САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Математико-механический факультет

Кафедра системного программирования

Зубрилин Андрей Викторович

Облачная платформа для мультиагентных систем на основе SPADE и Google App Engine

Выпускная работа бакалавра

Допущена к защите.
Зав. кафедрой:
д.ф.-м.н., проф. А. Н. Терехов

Научный руководитель:
к.ф.-м.н., доцент Д. Ю. Бугайченко

Рецензент:
к. ф.-м. н., доцент И. П. Соловьев

Санкт-Петербург

2013

SAINT-PETERSBURG STATE UNIVERSITY

Mathematics&Mechanics Faculty

System engineering department

Zubrilin Andrei

Cloud multi-agent framework with SPADE and Google App Engine

Graduate paper

Admitted for defence.
Head of the chair:
A.N. Terehov

Scientific supervisor:
D.Y. Bugaychenko

Reviewer:
I. Soloviev

Saint-Petersburg

2013

Содержание

1. Введение

В наше время постоянно приходится решать задачи взаимодействия большого числа объектов. Сложность программного обеспечения корпоративного уровня неимоверно велика. Огромную роль в этом играет «масштабный фактор» - чем больше система, тем сложнее она устроена. Одним из способов упростить такие системы является мультиагентный подход. Мультиагентные системы [1] (далее МАС) позволяют сократить число внутренних связей в системе. Онлайн-торговля, логистика, сфера мобильных технологий, моделирование социальных структур – могут служить отличными примерами для использования МАС.

Первое появление МАС, как распределенной системы, датируется 1987 годом [2], но многие компании не знают или не хотят использовать готовые инструменты разработки МАС и по сей день. Вместо этого они прибегают к кустарным методам, реализуя свою систему, отдаленно напоминающую МАС. При этом тратится много времени и средств на разработку и тестирование, а результат, как правило, далек от идеала.

Так же в настоящее время широко распространены и весьма выгодны облачные вычисления [3]. Они замечательны тем, что компаниям не надо покупать дорогие серверы, которые 90% рабочего времени будут использовать только 10% своей мощности, а так же занимать уйму места. Платишь только за те вычислительные ресурсы, которые используешь в данный момент. Если более формально, то облачные сервисы являются моделью обеспечения повсеместного и удобного сетевого доступа по требованию к общему пулу конфигурируемых вычислительных ресурсов (серверам, устройствам хранения данных, приложениям и сервисам — как вместе, так и по отдельности), которые могут быть оперативно предоставлены

и освобождены с минимальными эксплуатационными затратами и/или обращениями к провайдеру. Потребители облачных вычислений могут значительно уменьшить расходы на инфраструктуру информационных технологий (в краткосрочном и среднесрочном планах) и гибко реагировать на изменения вычислительных потребностей.

Итак, у нас имеется две замечательные технологии, основными характеристиками, которых являются гибкость и масштабируемость. Сразу же возникает вопрос: что мешает нам использовать облачные вычислительные ресурсы в мультиагентных системах? Дело все в том, что облачные сервисы, услугами которых мы хотим воспользоваться, накладывают ограничения на приложения, а платформы для разработки МАС не соответствуют этим ограничениям. Поэтому в рамках данной работы была поставлена задача реализовать облачную платформу для создания мультиагентных систем.

2. Используемые сокращения

SPADE – Smart Python multi-Agent Development Environment – платформа для создания мультиагентных систем

MAC – мультиагентная система

GAE – Google App Engine

XMPP - Extensible Messaging and Presence Protocol - расширяемый протокол обмена сообщениями и информацией о присутствии

FIPA - Foundation for Intelligent Physical Agents – организация, занимающаяся стандартизацией для MAC

MTP – Media Transfer Protocol

AMS – Agent Management Service\System

DF – Directory Facilitator

P2P – Peer-2-Peer

AWS – Amazon Web Service

SPARQL - SPARQL Protocol and RDF Query Language

RDF - Resource Description Framework

JSON - JavaScript Object Notation)

BDI - belief–desire–intention software model

MUC - Multi-User Chat protocol расширение протокола XMPP, для реализации многопользовательского чата

3. Постановка задачи

В рамках данной работы были поставлены следующие задачи:

- Изучить современные инструменты и принципы разработки мультиагентных систем.

- Реализовать облачную платформу для создания мультиагентых систем (MAC) на основе SPADE и Google App Engine.

- Реализовать модельную задачу на данной платформе.

  В качестве модельной задачи была рассмотрена, типичная для данной предметной области, задача составления расписания для конференций.

4. Обзор

4.1 Мультиагентные системы (MAC)

Мультиагентные системы – это системы, образованные несколькими взаимодействующими интеллектуальными агентами. Как правило, такие системы используются для работы с постоянно растущей децентрализованной средой. Базовым понятием, лежащим в основе мультиагентной теории, является понятие агента – в общем смысле, это любой объект способный действовать и воспринимать.

Наиболее признанным определением термина "Агент" является программная система, обладающая следующими свойствами [4]:

- Автономность: агенты функционируют без прямого вмешательства людей и кого-либо другого и обладают определенной способностью контролировать свои действия и внутреннее состояние.

- Социальность: агенты взаимодействуют с другими агентами (возможно людьми) посредством какого-либо коммуникационного языка.

- Реактивность: агенты обладают способностью воспринимать среду (которая может быть физическим миром, пользователем, взаимодействующим посредством графического интерфейса, коллекцией других агентов, Интернетом, или всем вместе взятым) и адекватно реагировать в определенных временных рамках на происходящие изменения.

- Проактивность – любой агент мультиагентной системы, обладая целенаправленным поведением, может проявлять инициативу путем переключения целей, изменения приоритетов управления и режимов работы, способен планировать свои взаимоотношения с внешней средой, совершать действия, направленные на достижение целей.

Существует большое количество определений понятия "Агент" в зависимости от взгляда на распределенную обработку знаний. С точки зрения распределенных вычислений, агент - это самостоятельный процесс, выполняемый параллельно, имеющий определенное состояние и способный взаимодействовать с другими агентами посредством передачи сообщений.

Преимущества МАС:

- Гибкость - мультиагентная система может быть дополнена и модифицирована без переписывания значительной части программы.

- Автономность - отсутствие в потребности постоянного контроля пользователя.

- Высокая производительность и масштабируемость решений.

- Самовосстановление и устойчивость к сбоям (благодаря самоорганизации).

4.2 Платформа SPADE

SPADE (Smart Python multi-Agent Development Environment) – платформа для создания мультиагентных систем (МАС), основанная на XMPP/Jabber технологии и написанная на высокоуровневом языке Python. Данная платформа содержит в себе множество функций и средств которые упрощают построение МАС. Основным протоколом взаимодействия агентов является XMPP (The Extensible Messaging and Presence Protocol), ранее известный как Jabber, снованный на XML, свободный для использования протокол для мгновенного обмена сообщениями и информацией. Данная технология соответствует FIPA-ACL стандарту (FIPA – организация, занимающаяся составлением стандартов для технологий основанных на взаимодействие агентов).

Данная платформа является одной из наиболее распространенных, и первой, в которой стали использовать XMPP технологию.

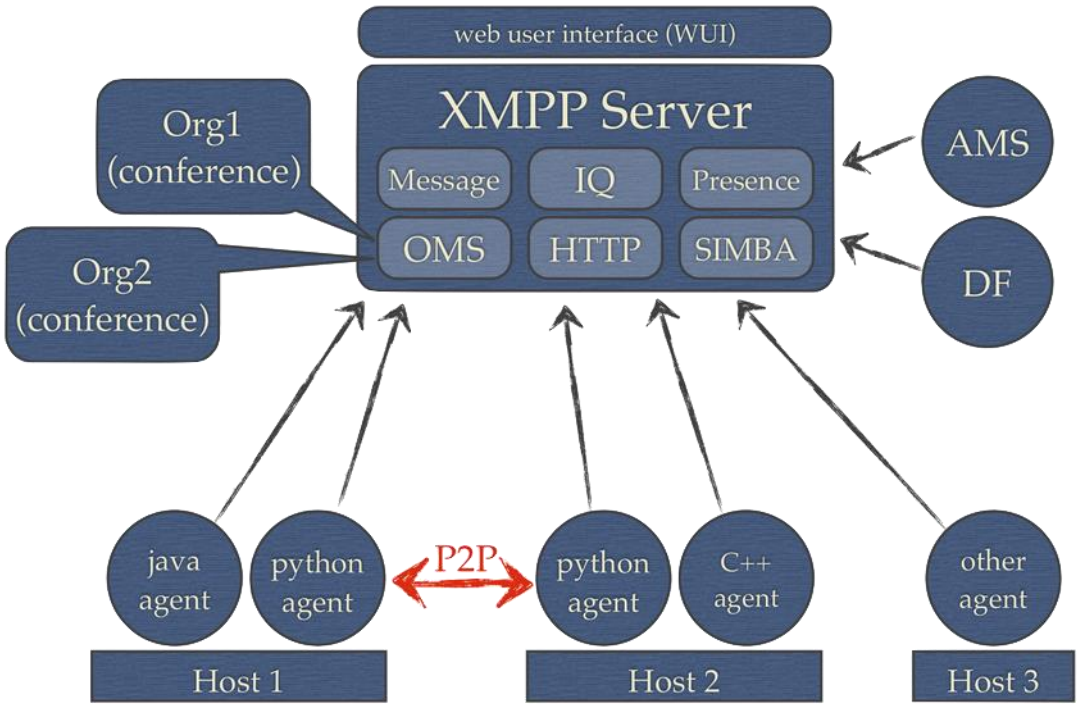

Рис.1 Архитектура SPADE

На изображении (Рис.1) представлена архитектура данной платформы.

Как мы видим, основным элементом является XMPP-сервер, через который агенты общаются друг с другом. Можно заметить, что SPADE'у без разницы на каком языке написан ваш агент, единственное требование - это поддержка FIPA-стандарта и одного из MTP (Message Transfer Protocol) протоколов: HTTP, XMPP, SIMBA или P2P.

Обозначения:

AMS, DF, Org1, Org2 – являются агентами,

AMS - Agent Management Services – управляет созданием и удалением агентов, ведет контроль за миграцией агентов, т.е. контролирует весь жизненный цикл агента.

DF - Directory Facilitator – компонент, предназначенный для использования агентами желтых страниц, т.е. сервисы который ведет учет всех зарегистрированных агентов.

Org1, Org2 – виртуальные конференции, основанные на протоколе MUC

4.3 Google App Engine

Google App Engine — облачная платформа - сервис хостинг сайтов и web-приложений на серверах Google [5]. Приложения, разворачиваемые на базе App Engine, должны быть написаны на Python, Java либо Go. Использование службы аккаунтов Google позволяет быстро начать работу с приложением, нет необходимости проводить отдельную регистрацию учётных данных на каждом сайте. Это также позволяет разработчику не заботиться о реализации ещё одной системы регистрации пользователей специально для своего приложения. В отличие от многих обычных размещений приложений на виртуальных машинах, таких как Amazon EC, платформа App Engine тесно интегрирована с приложениями и накладывает на разработчиков некоторые ограничения. Конкурирующие среды позволяют оперировать множеством программного обеспечения, созданного под *NIX системами, в то время как App Engine требует от разработчика обязательного использования языков программирования Python, Java или Go и сохранения информации в собственном хранилище (Datastore).

Ограничения

- Нет полного контроля операционной системы
- Нет доступа на запись в файловую систему
- Ограничения, накладываемые на код (импорт, использование дополнительных модулей, потоки)

4.3.1 Сравнение с аналогичными платформами

Изначально, при выборе облачной платформы, рассматривались три наиболее популярных сервиса в данной области: Windows Azure, Amazon EC2, Google App Engine. К сожалению, несмотря на большой выбор служб и хорошую поддержку .NET у Windows Azure [6], от данного сервиса пришлось сразу отказаться, ввиду того, что на текущий момент не существует адекватных средств для создания MAC на .NET.

Если сравнивать платформу GAE с Amazon EC [7], то последняя, грубо говоря, предоставляет нам простую виртуальную машину, с которой мы делаем, что хотим, в то время как Google предоставляет нам среду разработки, к которой мы должны подстраиваться, но которая сама по себе облегчает нам масштабирование и предоставляет ряд дополнительных сервисов.

Таблица 1 – Таблица сравнения платформ

Требование	**Google App Engine (GAE)**	**Amazon Web Services (AWS)**	**Победитель**
Лёгкая и быстрая разработка	**Да**	Нет	GAE
Стандартность и уверенность в будущем	Ограниченная	**Да**	AWS
Высокая доступность	Да	Да	Ничья
Лёгкость в обслуживании	**Да**	Нет	GAE
Лёгкость выкладывания сборок	**Да**	Да, с дополнительной работой	GAE
Лёгкость масштабирования	**Да**	Да, можно сделать	GAE
Гибкость	Ограниченно	**Да**	AWS

Google App Engine очень хорошо соответствует большинству требований выдвигаемых пользователями. GAE — это комплексная платформа, делающая разработку крайне быстрой и лёгкой, так как она предоставляет набор инструментов и средств управления, которые позволяют безболезненно создавать и выкладывать программы. Указанная платформа имеет готовые API для логина, хранения и индексации данных, отправки писем и даже для мемкешинга.

Так же, если рассмотреть данные платформы с точки зрения экономического фактора, выбор становится очевиден: во-первых, это объёмы вычислительных ресурсов, достаточные как минимум для тестирования, которые у Amazon стоят денег, Google предоставляет бесплатно, и в дополнение к ним ряд приложений, облегчающих разработку [8].

4.4 Сравнение технологий разработки SPADE и JADE

Перед началом разработки облачной платформы нам был представлен выбор между двумя популярными средствами для создания мультиагентных систем: SPADE (Smart Python multi-Agent Development Environment) и JADE (Java Agent DEvelopment Framework).

Рассмотрим по отдельности характеристики обоих технологий.

JADE:

Плюсы:

- Поддержка основных типов агентов
- Онтологии
- Встроенное GUI для работы с агентами
- Большое комьюнити
- Очень подробные мануалы и документация
- Поддержка стандарта FIPA
- Основные протокол HTTP и XMPP
- Хорошо показал себя в мобильных технологиях, используется в Telecom Italia, Motorola, France Telecom R&D

Минусы:

- Изначально нет поддержки BDI - модели
- Более слабая поддержка Java runtime по сравнению с Python в Google App Engine, сложность интеграции заключается в том, что поддержка потоков осуществляется только в backend-приложениях

Вывод: как платформу, так и агенты мы можем использовать только как backend-приложения в сервисе Google.

SPADE:

Плюсы:

- Поддержка основных типов агентов
- Онтологии
- Web UI для работы с агентами
- Поддержка стандарта FIPA
- Основной протокол XMPP, поддержка HTTP, SIMBA, P2P
- Используется многими университетами по всему миру
- Хорошая поддержка в GAE Python

- Поддержка BDI – модели
- Виртуальные конференции на основе MUC

Минусы:

- Слабое комьюнити по сравнению с JADE
- Небольшой и неполный User Manual
- Ориентирован на Linux OS (Плохая совместимость с Windows OS в последних версиях)
- Часть шаблонов и примеров использования разбросана по разным ссылкам\местам хранения

Не смотря на слабую поддержку комьюнити решающим фактором в выборе технологии явились предпочтения разработчиков Google Python. Благодаря этому основная часть нашей платформы будет являться backend-приложением. В frontend мы можем использовать как One-Shot (One-Task) и Event агенты, так и обычные агенты, но с одним условием: т.к. frontend-приложение работает только пока есть http-запрос (15 секунд после последнего), то по закрытию приложения нам придется сохранять последнее состояние нашего агента в одном из предоставляемых Google сервисах хранения данных, а при следующем запуске приложения восстанавливать наше состояние. Естественно такому типу агентов нельзя будет поручать задания вроде мониторинга, но как для агентов-клиентов, например в онлайн-торговле, такой тип будет более чем достаточен, ведь когда пользователь оффлайн, его агент тоже должен «засыпать».

4.5 Модельная задача

Как уже было сказано в качестве модельной задачи была взята задача составления расписания для конференций:

Любой человек, который хоть раз бывал на конференциях, несомненно сталкивался с такой ситуацией, когда два интересных ему доклада идут одновременно в разных потоках. Возникает вопрос: о чем думала организация, когда поставила эти два доклада в одно и тоже время?! В данной работе мы, как раз, и попытаемся помочь организаторам конференций минимизировать число недовольных людей.

5. Реализация

5.1 Модельная задача, описание, анализ

5.1.1 Описание и анализ задачи

В предыдущем пункте мы рассмотрели проблему, которая возникает на конференциях. Как угодить большинству людей, чтобы интересные им доклады не шли в одно время?

Для ответа на данный вопрос хотелось бы выделить решающие факторы, как со стороны организации, так и со стороны слушателя, и выступающего.

Во-первых, к чему стремится организация данных мероприятий? Конечно же к тому, чтобы спрос и интерес большинства слушателей был удовлетворен, чтобы все выступающие были довольны, и чтобы ни один слушатель не пропустил интересный ему доклад.

Во-вторых, выступающие. Их первостепенной целью является проявление интереса слушателей к выступлению.

Что же касается, слушателя, то у него всего два фактора: получить интересующую их информацию, и пообщаться со специалистами в данной области.

Из перечисленных потребностей мы можем выделить параметры, которые будут играть основную роль:

- Именитость выступающего (Известность)
- Интерес доклада со стороны слушателей (Спрос)
- Актуальность\Новизна доклада

Рассмотрим последний пункт. Как правило некоторые инновационные идеи, которые содержатся в новых докладах, не доходят до обычного слушателя. Слушатель не считает этот доклад привлекательным для затраты своего времени, несмотря на то, что, возможно, доклад будет ему очень познавателен. Поэтому организаторы должны выделять такие доклады, у которых действительно интересное содержание, но малый спрос, и перемешивать с докладами, которые наиболее интересны слушателям. Как раз благодаря такому смешиванию, мы сможем лучше и более равномерно распределить и доклады, и слушателей по всем потокам.

5.1.2 Модельная задача, мультиагентный подход, SPADE.

К сожалению, задача данного типа является NP-полной задачей, поэтому мы не можем предложить алгоритм для получения идеального результата за короткие сроки, когда все слушатели будут довольны. Приблизиться к оптимальному решению нам поможет мультиагентный подход [9].

Соответственно, создадим ряд агентов:

- conference_manager – данный агент будет регистрировать для себя агентов слушателей и докладчиков, а так же составлять расписание в зависимости от запроса пользователей
- listener – агент-слушатель, в его задачи входит оставить или изменить свои пожелания, и запросить текущее расписание конференции, а так же предложить изменения текущего расписания
- reporter - агент-докладчик, в его задачи входит предоставление темы доклада, а так же по данному агенту менеджер конференций определяет именитость докладчика, так же докладчик имеет все функции слушателя

Помимо простых действий, таких как, запросить расписание или оставить пожелание, агенты listener и reporter должны будут оценивать расписание и, в случае пересечения интересных докладов, предлагать оптимальную для них замену.

Опишем работу данной системы:

Этап 1:

У нас в системе находится только менеджер конференций, постепенно регистрируются агенты Слушатель и Докладчик, и оставляют свои пожелания.

Этап 2:

Настает момент, когда считается, что все слушатели и докладчики зарегистрированы, агент-менеджер расставляет доклады в случайном порядке и рассылает всем агентам расписание для оценки. Если за

изменение расписания проголосует больше 50% агентов, то данная опция будет разрешена, иначе переходим к Этапу 4.

Этап 3:

За изменение проголосовало больше 50% агентов, соответственно теперь агенты предлагают доклады, которые им интересны, переставить следующим образом, Рис.2:

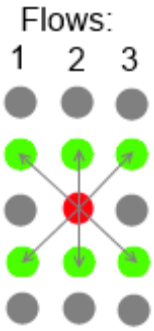

Рис.2 Красный круг – доклад, который хотим переставить, Зеленые круги – доклады с которыми хотим меняться, Flows – потоки, для наглядности рассматриваем 3 потока.

Как можно заметить, при реализации данной задачи мы рассматривали только окрестность из ближайших докладов. Естественно, если ее увеличить, то конечный результат возрастет, но и сложность задачи также возрастет на порядок.

В итоге, после того, как агенты предложили свои перестановки, агент-менеджер совершает наиболее популярные из них и снова делает опрос, данный процесс будет повторяться до тех пор пока не будет нужного числа положительных отзывов о расписании (например 51%) или не более N (например 10) итераций.

Этап 4:

Расписание утверждено, нельзя вносить изменения, все готово для проведения конференции.

Далее представлен график, демонстрирующий результаты работы данной системы, в качестве входных данных были взяты данные небольшой однодневной конференции.

Входные данные:

- ***reports_qt = 30*** – количество докладов
- ***flow_qt = 3*** – количество потоков
- ***listener_qt = 130*** – количество слушателей
- ***listener_pref_qt = 5*** – количество предпочитаемых докладов у одного слушателя
- ***run_qt = 50*** – количество запусков

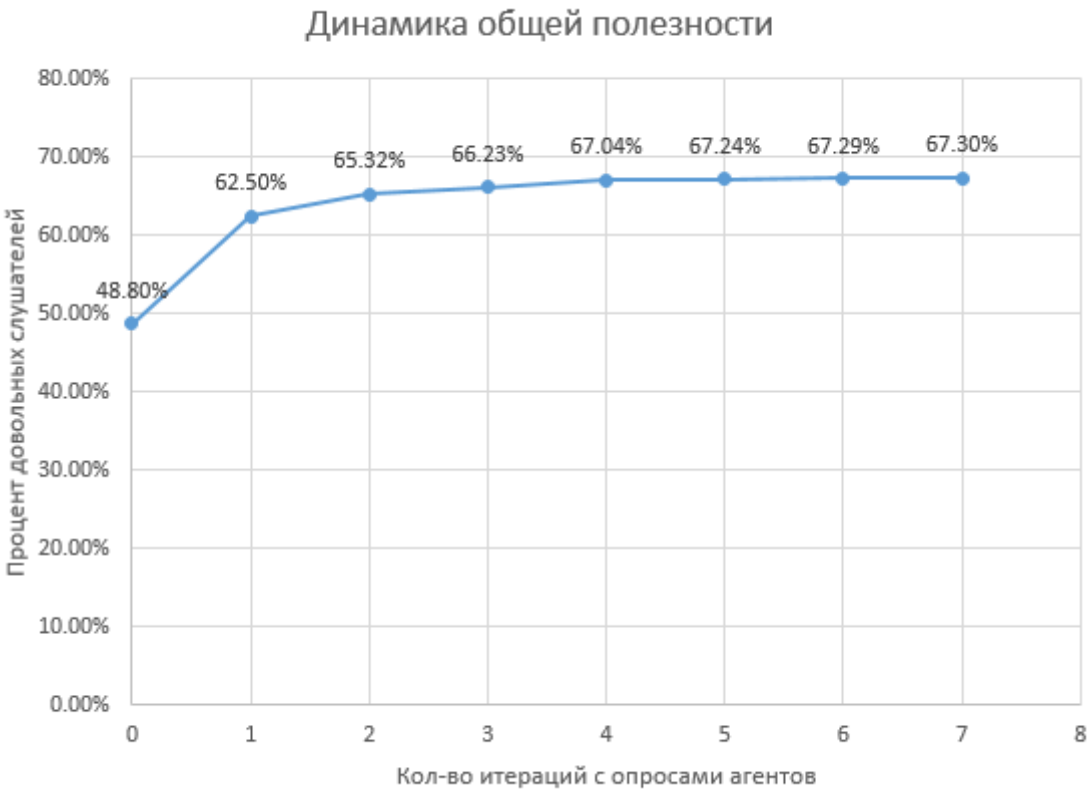

Рис.3 Динамика общей полезности по результатам симуляции по мере увеличения числа итераций, усредненная после 50 прогонов

На графике показаны средние значения каждой итерации после 50 тестов. Хочется отметить минимальные и максимальные значения при старте и по окончании работы системы во время тестирования на Рис.4.

	Min	Max
Start	42.50%	52.40%
End	65.40%	72.30%

Рис.4 Максимальные и минимальные значения после тестирования

Таким образом можно заметить, что среднее значение по графику составило 18.5%. Так же хочется отметить, что после 4 итерации изменения видны только в 1-2 случаях из 10, поэтому, если входные данные велики, то не имеет смысла проводить большее количество опросов. Разность максимального и минимального значения на последней итерации составила 6.9%, это значение говорит нам о том, что при увеличении сложности алгоритма перестановки докладов можно достичь более высокого и стабильного значения, но, естественно, в ущерб производительности.

Архитектура системы выглядит следующим образом Рис.5:

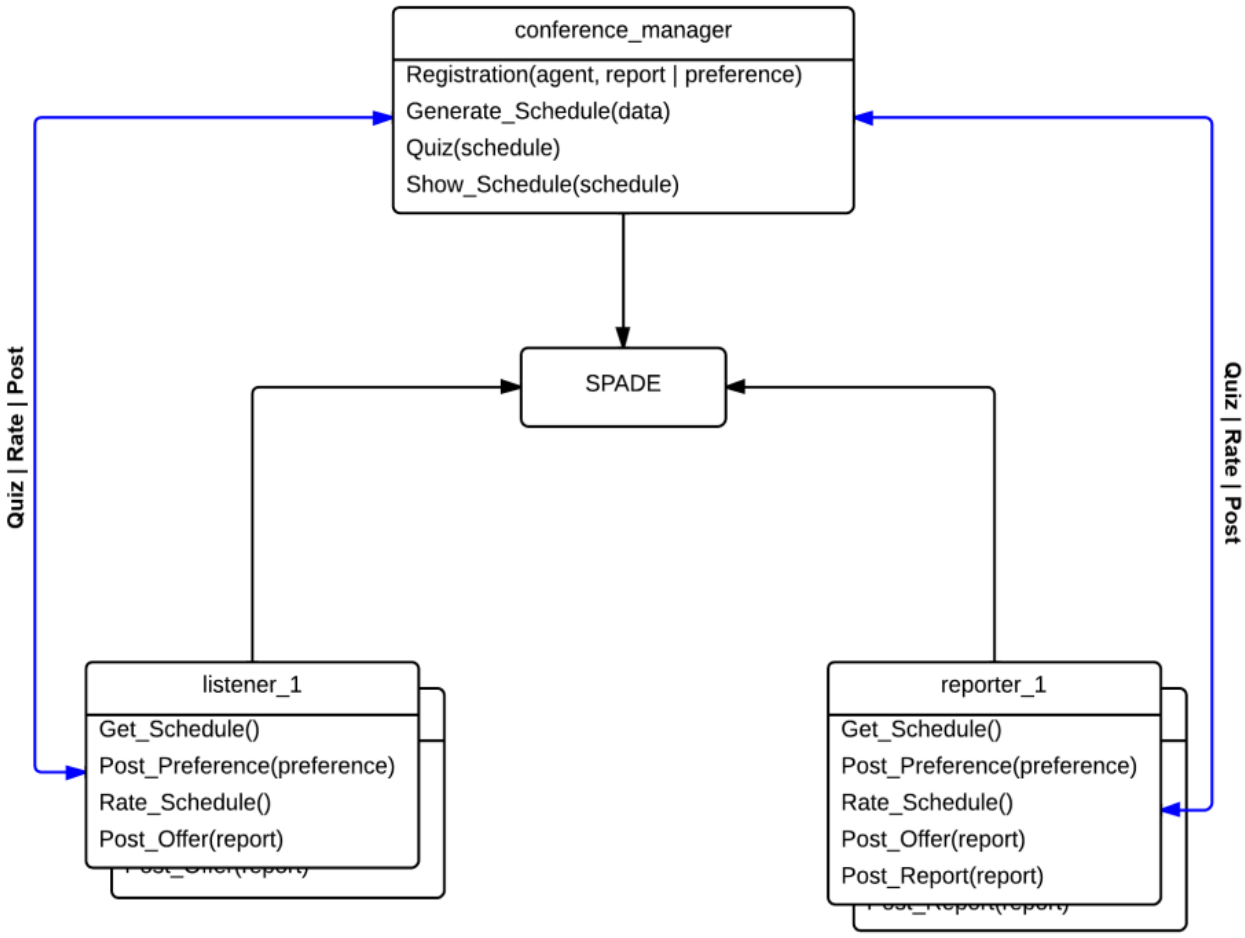

Рис.5 Архитектура системы для решения задачи о составление расписания для конференций.

Синий стрелкой отдельно обозначен 3й Этап общения: Опрос -> Оценка -> Варианты изменения.

5.2 Реализация облачной платформы, проблемы, ограничения.

5.2.1 Ограничения, проблемы:

Компания Google предоставляет в пользование свои облачные серверы, но при этом вводит ограничения на используемые приложения [10]. Одним из таких ограничений, хотя скорее неудобств, является конвертация импорта, ввиду того, что исходный код платформы SPADE разделен на множество 1-килобайтных файлов, в которых часть импорта нужно изменить, а часть попросту удалить. Данный факт весьма замедляет интеграцию с облачными сервисами.

Другим, уже весьма проблемным ограничением является то, что хотя GAE и поддерживает большинство библиотек Python'а, но добавление тех, которые не поддерживает, требует большого количества времени. В нашем случае SPADE как раз использует одну из таких библиотек, и попросту ее откинуть не получится. Этой библиотекой оказался SPARQLWrapper – SPARQl – язык запросов данных, коммуникация на основе данного языка является одной из основных при взаимодействие агентов. Но в данном случае, вариант с полным переносом модуля в саму платформу решает данную проблему.

Так же был выявлен конфликт веб модулей обеих платформ, и ввиду не возможности изменения Google API пришлось частично вырезать web UI из SPADE. С другой стороны, данная проблема имеет и положительный оттенок, т.к. в модуле от разработчиков Google на порядок больше возможностей, хотя скорее, функционал SPADE был весьма беден (разработчики только начали развивать web UI), поэтому потеря была не велика.

Проделав вышеописанные операции, которые хоть и занимают мало места на листе бумаги, но требуют порядочной усидчивости, был предпринят очередной тестовый запуск в среде App Engine Environment. Данную среду Google предоставляет для разработки приложений, которая теоретически полностью эмулирует серверную среду. Тестовый запуск провалился, проблемой оказалось то, что на тот момент данная среда не поддерживала backend-приложения, кстати говоря, это противоречило документации. В связи с этим было решено добавить веб-интерфейс с помощью которого можно было бы запустить данное приложение. Результат оказался успешным, но не полностью. Было выявлено, что данная среда использует стандартные библиотеки пользователя, в то время как на стороне сервиса, они частично изменены. В тоже время SPADE тоже использует видоизмененный под себя

модуль. В данном случае камнем преткновения стал XMPP модуль, который оказался итоговой и самой большой проблемой при интеграции двух платформ. Для решения этой проблемы часть функционала пришлось заменить на функционал GAE.

Еще одним ограничением, усложняющим использование приложений в данном сервисе является то, что приложения не идентифицируются уникально по ip-адресу. Этот факт затрудняет взаимодействие с внешними сервисами, так же нет поддержки SSL.

В тоже время ограничение на использования CPU для данного вида систем является одним из решающих факторов, и хотя к нашему случаю это не относится, т.к. задача не требует большого выделения ресурсов, но если в будущем развивать систему, то потребуется разбивать платформу на несколько приложений. В этом случае разработчики Google так же предусмотрели отличное взаимодействие между приложениями, поэтому особых проблем возникнуть не должно.

5.2.2 Особенности реализации

В рамках реализации данной платформы был модернизирован XMPP [11] модуль для устранения конфликта совместимости двух технологий, так же был заменен web UI исходной платформы на предоставляемый Google, и налажено их взаимодействие.

Был добавлен и настроен пакет SPARQLWrapper. SPARQL- язык запросов данных, представленных по модели RDF. Коммуникация, основанная на SPARQL, является одной из технологий семантической паутины, а так же одной из основных при взаимодействии агентов.

Так же был добавлен модуль simplejson, предназначенный для конвертации JSON - текстовый формат обмена данными, основанный на Java Script. Несмотря на происхождение от JavaScript, формат считается языконезависимым и может использоваться практически с любым языком программирования. Данный формат очень хорошо зарекомендовал себя в MAC. За счёт своей лаконичности по сравнению с XML, формат JSON может быть более подходящим для сериализации сложных структур.

6. Заключение

В рамках данной работы были достигнуты следующие результаты:

1) Изучены современные инструменты и особенности разработки мультиагентых систем: SPADE, JADE
2) Изучены облачные сервисы: Google App Engine, Amazon EC2, Windows Azure
3) Реализована платформа для создания мультиагентых систем
4) Реализована и протестирована на различных данных модельная задача составления расписания для конференций

7. Список литературы

[1] *Wooldridge M.J. An Introduction to Multiagent Systems. Wiley. 2009.*

[2] *A Flexible Testbed for Distributed AI Research. http://www.agent.ai/doc/upload/200302/gass87_1.pdf*

[3] *DOT.CLOUD: The 21st Century Business Platform Built on Cloud Computing, Fingar P. 2009.*

[4] *Смирнов А. В., Шереметов Л. Б. Многоагентная технология проектирования сложных систем. 1999.*

[5] *Developing with Google App Engine. Eugene Ciurana. 2009.*

[6] *Windows Azure. http://www.windowsazure.com/en-us/*

[7] *Google App Engine, Amazon Web Services and the Cloud.*

http://consultingblogs.emc.com/jaddy/archive/2010/04/09/google-app-engine-amazon-web-services-and-the-cloud.aspx

[8] *Google App Engine. https://developers.google.com/appengine/*

[9] *Рассел С., Норвиг П. Искусственный интеллект. Современный подход. Вильямс. 2006 (2007)*

[10] *What are the pros and cons of using Google App engine for my startup?*

https://groups.google.com/forum/?fromgroups#!topic/google-appengine/EjQleHJzrzI

[11] *XMPP technology. http://xmpp.org/about-xmpp/*