) {
        log.log(Level.FINE, "Successfully joined to smart space.");
    } else {
        log.log(Level.SEVERE, "Could not join to smart space. Will not function properly!");
        return;
    }
    kpi.setEventHandler(this);
}

public void stop() {
    // Cancel all the subscriptions
    unSubscribeAll();

    // Leave smart space
    String retXml = kpi.leave();
    if (retXml != null && kpi.xmlTools.isLeaveConfirmed(retXml)) {
        log.log(Level.FINE, "Successfully left smart space.");
```

```
    } else {
        log.log(Level.WARNING, "Could not leave smart space. It is not fatal but a little but annoying.");
    }
}

public List<String> getSubscriptionIdList() {
    return subscriptionIdList;
}

public void setSubscriptionIdList(List<String> subscriptionIdList) {
    this.subscriptionIdList = subscriptionIdList;
}

static class SmartSpaceException extends Exception {
    private String ssapResponse;

    SmartSpaceException(String msg, String ssap) {
        super(msg);
        ssapResponse = ssap;
    }

    public String getSsapResponse() {
        return ssapResponse;
    }
}

public static void main(String[] args) {

    SS_Agent service = new SS_Agent("192.168.56.101", 10010, "X");
    service.start();
    log.info("Started");

    service.subscribe(null, "http://cais.iias.spb.su/RDF/agent#is_a", null, "literal");

    BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));

    System.out.print("Type 'quit <Enter>' to stop service and leave.\n> ");
    try {
```

```
            String cmd = bufferedReader.readLine();
            while (true) {
                if (cmd.equals("exit") || cmd.equals("quit") || cmd.equals("stop")) {
                    break;
                }
                if (cmd.equals("insert")) {
                    try {
                        service.remove("bob@gmail.com", "http://cais.iias.spb.su/RDF/agent#is_a", "tourist", "literal");
                        service.insert("bob@gmail.com", "http://cais.iias.spb.su/RDF/agent#is_a", "tourist", "uri", "literal");
                    } catch (SmartSpaceException e) {
                        e.printStackTrace();
                    }
                }
                if (!cmd.isEmpty()) {
                    System.out.println(String.format("Unknown command: '%s'", cmd));
                }
                System.out.print("> ");
                cmd = bufferedReader.readLine();
            }
        } catch (IOException ex) {
            System.out.println("Can't read next command. Stopping...");
        } finally {
            service.stop();
        }
        log.log(Level.INFO, "Bye.");
    }
}
```

4. ВАРИАНТЫ ЗАДАНИЙ НА ЛАБОРАТОРНУЮ РАБОТУ

1. Создать систему учета сотрудников для филиалов организации. Система должна отвечать на вопрос о том, находится ли сотрудник в оплачиваемом отпуске, на больничном, в командировке, взял ли он отпуск за свой счет или выходит на работу в обычном режиме. Работник соответственно должен уведомлять систему о том, какого числа он ушел в отпуск, заболел и т.п., а так же какого числа он вернулся на работу. На основе этих данных, система должна оперативно произвести перерасчет заработной платы сотрудника, основываясь на данных его рабочей активности. Система должна состоять из двух самостоятельных компонентов: сервис учета сотрудников и клиента для сотрудника.

Рис. 4.1 – Схема работы сервиса для варианта №1

2. Создать систему подачи заявок в консульство или визовый центр для лиц, получающих туристическую визу. Система должна отправлять запросы от туристов на получение визы. В ответ на запрос со стороны туриста, визовый центр должен назначить ему время и дату для подачи документов. Кроме того, система должна ответить на запрос туриста о том, готова ли виза и время, а так же уведомит туриста о том ,когда ее можно получить. Система должна состоять из двух самостоятельных компонентов: сервис обработки заявок на визу и клиента для туриста.
3.

Рис. 4.2 – Схема работы сервиса для варианта №2