УНИВЕРСИТЕТ ПО БИБЛИОТЕКОЗНАНИЕ И ИНФОРМАЦИОННИ ТЕХНОЛОГИИ
КАТЕДРА ”НАЦИОНАЛНА СИГУРНОСТ”
СПЕЦИАЛНОСТ «ИНФОРМАЦИОННА СИГУРНОСТ»

МАГИСТЪРСКА ТЕЗА

на тема:

ЕКСПЕРИМЕНТИ С МНОГОАГЕНТНИ СИСТЕМИ ЗА JAVA УПРАВЛЕНИЕ НА БИЗНЕС ПРОЦЕСИ

Изготвил:
Димитър Куменов
Редовно обучение
Ф.№ 046-исмр

Научен ръководител:.....
(Проф. д.ик.н. Владимир Йоцов)

София
2017

Резюме

Куменов Д. Експерименти с многоагентни системи за Java управление на бизнес процеси. Научен ръководител Проф. д.ик.н. Владимир Йоцов.С.2017.УниБит. Факултет по информационни науки. Катедра „Национална сигурност“. Специалност „ Информационна сигурност“.

Целите на магистърската теза са: Представяне на системите за управление на бизнес процеси и положителното им влияние върху качеството на изпълнение на процесите. Предимства на многоагентните системи при изграждането на системи за управление на бизнес процеси. Експерименти със софтуер за разработване на многоагентни системи JADE и WADE. Предстваяне на собствен проект за автоматизиране на бизнес процеси посредством Google приложнения. Тенденциите за разпространението на многоагентните системи и ползата от тях при управлението на бизнес процеси .

Ключови думи: агенти,автоматизиране, бизнес процеси, многоагентни системи, JADE, Java,WADE, управление

СЪДЪРЖАНИЕ

Увод

В тази курсова работа ще бъде разгледан проблемът за качественото и продуктивно управление на бизнес процесите, като неразделна част от успешното реализиране на всяка една компания, предприятие или администрация, и в частност интегрирането на многоагентни системи за управление на тези процеси. Темата е избрана на база наблюдения в реална работна среда и осъзнаване на нуждата от минимизиране на човешкият фактор (респективно възможността за грешка) при изпълнението на рутинни и ясно дефинирани работни процеси. Разработването на темата ще представи същността на многоагентните системи и бизнес процесите, както и съществуващите системи за управлението им. Ще бъдат представени предимствата и недостатъците на системите за управление на бизнес процеси, както и причините за въвеждането на много- агентните системи при управлението на бизнес процеси. Възможностите на многоагентните системи ще бъдат представени графично, изразявайки експериметни със програмните продукти JADE (Java agent development framework) и WADE (Workflow and agents development framework). Целта на курсовата работа е да покаже предимствата на многоагентните системи, при управлението на бизнес процеси, техните ограничения и изисквания за имплементиране, както и дългосрочните възможности за надграждане и усъвършенстване.

I. ГЛАВА ПЪРВА- СИСТЕМИ ЗА УПРАВЛЕНИЕ НА БИЗНЕС ПРОЦЕСИ

Системите за управление на бизнес процеси (СУБП) се използват за контролиране, анализиране и управление на бизнес процеси в организациите. Употребата им помага за намаляването на административните усилия и позволява фокусирането върху процеси, които носят печалба на организацциията. Модерната СУБП трябва да предоставя инструменти за дефиниране и изпълнение на бизнес процеси, измервателни инструменти за качеството не изпълнението, мениджърски инструменти, както и да поддържа динамични промени в процесите. Една такава система тряба също да притежава набор от инструменти, които да позволяват интегрирането към други СУБП, посредством различни протоколи, както и да поддържа бизнес процеси между различни организации. В допълнение, модерните СУБП трябва да могат да се адаптират към промени в средата.

Най- често срещаната архитектура за имплементиране на СУБП е тази,

която е ориентирана към Работни процеси (Workflows). Тази архитектура се фокусира върху самият работен процес. Системи от този тип се наричат още и централизирани, защото те имат един процес, който осигурява извършването на целият бизнес процес. Работният процес (Workflow) представлява автоматизиране на конкретен бизнес процес. Типичният работен процес представлява съвкупност от следните елементи:

- Съобщения – осигуряват комиуникацията между служителите (имейли, файлове, хартиени документи и др.)
- Дейности – задачи които трябва да бъдат изпълнени след получаване на съобщение от системата.
- Бизнес правила – описват логиката на бизнес процесите
- Диаграми (Flowcharts) – задават начина, по който протичат процесите в организацията.

Изграждащите блокове на подобен тип система са дейностите (activities). Те се използват за създаване на диаграми и за дефиниране на начина на протичане на работата в организацията. Дейностите могат да бъдат прости, като дейности – задачи , или сложни, като свързване с външнни уеб услуги.

I.1. Предимства на Ориентираните към работни процеси СУБП

Ориентираните към работни процеси СУБП са ясни и прости за имплементиране, и позволяват на организациите да извлекат следните ползи от тях:

- Системата помага за дефиниране и формализиране на работните процеси. По този начин на всички е ясно какво се случва в организацията и каква роля изпълнява всеки индивид.
- Процесите се оптимизират и анализират по-лесно ако са добре дефинирани.
- Сложните бизнес процеси се разбиват на по-малки и по-прости части. Възможно е тази процеси да се анализират и подобрят част по част.
- Интегрира различни приложения с цел да подсигури единичният бизнес процес.
- Осигурява лично работно място на всеки служител.
- Системата разделя бизнес логиката на процеса от реалните работни елементи. Целта на разделянето е всеки работник да се фокусира върху собствените си задачи. Не е необходимо всеки работник да познава в детайли всеки един работен процес.

Изброените предимства са явни и благодарение на тях СУБП стават все по популярни.

I.2. Недостатъци на СУБП

Въпреки предимствата които претижават тези системи, при по-близък поглед върху функционалностите им става ясно че те имат и своите недостатъци. При по сложните бизнес процеси, които съществуват в организациите, те разкриват проблеми при изпълнението на определени сценарии с които не могат да се справят толкова добре. Тези проблеми са следните:

- Този тип системи са базирани в един централен сървър за работни процеси (workflow server). Тази архитектура може да бъде неподходяща при компании, които имат множество офиси в различни региони и всеки регион управлява свои собствени бизнес процеси.
- Недостатъчна автоматизация: Системата описва логиката на бизнес процеса. Логиката представлява последователност от стъпки, които трябва да бъдат извършени, за да се постигне целта.

Проблемът е, че всички задачи се извършват от служители.

- Системата не е достатъчно гъвкава. Необходимо е целият процес с всички региони да бъде ясно дефиниран.
- Липса на управление на ресурсите (resource management). Системата не контролира това дали всички ресурси необходими за изпълнението са налични. Тя предполага, че това е взето под внимание при планирането на процеса. Това прави системата недостатъчно адаптивна.
- Системата на притежава познание за семантиката на процеса. Това се изразява в липсата на информация за самият бизнес процес. Тази информация не може да бъде използвана да се вземат адекватни решения спрямо ситуацията.
- Липса на добре дефинирани протоколи за размяна на информация между системи и интегрирането на различни СУБП за поддръжка на междуфирмени работни процеси.
- Липса на обработка на грешки: грешка при една от стъпките води до грешка в целият процес. Разработчиците трябва да предвидят бъдещи грешки и да имплементират обработчици на грешките (Error Handlers).
- Потребителите не могат да настроят изпълнението на бизнес процеса, по този начин при нестандартна ситуация потребителя не може да изпълни определено действие, защото то не е било

уточнено по време на дефинирането на процеса.

- Сложен процес на версионализиране и ъпгрейд.

Изброените недостатъци са явни и представляват следствие от спецификите на ориентираните към работни процеси и дизайна на централизираните СУБП. Тъй като инжинерите, както и крайните потребители в лицето на организациите се нуждаят от гъвкавост при работата на СУПБ, съществуват и други подходи за изграждането им, които се прилагат. С оглед на недостатъциите на този тип системи, може да се заключи, че проблема за подобряване на ефективността остава, тъй като системите се използват от потребители (работници). Липсата на адаптивност от гледна точка на софтуерно изпълнение и централизирано управление е другата причина, която предполага употребата на по гъвкав подход при управлението на бизнес процесите. Решението на този проблем е свързано с добаввяне на възможност за реакция при промени в средата и редуцирането на участието на потребители в лицето на хора, с цел намаляване на ефекта на човешкият фактор при изпълнението на процесите.

Добавянето на агенти в управлението на бизнес процеси е следващата стъпка в автоматизирането и повишаването на ефективността и

производителността на процесите. Възможността на интелигентните агнти да изпълняват действия и задачи автономно (без нуждата от човешка намеса) в различни случаи и при ясно дефинирани условия ги прави чудесни заместители на потребителите в някои конкретни процеси. Възможността за адаптивност и за употреба на собствена семантика при дефинирането на взаимо-връзките и същността на задачите разширява възможността им за употреба при по-голям брой процеси и дейности. В следващата глава ще бъде разгледана същността на многоагентните системи, както и софтуерите за разработване на такъв тип системи JADE и WADE.

II. СЪЩНОСТ НА МНОГОАГЕНТНИТЕ СИСТЕМИ

Една многоагентна система представлява набор от софтуерни агенти които взаимодействат за разрешаването на определени проблеми, които са извън обхвата на индивидуалните функции или познание на всеки индивидуален агент. „Платформа“ се нарича всичко, което позволява на тези агенти да взаимодействат, без да се взима под внимание вида на самата платформа (централизирана или не, вградена в агентите или не и т.н). Тази платформа обикновено осигурява на агентите набор от услуги в зависимост от нуждите на системата и тя е считана за инструмент на

агентите. Платформата не проявява автономно или проактивно поведение. Според дефиницията за многоагентни системи агентите притежават следните свойства:

- Автономност: Агентът притежава индивидуални цели, ресурси и компетенции. Като такъв той оперира без директна човешка или друга намеса и до някаква степен разполага с контрол върху действията си и вътрешното си състояние. Едно от най-важните последствия от автономността на агентите е тяхната адаптивност. Тъй като агентът има контрол върху собственото си състояние, той може да регулира собственото си функциониране без нуждата от външна намеса или наблюдение.
- Социалност: Агент може да взаимодейства с други агенти, дори и с хора, посредством някакъв вид език за между-агентна комуникация. Посредством това агентът е способен да предоставя и да моли други агенти за определени услуги.
- Реактивност: Агентът възприема и действа, до известна степен, в собствената си затворена среда. Той може да отговори за приемливо време на промени, които възникват в средата.
- Проактивност. Въпреки че някои агенти, наречени реактивни агенти, просто ще реагират на симулация от тяхната среда, агентът може да е способен да прояви поведение което е контролирано от

целта , посредством поемане на инициативата.

Поради нуждата многоагентните системи да бъдат готови за имплементиране в организациите и в бизнеса е било необходимо процеса на изграждане на системите да бъде регулиран, за да могат те да бъдат разбираеми за всички видиове апликации и среди. Следвайки тази цел неправителствената организация FIPA (Foundation for Intelligent Physical Agents) предлага набор от норми и стандарти, които дизайнерите на многоагентни системи трябва да спазват за да могат техните системи да бъдат съвместими с други такива. Според тези стандарти агентите трябва да притежават следните характеристики и свойства:

- Един агент трябва да може да комуникира с всеки друг агент.
- Един агент трябва да осигурява набор от услуги (services), които да са на разположение на всеки друг агент в системата.
- Всеки агент носи отговорност за това да ограничава своята достъпност от други агенти.
- Всеки агент носи отговорност за това да дефинира своите отношения , договори и др. с други агенти. По този начин агента „знае“ директно кой е наборът от агенти, с които може да

взаимодействия.

- Всеки агент съдържа в името си пътят, по който той може да бъде достъпен отвън. Следователно агентите би трябвало да бъдат автономни и да няма поставени ограничения в начина, по който те взаимодействат.

2.1. Организационно- центрирани многоагентни системи

Дефинициите за организация са много и всяка една от тях е насочена към определен аспект на дадено изследване. Едно от определенията предложено от Дженингс и Улдрич, целящо практичност гласи следното:

„ Организацията е сбор от роли, които участват в определени взаимоотношения едни с други, като по този начин участват в систематично институционализирани пътища на взаимодействие с други роли. „

В аспекта на тази дефиниция, както и на други определения на понятието може да се каже че основните свойства на организациите са:

- Организацията е съставена от агенти (индивиди), които проявяват поведение.
- Като цяло организацията може да се разбие на отделни части, които могат да се припрокриват.
- Поведенията на аегнтите са функционално свързани с общата организационна активност.
- Видовете поведение са свързани посредством взаимодействия между ролите, задачи и протоколи.

Ролята описва ограниченията (задължения, изисквания, способности), които един агент трябва да притежава за да изпълнява роля, ползите (авторизации, печалби) които агента ще получи изпълнявайки тази роля и отговорностите свързани с тази роля.

2.2. Агентно базирано управление на бизнес процеси

Вече споменатият модел на СУБП ориентирана към работните процеси е успешен и доходоносен, но е ограничен до употреба само в случаите на управление на по-прости бизнес процеси. Наричан още централизиран, моделът изисква пълно и подробно описание на бизнес процесите от централизирана гледна точка. При толкова подробна информация и при наличието на множество дефиниции, системата има единствено простата задача да изпълнява процесите. За да могат да бъдат превъзмогнати недостатъците на споменатият вид СУБП, е необходимо да се използва изцяло различен подход. Предложеният подход за въвеждането на агенти при управлението на бизнес процеси, има за цел да прехвърли отговорността за изпълнението на определени дейности свързани с бизнес процесите на единиците, които ги изпълняват, а не тя да бъде централизирано поддържана и управлявана. По този начин всяка една от основните дейности на бизнес процеса е възложена на определена единица, която е програмирана да реши определен проблем.

Основните предимства на агентно- базираните СУБП пред ориентираните към работни процеси са следните:

- Осигурява по-голяма гъвкавост, тъй като действията могат да се базират на настоящата ситуация в която се намира агента,

отколкото да бъдат предварително дефинирани.

- Осигурява по-голяма скорост при изпълнението, тъй като новите услуги могат да бъдат добавяни и конфигурирани , така че да повлияят минимално на други агенти.
- Този подход предлага по-голяма адаптивност поради факта, че изборите на агентите могат да бъдат напътствани от налична обратна връзка, получена при предишни повиквания на опредлени части от бизнес процеса.

За използването на агентите в управлението на бизнес процесите съществуват два сценария на изпълнение:

2.2.1. Агентно поддържана СУБП.

При този вид използване на агенти в системите за управление, агента поддържа изпълнението на процесите работейки заедно със самата СУБП. Типичната роли на агентите при това изпълнение са:

- Потребителски интерфейс: Агентът помага на потребителя да

изпълнява задачи, зададени по време на изпълнението на определен бизнес процес. Помощта се изразява във филтриране на имейли, автоматично отговаряне на имейли, напомняне за предстоящи събития и др.

- Автономна обработка на работни обекти: Агентът изпълнява задачи без външна намеса.
- Интерфейс до външни системи: Агентът подсигурява комуникацията между СУПБ и външни приложения.Наличието на този интерфейс позволява на агентите да се свързват с други СУБП да достъпват информация и да извършват определени действия.

2.2.2. Агентно управлявана СУБП

При този вид имплементиране на агенти, самите агенти контролират протичането на процеса. Всеки агент представлява определена част от бизнес процес. Съдействието между агентите подсигурява целият процес. За да се постигне това, целият процес трябва да бъде разделен

на отделни функции и операции. След това във всяка функция и операция се имплементира интелигентен агент . По този начин всеки агент е отговорен за един или множество стъпки от работният процес и комуникирайки с другите агенти решава коя трябва да бъде следващата изпълнена стъпка.

3. ЕКСПЕРИМЕНТИ СЪС СОФТУЕР ЗА РАЗРАБОТВАНЕ НА МНОГОАГЕНТНИ СИСТЕМИ JADE и WADE

3.1. Същност на JADE

JADE представлява софтуер, който служи като мост между операционна система и бази от данни или апликации. Той улеснява разработката на многоагентни системи. Използва се за създаване и терминиране на агенти, комуникацията между тях, както и за мониторинг над процесите между различните агенти. JADE съдържа следните компоненти:

- Среда за изпълнение (Runtime environment) – това е средата в която агентите „живеят“. Тази среда трябва да е активна в определен хост преди един или повече агени да могат да бъдат стартирани от този хост.
- Библиотека (library) от класове , които работещите с JADE могат/трябва да използват за разработване на агенти
- Набор от графични инструменти които позволяват администриране и наблюдение над работещите агенти.

3.1.1.Употреба на Онтологии

За да могат да комуникират помежду си, агентите трябва да предават помежду си съобщения, които имат смисъл за тях. JADE подържа голямо ниво на стандартизация в създаването на събощенията от агентите, според изискванията на FIPA. Вградените в софтуера съобщения са полезни, когато става въпрос за предаване на атомен вид данни/информация. В случите в които съобщенията трябва да представляват абсрактни понятия, обекти или структурирани данни вградените в JADE съобщения са недостатъчни. В такива случаи се използват онтологии. Дефиниране на онтология представлява дефинирането на собственa лексика и семантика за съдържанието на

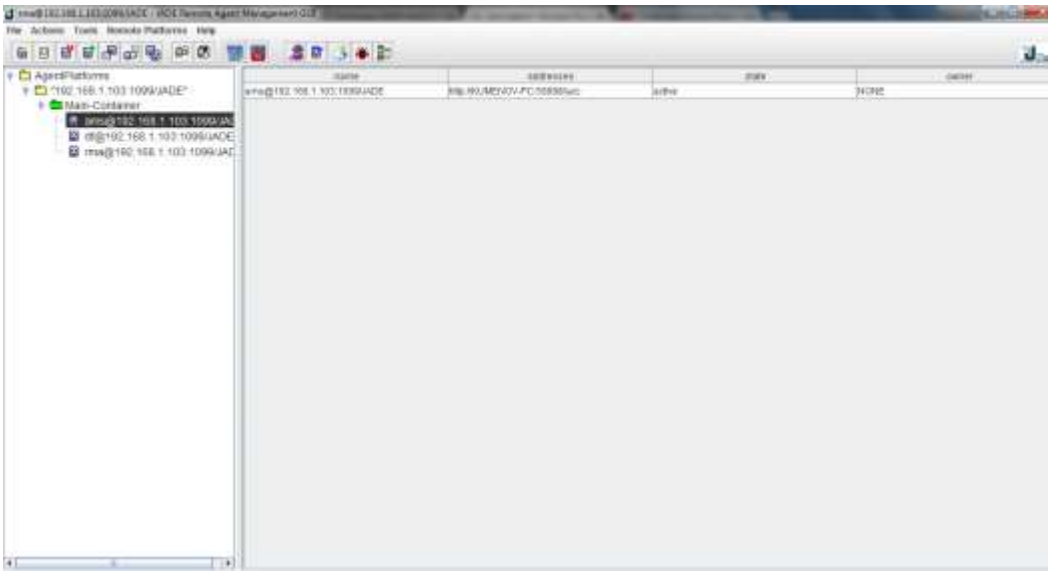
съобщенията, които се разменят между агентите.

3.1.2. Архитектура на JADE

Агентите представляват компоненти на JADE, които могат да изпълняват задачи и да комуникират с други агенти посредством предаване на съобщения. Агентите „живеят“ във платформа, която им предоставя основни функции, като доставяне на събощения. Всяка платформа е съставена от един или повече контейнери. Контейнерите могат да бъдат изпълнявани на един или повече хостве, като по този начин те достигат до създадена платформа. Всеки контейнер може да съдържа нула или повече агенти. Във всяка платформа съществува специален контейнер наречен Главен контейнер (Main container), който сам по себе си е контейнер но се различава от другите по следните показатели:

- Главният контейнер винаги е първият стартиран контейнер в платформата и всички останали контейнери се регистрират към него при стартиране
- Той съдържа в себе си два специални агента:

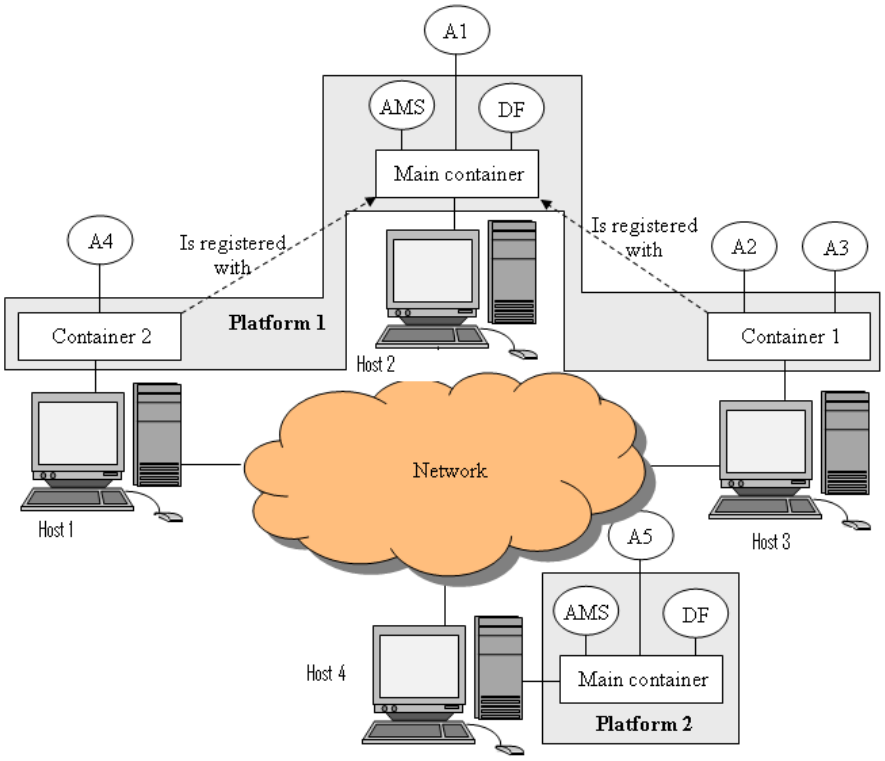
а) AMS (Agent management system) агент – това е агента представляващ пълномощията в платформата. Той е единственият агент, който може да активира мениджърски действия в платформата като: създаване/терминиране на други агенти, терминиране на контейнери. Всички останали агенти, които желаят да предприемат подобни действия, трябва да отправят заявки за тях до AMS агента. Този агент действа и ограничен в границите на платформата, за която отговаря и в която съществува. В една платформа съществува само един AMS агент. AMS и DF агентите се стартиират автоматично при стартиране на конзолата на JADE.Двата агента са показани на фигура



Фиг 1.

б) DF агент (Directory facilitator) – този агент въвежда „ услугата на жълтите страници“ в смисъла на това кои агенти могат да обявяват своите услуги и намира други агенти предлагащи услугите които са им необходими.

На Фигура 2. е показана условно архитектурата на JADE.



Фиг. 2

3.1.3.Комуникация между агенти

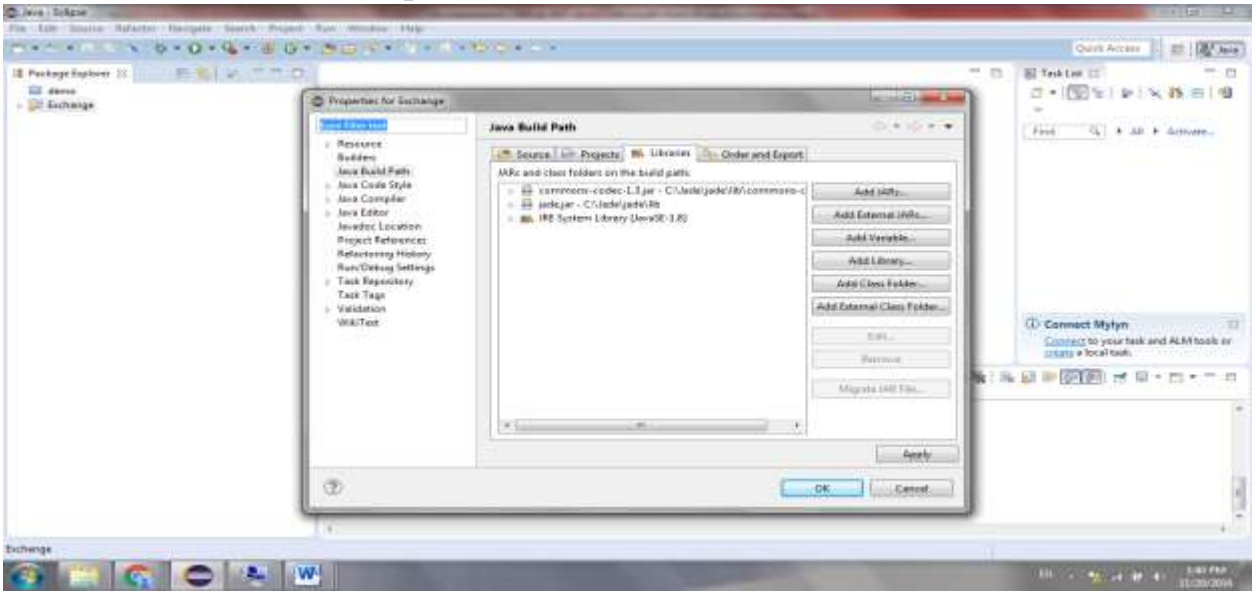
Агентите могат да комуникират прозрачно независимо от това дали живеят в един и същ контейнер, в различен контейнер (намиращ се в един и същ или различен хост), принадлежат към една или различни платформи. Комункацията е базирана на асинхронна парадигма за предаване на съобщения. Формата на съобщенията е дефиниран от ACL (Agent communication language) език. Съобщенията съдържат набор от полета включващи следното:

- Изпращач
- Получател/и
- Комуникативен акт – това представлява намерението на изпращача на съобщението. Например ако изпращача изпраща информиращо съобщение, се очаква получателя да бъде осведомен относно някакъв факт. Ако изпращача изпраща съобщение заявка, се очаква получателя да предприеме някакво действие.
- Съдържание- същинската информация предадена в съобщението.

3.2. Стартиране на агенти в JADE под Eclipse

В настоящият експеримент ще бъдат стартирани два агента (Alice и John), които ще бъдат конфигурирани в Eclipse. Агентите в JADE са дефиринирани, като подкласове на предефиринираният клас **Agent**.

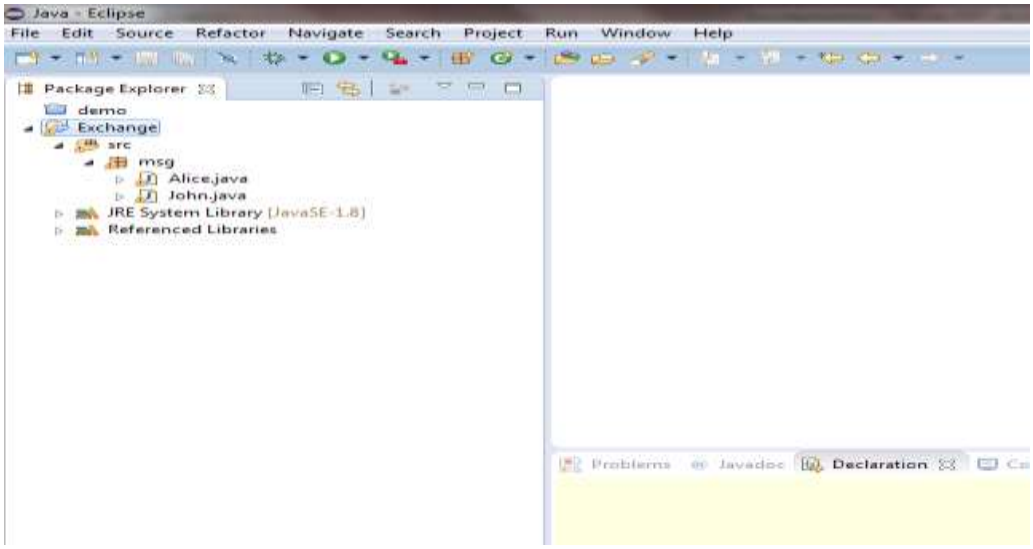
- За целта на експеримента е създаден нов проект в Eclipse с име Exchange. За да могат да се използват ресурсите на JADE, както и за да бъде стартирана управленската конзола е необходимо JAR файловете да бъдат добавени към библиотеките



Фиг. 2

Фигура 4.

За да бъдат създадени агентите е необходимо да се създат два нови класа с имена Alice и John под пакета msg. Създадените класове са показани на Фигура 5.



Фиг. 5

За разлика от създаването на агенти посредством потребителската конзола, при конфигурирането им, чрез Java код, е необходимо да се дефинират всички спецификации, модели на поведение, променливи, действия и задачи, които ще изпълнява съответният агент или група

от агенти.

Конфигурирането на първият агент alice, определя агента да изпълнява едно действие (изпращане на съобщение) еднократно. Задачата на агента е при стартирането на платформата да изпрати съобщение до друг агент (този който ще бъде стартиран от другият клас John). Кода на alice изглежда по следният начин в Eclipse:

```
-   package msg;
-
-   import jade.core.AID;
-   import jade.core.Agent;
-   import jade.core.behaviours.OneShotBehaviour;
-   import jade.lang.acl.ACLMessage;
-
-   public class Alice extends Agent{
-
-
-
-          protected void Setup () {
-                 addBehaviour(new OneShotBehaviour() {
-
-
-          public void action () {
-                 ACLMessage msg=new ACLMessage(ACLMessage.INFORM);
-             msg.setContent("Hello John, My Name is Alice");
-             msg.addReceiver(new AID("john",AID.ISLOCALNAME) );
-           send(msg);
-          }
-
-                 });
-   }
-          }
```

В началото на кода е дефиниран клас, който разширява съществуващият jade.core. Agent и е част от стартирането на всеки един Jade агент. След него е дефиниран клас, разширяващ съществуващият клас за видовете “поведения” на агентите. Към него е добавен типът поведение “OneShotBehaviour”, който представлява: извършване на еднократно действие при стартиране на агента. Избран е OneShotBehaviour, защото за целта на експеримента е необходимо агента да изпрати съобщение до другият агент еднократно, след стартирането си. След дефинирането на вида поведение, с цел агента да може да изпраща/получва съобщение е необходимо да се дефинира клас, разширяващ съществуващият jade.lang.acl.ACLMessage, който се отнася за изпращане на събощение според дефинициите на ACL (Agent Couminication Languague) езика. Спазването на изискванията за съобщение дефинирани в ACL е задължително, за да могат агентите да комуникират помежду си. След дефинирането на класовете е дефиниран агентът alice, който разширява съществуващия клас Agent. Използвайки метода Setup () е дефинирано поведението OneShot , което агента ще изпълнява. Действието на агента alice в смисъсла на участващ в комункацията е на изпращач и е дефинирано с метода action () , посредством който са описани характеристиките на съобщението:

- Ново съобщение от вида INFORM
- Съдържание на съобщението
- Получател на съобщението.

Съобщението, което агентът трябва да изпрати при стартиране на платформата е информативно (Inform) и съдържа текста „Hello John My name is Alice”. Получателя на съобщението е дефиниран посредством разширеният клас jade.core.AID, който представлява идентификатор на агенти. Този клас записва и съхранява имената и адресите на агентите. В настощят експеримент дефинираният получател е агент john и при стартиране на агент alice, той ще адресира и изпрати съощение само до агент с това име. Метода action () завършва с команда за изпращане на съобщение. Командата е поставена на последно място в кода описващ действието на агента, тъй като преди да пристъпи по изпращане на съобщение, агента alice трябва да изпълни действията по подготовка и адресирне на съобщението.

- След конфигурирането на изпращача на съобщението alice , е необходимо да конфигурираме агента john, който ще бъде получателят на съобщението изпратено от alice. За

да бъде визуализиран процеса по изпращане и получаване на съобшение в кода на john ще бъде въведен метод за изобразяване на диалогов прозорец след получаване на съобщение.

```
-   package msg;
-
-   import javax.swing.JOptionPane;
-
-   import jade.core.AID;
-   import jade.core.Agent;
-   import jade.core.behaviours.CyclicBehaviour;
-   import jade.lang.acl.ACLMessage;
-
-   public class John extends Agent {
-
-          public void setup () {
-                 addBehaviour(new CyclicBehaviour() {
-
-             public void action () {
-                                ACLMessage msg = receive();
-
-                                      if (msg != null){
-                                         JOptionPane.showMessageDialog(null,
    "Message received" +
-                                        msg.getContent());
-                                         {ACLMessage arg=new
    ACLMessage(ACLMessage.INFORM);
-                                            arg.setContent("Hallo I Received
    your Massage");
-                                            arg.addReceiver(msg.getSender() );
-                                        send(arg);
-                                        } }
-                                        else block ();
-                               }
-
-                    });
-                 }
-   }
```

- В началото на кода на агент john е добавен разширеният клас javax.swing.JOptionPane, който се използва за създаване на диалогови прозорци (информативни, или изискващи въвеждане на данни). Следващите класове дефинирани в началото на кода са идентични с тези, които бяха дефинирани при alice:

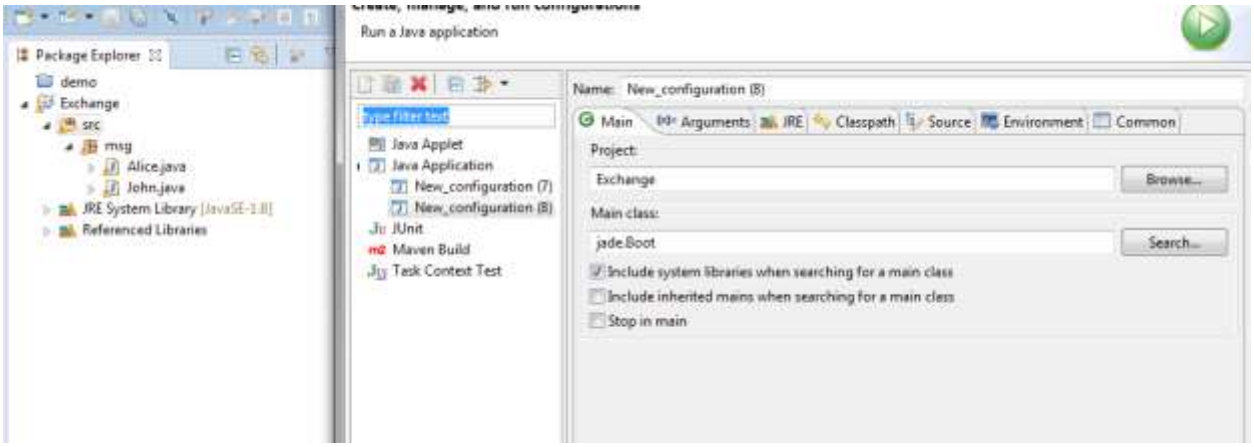
```
- import jade.core.Agent;
- import jade.core.behaviours.CyclicBehaviour;
- import jade.lang.acl.ACLMessage;
```

Видът поведение (Behaviour) на агент john, който е избран е CyclicBehavaiour. Това поведение отговаря на повтарящ се цикъл, при който се търси среща с определен критерий за извършване на действие. Цикълът който е определен в следващите редове на кода е следният:

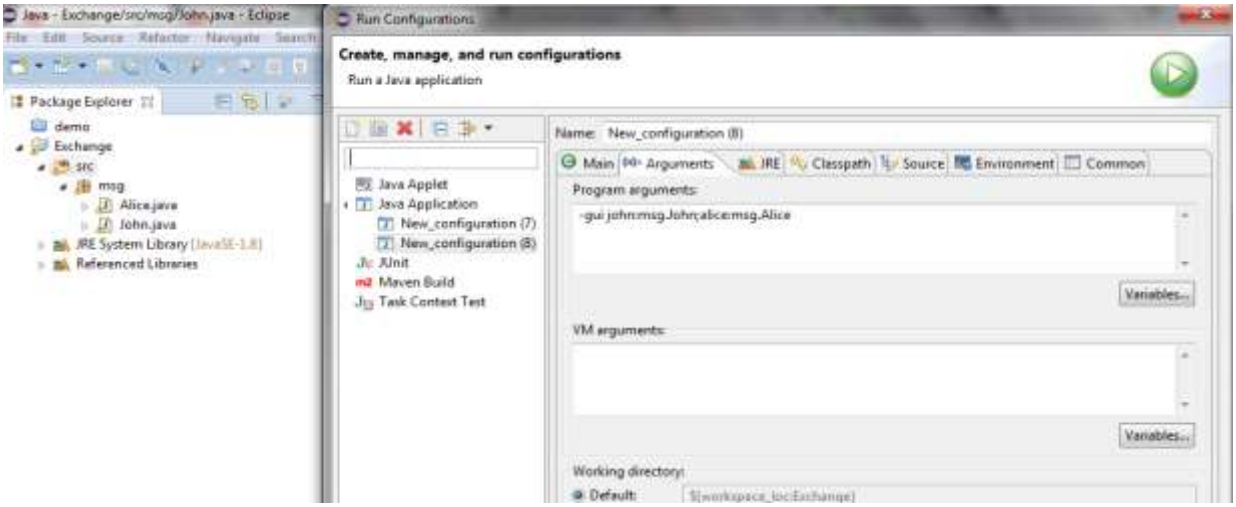
- Получаване на съобщение
- Проверка на съдържанието на съобщението
- Ако съдържанието е различно от „null” (нула), изобразаяне на диалогов прозорец със съдържание: “Message received” + съдържанието на полученото събощение.
- Изпращане на ново съобщение от вида INFORM до изпращача на събощението със съдържание : „Hello, I received your message”.
- Ако полученото от агента john съобщение е “null” ,

агентът прекратява действието и се връща в първоначална позиция, очаквайки ново съобщение.

- При стартирането на агентите посредством Eclipse не е необходимо те да бъдат предварително компилирани. За да се стартират два различни агента, по един за всеки от класовете Alice и John, при конфигурациите за стартиране трябва да се добави, главният клас (jade.Boot), както и подходящите аргументи, които да създадат агентите и да стартират конзолата за управление. Предварителното конфигуриране, както и необходимите за стартирането на агентите и конзолата аргументи са показани на фигури 6 и 7:

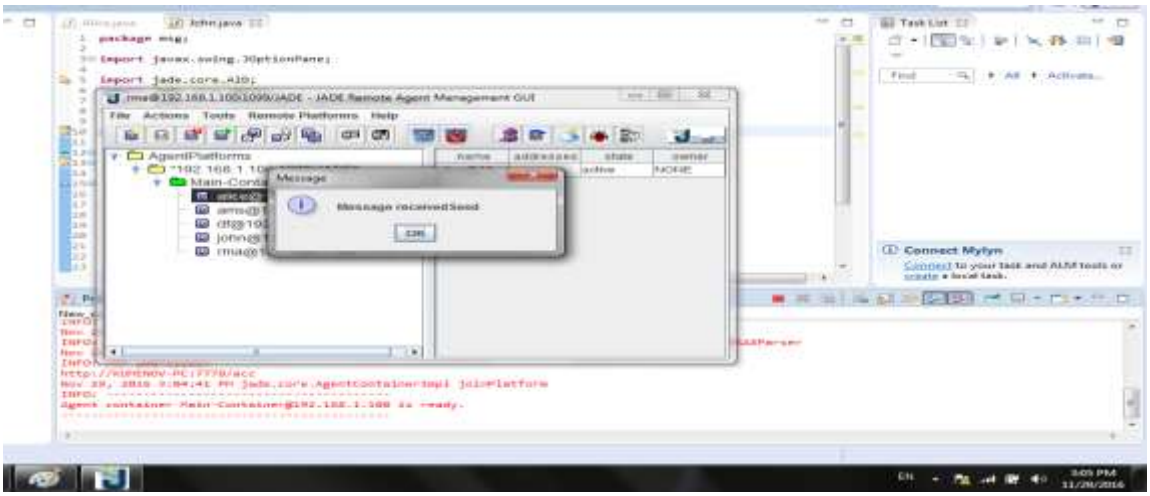

Run a Java application
Package Explorer
demo
Exchange
src
msg
Alice.java
John.java
JRE System Library [JavaSE-1.8]
Referenced Libraries
Java Applet
Java Application
New_configuration (7)
New_configuration (8)
JUnit
Maven Build
Task Context Test
Name: New_configuration (8)
Main
Arguments
JRE
Classpath
Source
Environment
Common
Project:
Exchange
Browse...
Main class:
jade.Boot
Search...
Include system libraries when searching for a main class
Include inherited mains when searching for a main class
Stop in main

Фиг.6.


Java - Exchange/src/msg/John.java - Eclipse
Run Configurations
Create, manage, and run configurations
Run a Java application
Package Explorer
demo
Exchange
src
msg
Alice.java
John.java
JRE System Library [JavaSE-1.8]
Referenced Libraries
Java Applet
Java Application
New_configuration (7)
New_configuration (8)
JUnit
Maven Build
Task Context Test
Name: New_configuration (8)
Main
Arguments
JRE
Classpath
Source
Environment
Common
Program arguments:
-gui john:msg.John;alice:msg.Alice
Variables...
VM arguments:
Variables...
Working directory:
Default:

Фиг.7.

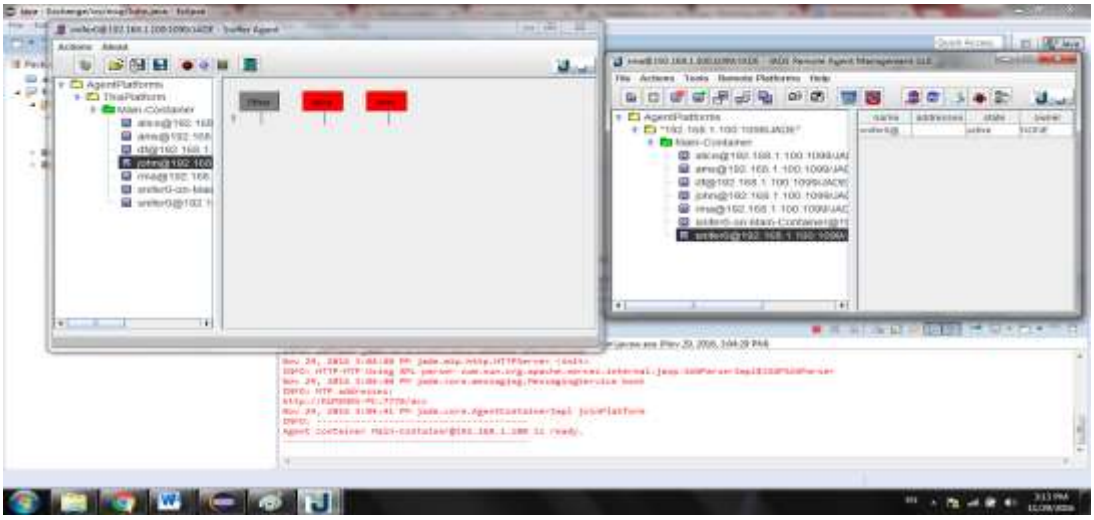
След потвърждаване на конфигурациите и стартиране на агентите, графично се визуализира управленската конзола на JADE. Изпълнението на кода на агентите визуализира диалоговият прозорец, конфигуриран в кода на john, даващ информация за полученото съобщение. Диалоговият прозорец е показан на фигура 8.



Фиг. 8

- Тъй като комуникацията между двата агента, както и съпътстващите я процеси, се извършват автономно след

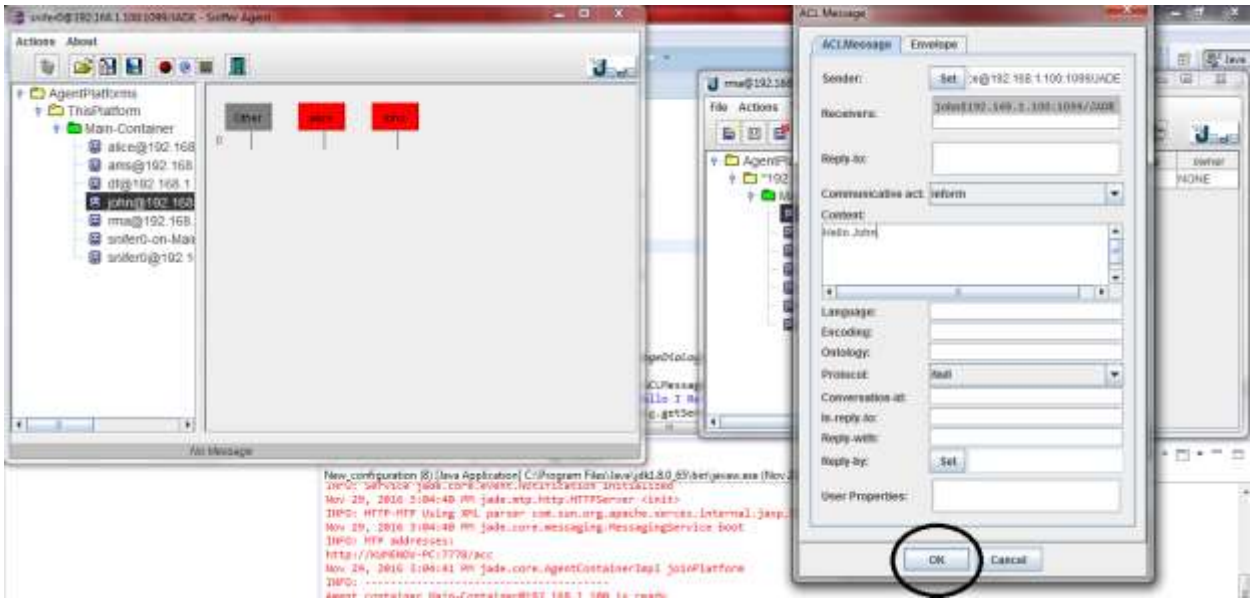
стартиране на проекта в Eclipse, за да бъдат визуализирани графично ще бъде стартураб Snifer агент, който да наблюдава комуникацията между alice и john. Snifer агента инспектира разговорите между агентите. Той е вграден агент в конзолата на Jade. На фигура 7 е показан Snifer агента след стартирането му.



Фиг. 9

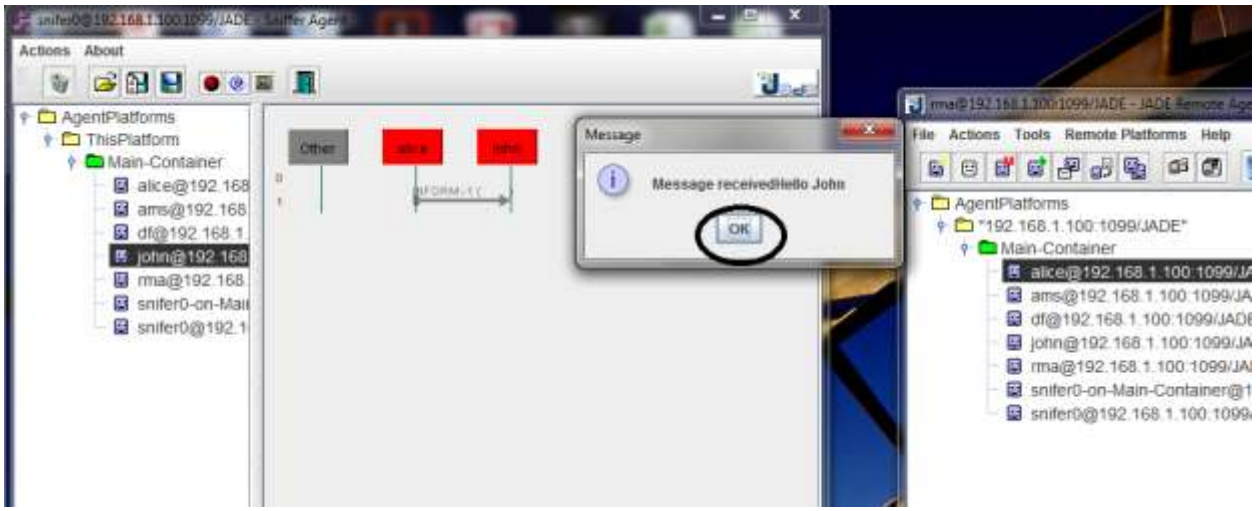
Тъй като проекта вече е стартирана и alice е изпълнила своя код при стартирането си, за да се визуализира комуникацията между двата агента след стартирането на агента snifer0, ще бъде изпратено ново съобщение от alice до john, използвайки вече стартираната конзола на

JADE. Изпращането на съобщение от alice към john е показано на фигура 10.

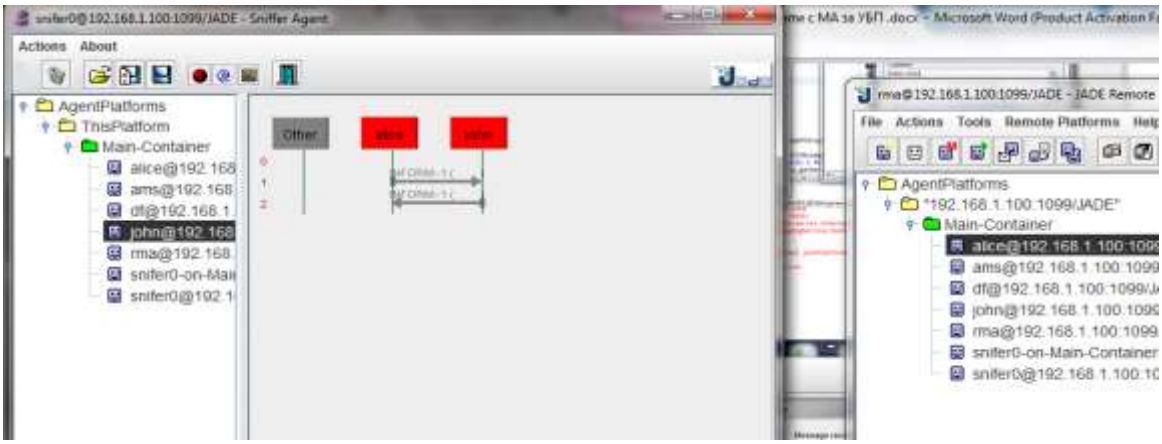


Фигура 10.

След получаване на съобщението , john автоматично изпраща съобщение до alice, при което се отваря нов диалогов прозорец, съдържащ информация за полученото съобщение,както е показано на фигури 10 и 11 .


Actions About
AgentPlatforms
ThisPlatform
Main-Container
Other
Message
Message received!Hello John
OK
File Actions Tools Remote Platforms Help
"192.168.1.100:1099/JADE"

Фиг. 10


Actions About
AgentPlatforms
ThisPlatform
Main-Container
Other
File Actions Tools Remote Platforms

Фиг. 11

3.3. WADE – Workflow Agent Development Environment

WADE е софтуер базиран на JADE, който позволява поддръжка на изпълнението на задачи, в смисъла на термина работен процес. Ключовият компонент на WADE е WorkflowAgentEngine клас, който разширява базовите класове от агенти на JADE и осигурява единен „двигател“ за управление на работни процеси. WADE поддържа „нодпад –програмиране“ в смисъла на това, да не съществуват неща който програмистите и девелопърите да не могат да контролират. Основното предимство на софтуера е това, че позволява изобразяването на процеси в удобен графичен вид. Ключовият елемент в подхода на WADE е WOLF (Workflow Life cycle management system), базираната на Eclipse среда за разработване на WADE-базирани приложения. Използвайки WOLF програмистите са свободни да използват смисъла на термина работен процес (workflow metaphor), когато сметнат това за необходимо и имат възможността лесно да превключват от графично представяне на Java код, в зависимост от това кое сметнат за по-подходящо. Самото име на WOLF подсказва, че това не е само инструмент за графично създаване на работни процеси за WADE, ами е завършена среда за управление на пълният жизнен цикъл на работните процеси: от първичен стадий на прототипи до тестване и крайно пускане в експлоатация.

Конзола за управление

чКонзолата за управление която е достъпна при инсталация на софтуера позволява базови действия, като стартиране и спиране на приложение, експортиране и импортиране на конфигурации, както и разгръщане на нови или редакция на процеси без да се затваря съответната апликация. WADE SUITE предлага и развита уеб базирана администраторска конзола, която предлага много допълнителни опции за управление, създаване и развиване на приложения и процеси.

3.3.1. Архитектура на WADE

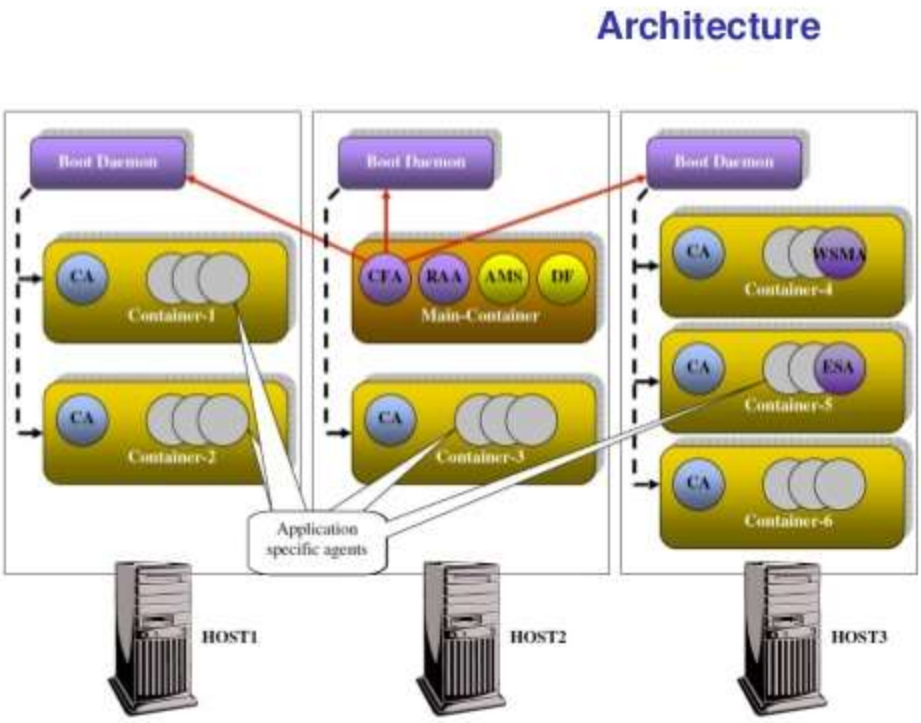
Базираното на WADE приложение е подобно на JADE. WADE работи с няколко хоста, съдържащи един или повече JADE контейнери, и както и при JADE главният контейнер (Main Container), трябва да бъде активиран първи, заедно със всички други контейнери които се регистрират към главният. Специфичните за WADE компоненти са:

- BootDeamon процеси- тези процеси са по един за всеки хост и

всеки от тях е отговорен за активиране на контейнерите в хоста за който се отнася.

- ConfigurationAgent (CFA) – винаги съществува в главният контейнер и взаимодейства с BootDeamon процесите и контролира жизненият цикъл на приложението
- ControllerAgents (по един за всеки контейнер) – отговаря за наблюдението на процесите в локалният контейнер както и за всички механизми за отказоустойчивост предоставяни от WADE.
- Worflow Engine Agents- агентите които могат да изпълняват задачи, дефинирани като работни процеси.

Условно архитектурата на WADE е представена на диаграмата показана на фигура 12.



Фиг. 12

3.3.2. Работни процеси (Workflows)

Тъй като основната цел на WADE е създаването, управлението и интегрирането на работни процеси базирани на JAVA език, в тази точка са описани елементите на така наречения мета-модел на работни

процеси. Елементите които съдържа мета- модела са следните:

- Процес (Process) – това е задача, която е подробно описана
- Всеки процес съдържа набор от дейности (Activities), всеки от които отговаря за изпълнението на дадена операция.
- Всеки процес съдържа един стартова дейност Start Activity дефиниращ входните точки на изпълнението и една крайна дейност- End Activity , която дефинира последните точки от изпълнението на дадена операция.
- За процеси които не съдържат End Activity съдържа едно или повече преубразования Transitions, в повечето случаи свързани със състояние, водещо до друга дейност (Activity) в процеса.

Основните видове дейности (Activities) поддържани от WADE са следните:

- Code Activities кодови дейности- Операциите включени в кодови дейности се дефинират директно със JAVA код и се добавят още при описанието на самият работен процес
- Tool Activities – инструметнтни дейноси – операциите включени в

инструментни дейности се състоят в обръщане към един или повече външни инструменти (дефинирани като Приложения).

- Sub Flow Activities -под процесни дейности – операциите включени в под процесните дейности се състоят в обръщане към друг работен процес.
- Web Service Activity – под процесни дейности, които осигуряват поддръжка при обръщане към уеб сървис при изпълнението на работен процес.
- Wait Event activity: представлява дейност, изчакаваща определено събитие. Когато е активирана, тази дейност блокира изпълнението на процеса, докато не се случи конкретно събитие.
- Wait Web Service activity – представлява синхорнизираща дейност , която при повикване, блокира изпълнението на работният процес, докато събитие, кореспониращо с обръщане към определен Web Service се случи.

4. Управление на WADE – базирани проекти.

При употребата на WOLF в Eclipse програмистите имат възможността да използват свойствата на WADE при разработването на проекта. Основните свойства за управление на проекти които притежава WADE са следните:

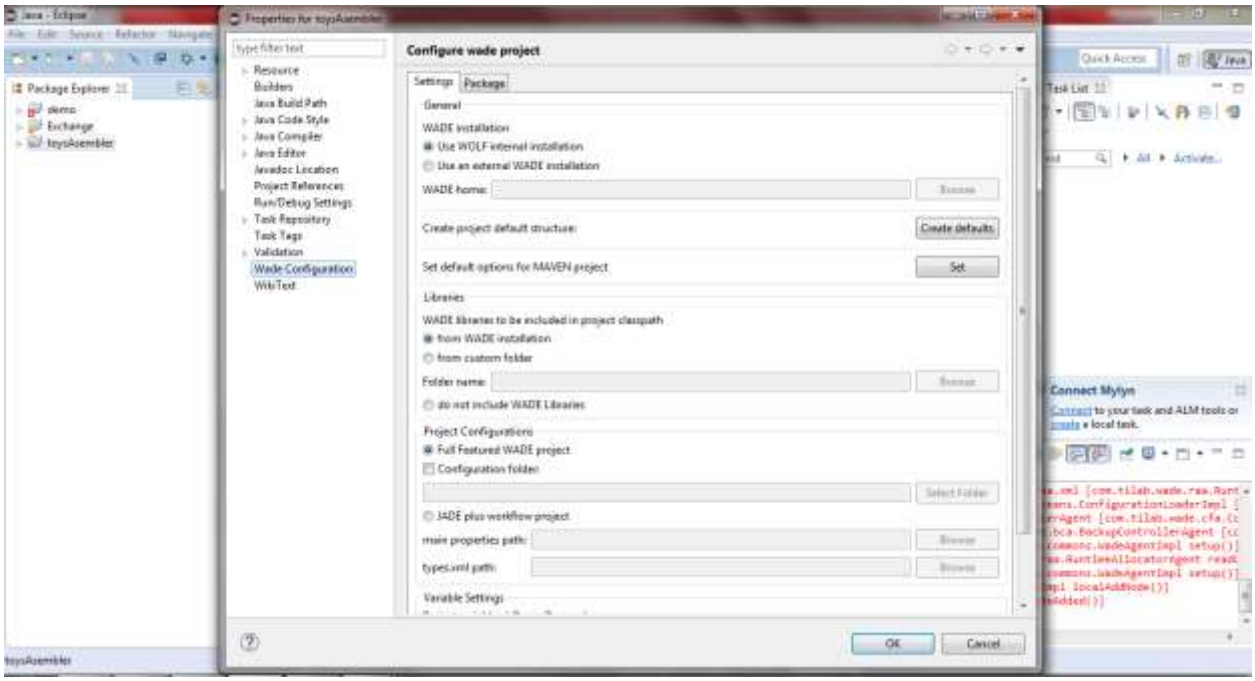
- Редактиране, разгръщане и изпълнение на работни

процеси. WOLF осигурява подкрепа при рекатирането на работни процеси и разгръщане на работни процеси при действаща платформа.

- Контролиране / Активиране на WADE базирани приложения/апликации. За да бъдат тествани новосъздадените работни процеси, WOLF позволява стартиране на вътрешна WADE платформа. В този случай изпълнението на boot deamon и main container се контролира от Eclipse и двете от тях могат да бъдат стартирани в нормален или debug режим.
- Дърво на агентите – WOLF осигурява поддръжка за свързване към вътрешна или външна платформа и за управление на приложенията който са действащи в нея.

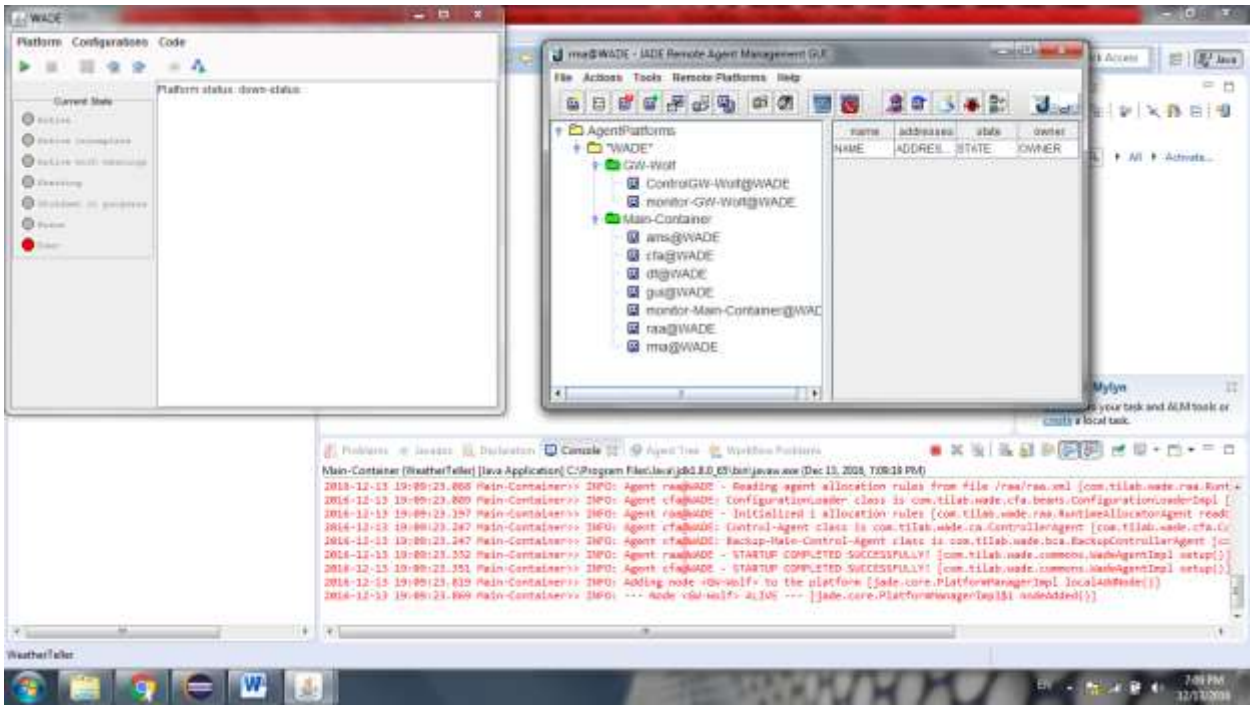
5. Експерименти с WADE

За да създадем нова WADE базирана апликация е необходимо да се създаде нов Java проект в Eclipse. Към проекта се добавя WADE природа (Wade nature), за да може да се създаде главният контейнер и да бъде стартирана администраторската конзола. Преди стартиране на контейнера в менюто Wade configuration се задават спецификациите и структурата на проекта, както е показано на фигура 13.



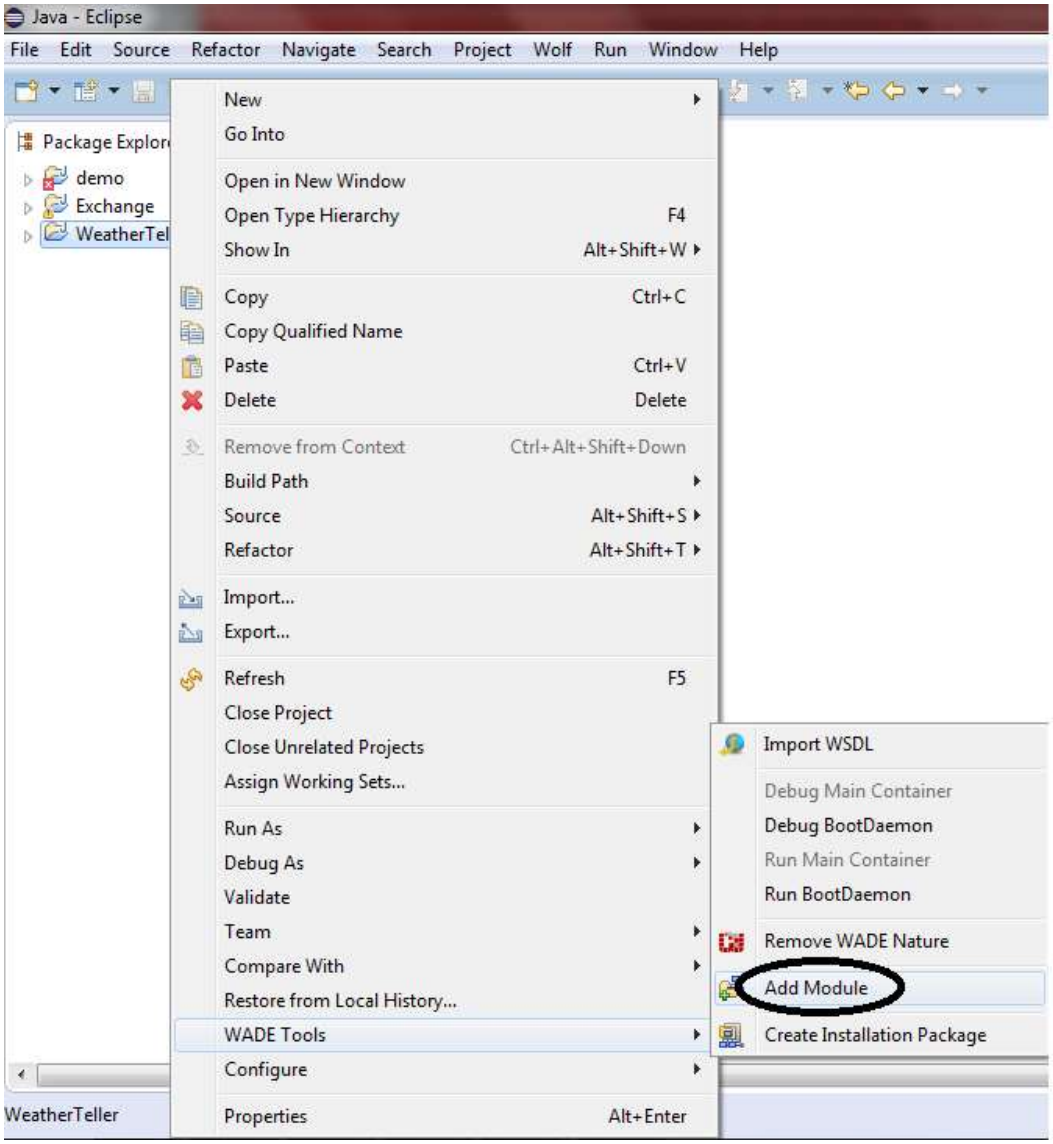
Фиг. 13.

След потвърждаване на промените, в менюто със опции на проекта, който е създаден, се избира Wade Tools Run Main Container. По този начин ако проекта е конфигуриран правилно се стартира администратроската конзола на WADE, както и дървовидният вид на агентите създаден по default. Новосъздадената платформа е показана на фигура 14.



Фиг. 14

За да създадем нов работен процес (workflow), е необходимо да бъде създаден модул, в който този работен процес ще се изпълнява. За целта на експеримента ще бъде създен нов работен процес, който ще изпълнява дейностите по проверка на времето и изпращане на имейли при дъждовно време. Създаването на модула се извършва посредством едно от менютата на менюто Wade Tools в свойствата на новосъздадения проект, както е показано на фигура 15.

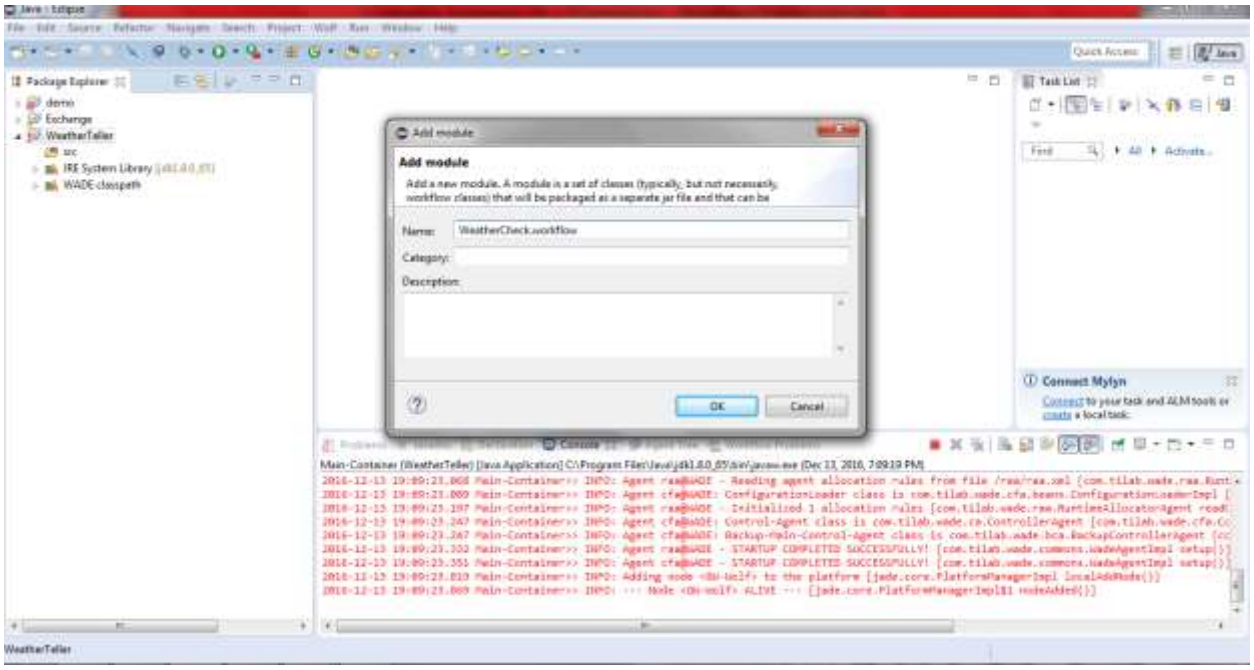

Java - Eclipse
File Edit Source Refactor Navigate Search Project Wolf Run Window Help
Package Explor
demo
Exchange
WeatherTel
New
Go Into
Open in New Window
Open Type Hierarchy F4
Show In Alt+Shift+W
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Delete Delete
Remove from Context Ctrl+Alt+Shift+Down
Build Path
Source Alt+Shift+S
Refactor Alt+Shift+T
Import...
Export...
Refresh F5
Close Project
Close Unrelated Projects
Assign Working Sets...
Run As
Debug As
Validate
Team
Compare With
Restore from Local History...
WADE Tools
Configure
Properties Alt+Enter
Import WSDL
Debug Main Container
Debug BootDaemon
Run Main Container
Run BootDaemon
Remove WADE Nature
Add Module
Create Installation Package
WeatherTeller

Фиг. 15

За целта на експеримента ще бъде създаден проект с име Wether Teller , който има за цел да проверява времето и да съобщава на потребителя, когато на мястото на което живее вали дъжд. Условно приложението има следната структура и жизнен цикъл:

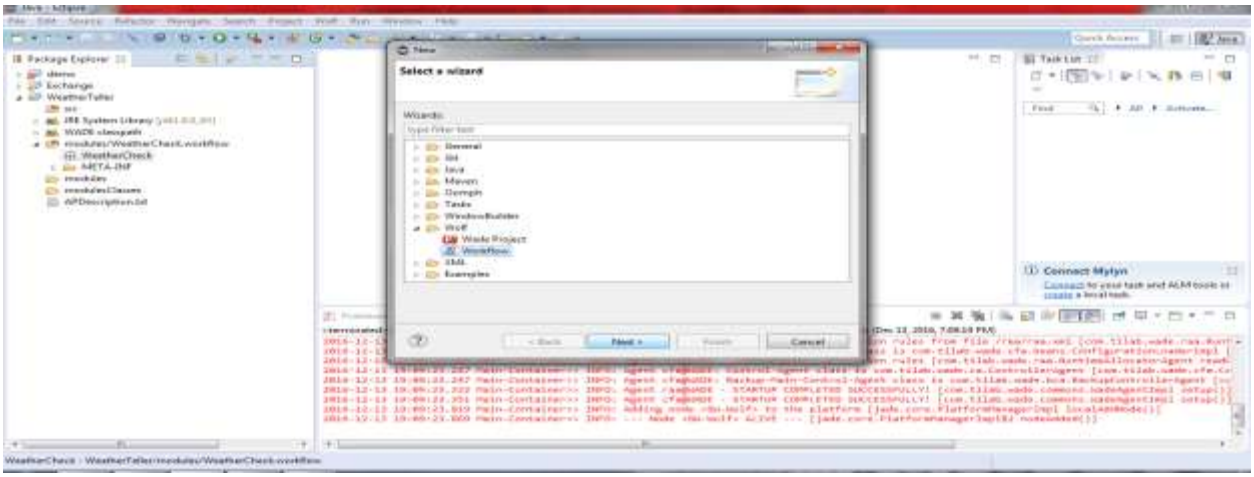
- Проверка на времето в определен час от деня
- Изтегляне на информация относно конкретно състоятние на времето. В случаят наличие на валежи.
- Проверка на условие
- Изпълнение на действия според условието

Името на проекта е WeatherTeller , съответно модула който създадваме ще бъде наименуван WeatherCheck.workflow. Той ще съхранява java файловете на самият работен процес. Създава се нов модул, както е показано на фигура 16.



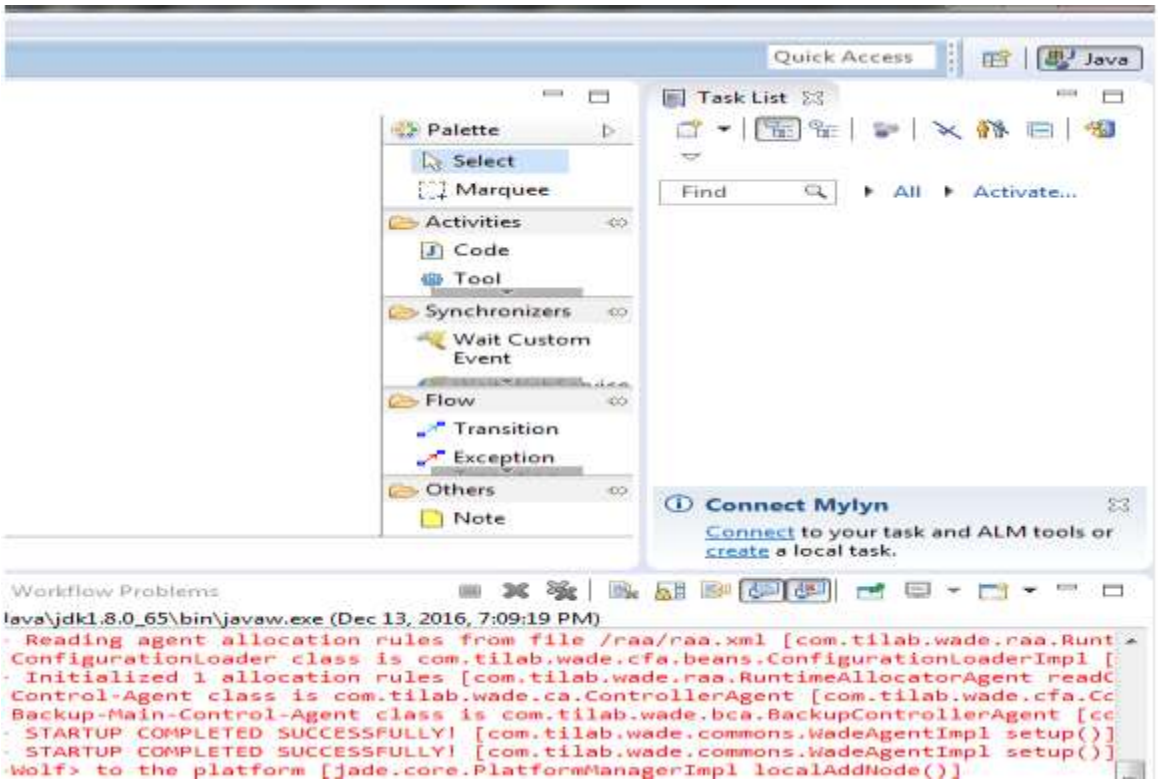
Фиг. 16

За да бъде визуализиран панелът със опции на WADE, е неоходимо да от създаденият пакет WeatherCheck, да изберем следното: от менюто New се избира Other и от опциите на WOLF се избира workflow. Менюто е изобразено на фигура 17.



Фиг. 17

Новосъздаденият workflow е със име weatherTeller и след потвърждаването на създаването му, от дясната страна се появява панел със опциите на Wade за графично изобразяване на работни процеси, както е показано на фигура 18.

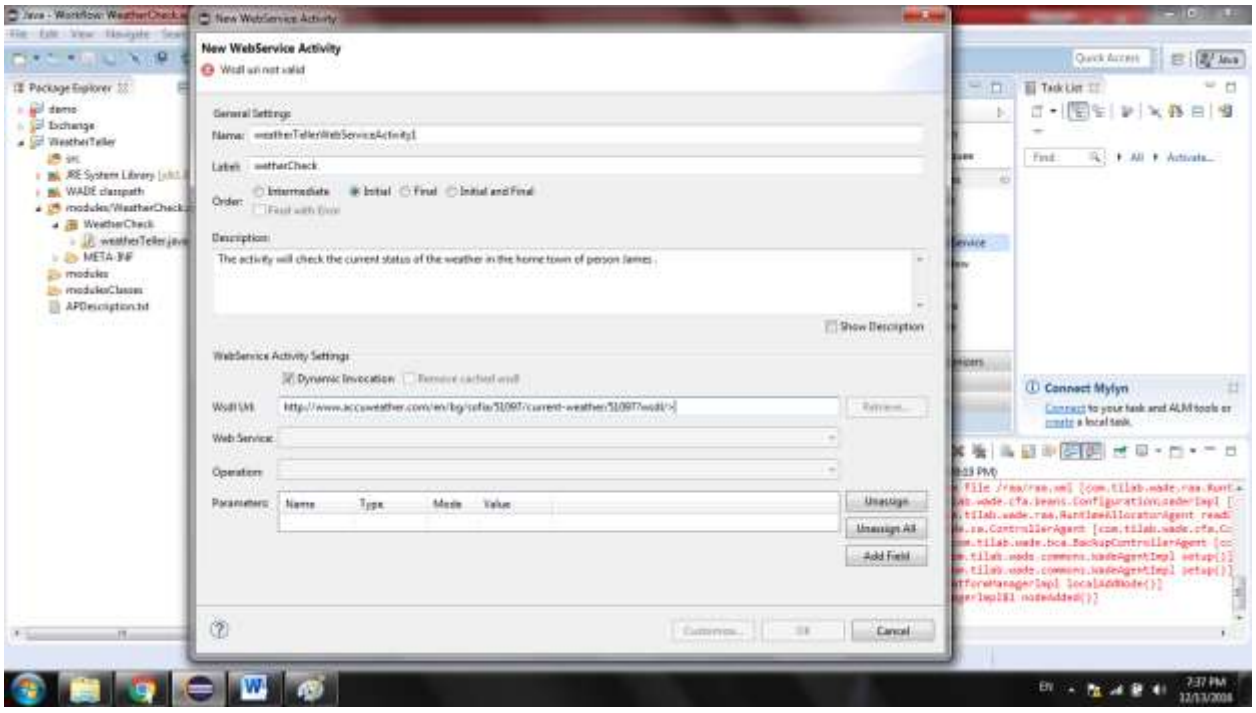


Фиг. 18

В следващите стъпки ще бъде описан и графично създаден процес по проверка на състоянието на времето и автоматично уведомяване при наличието на дъжд.

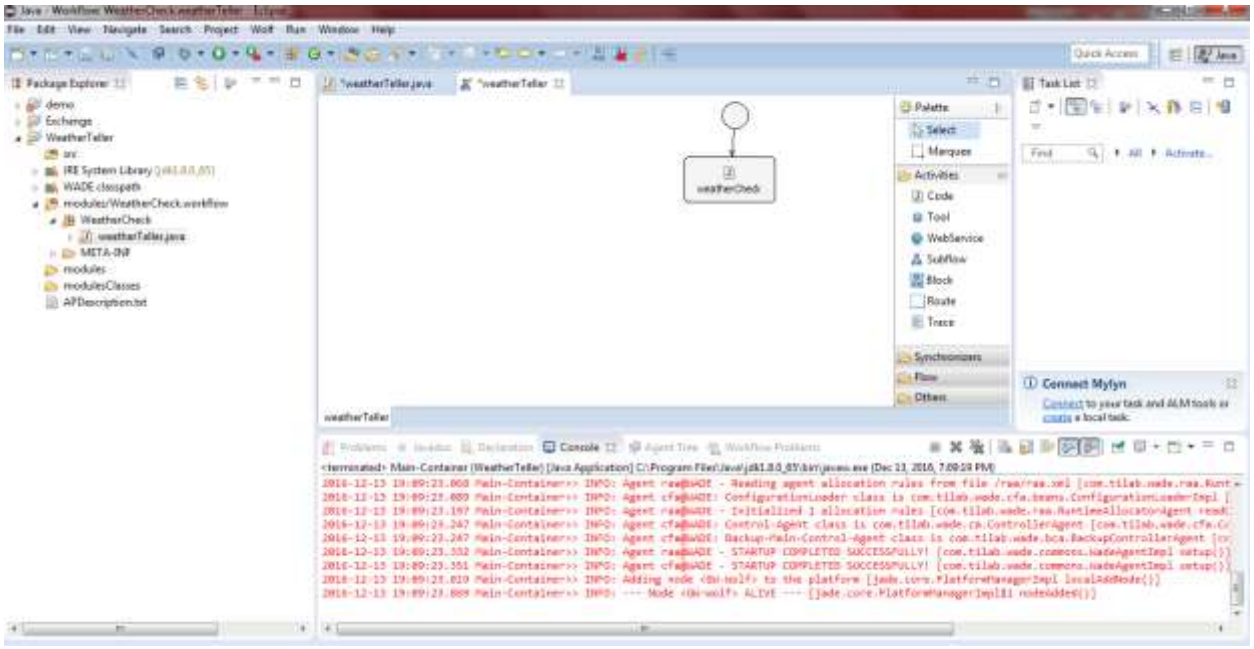
Първата създадена дейност ще бъде именувана wetherCheck и ще

представлява дейност по смисъла на wade, която е от типа webService. Конфигурацията на тази дейност представлява посочването на определен WebService, към който тя да се обръща когато е необходимо. В случая посоченият webService е на уеб-сайт за времето в момента. Посоченият url дава информацията за времето над град София във всеки един момент, в който някой се обърне към адреса.Конфигурирането на уеб адреса е показано на фигура 19.

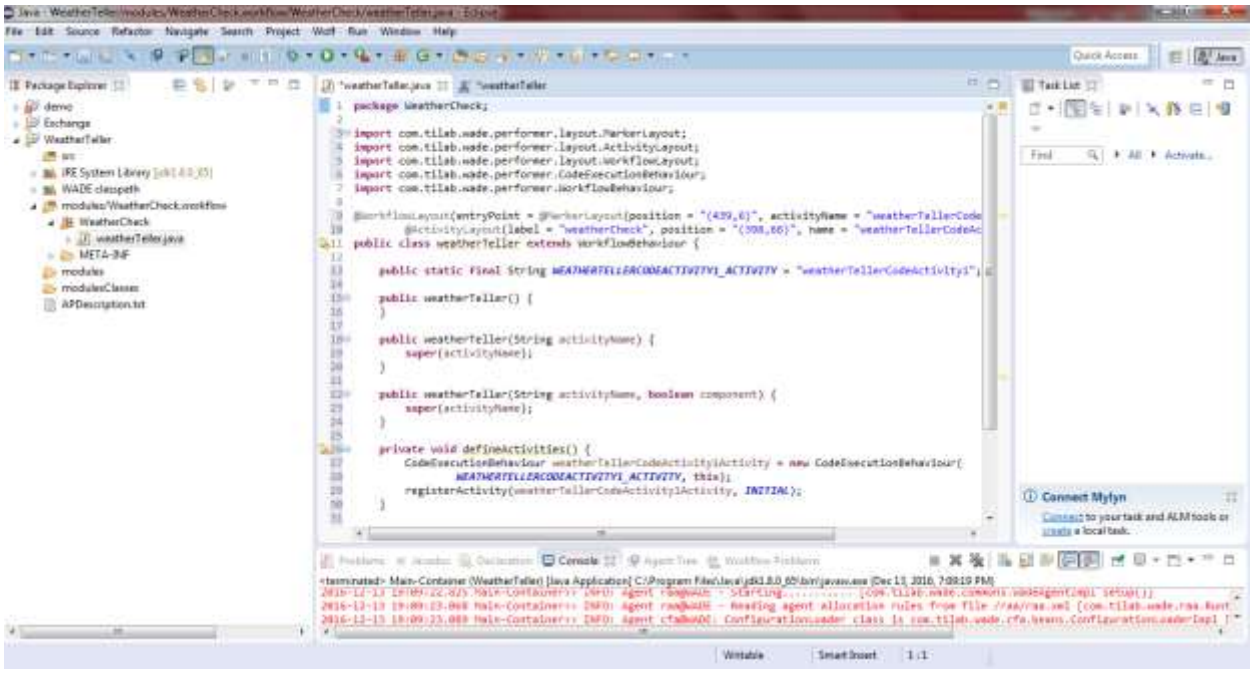


Фиг. 19

Тъй като това е първата дейност,която се създава тя ще бъде от вида Initial, което означава,че това ще бъде първата изпълнена дейност при стартирането на процеса. След потвърждаване на конфигурацията, графичната среда на Wade , представя новосъздадената дейност в графичен вид, заедно със създаване на базовият код на дейността, който не включва конкретни парамаетри. Графично изобразената дейност и автоматично създаденият код са показани на фигруи 20 и 21.



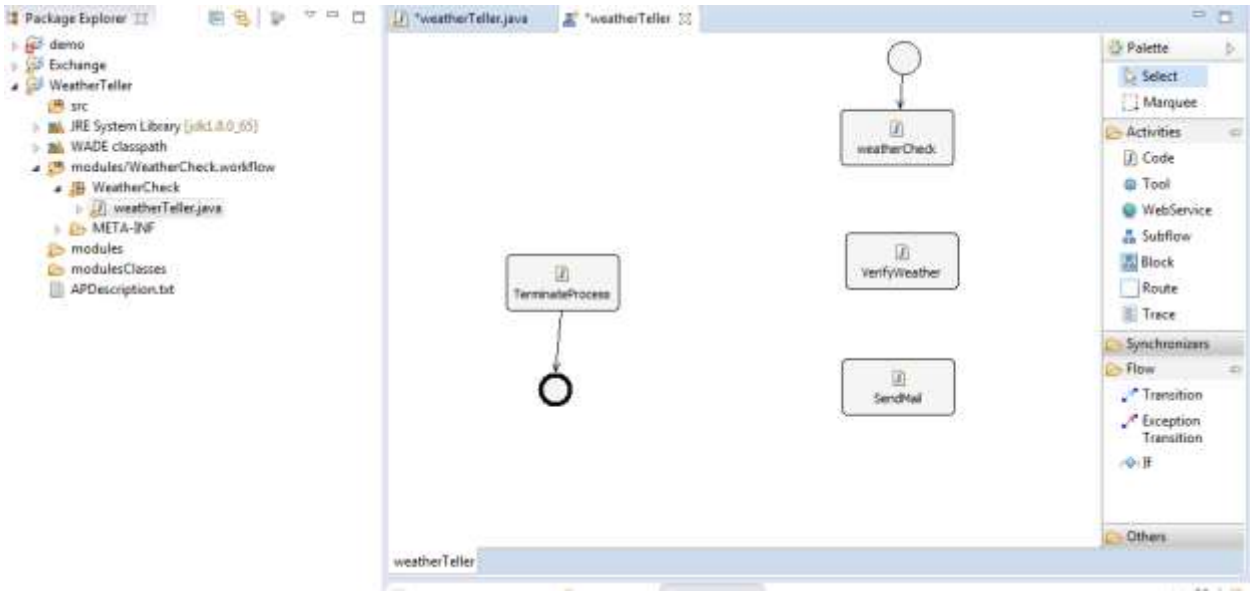
Фиг. 20



Фиг 21.

След получаване на информацията относно времето, посредством обръщане към посоченият url адрес, информацията се предава на следващата дейност с име VerifyWeather, която ще провери състоятнието на времето и дали то отговаря на изискването за уведомяване.

Следващата дейност,която ще изпраща имейл при наличие на дъждовно време, на база получената до момента информация е SendMail и тя представлява последната дейност от процеса. За да може работният процес да бъде автоматично прекъснат след като бъде извършен веднъж е необходимо да се създаде допълнителна дейност, която е с име TerminateProcess и отговаря за прекъсването на процеса след изпълнението му. Графичното изображение на създадените дейности е показано на фигура 22.

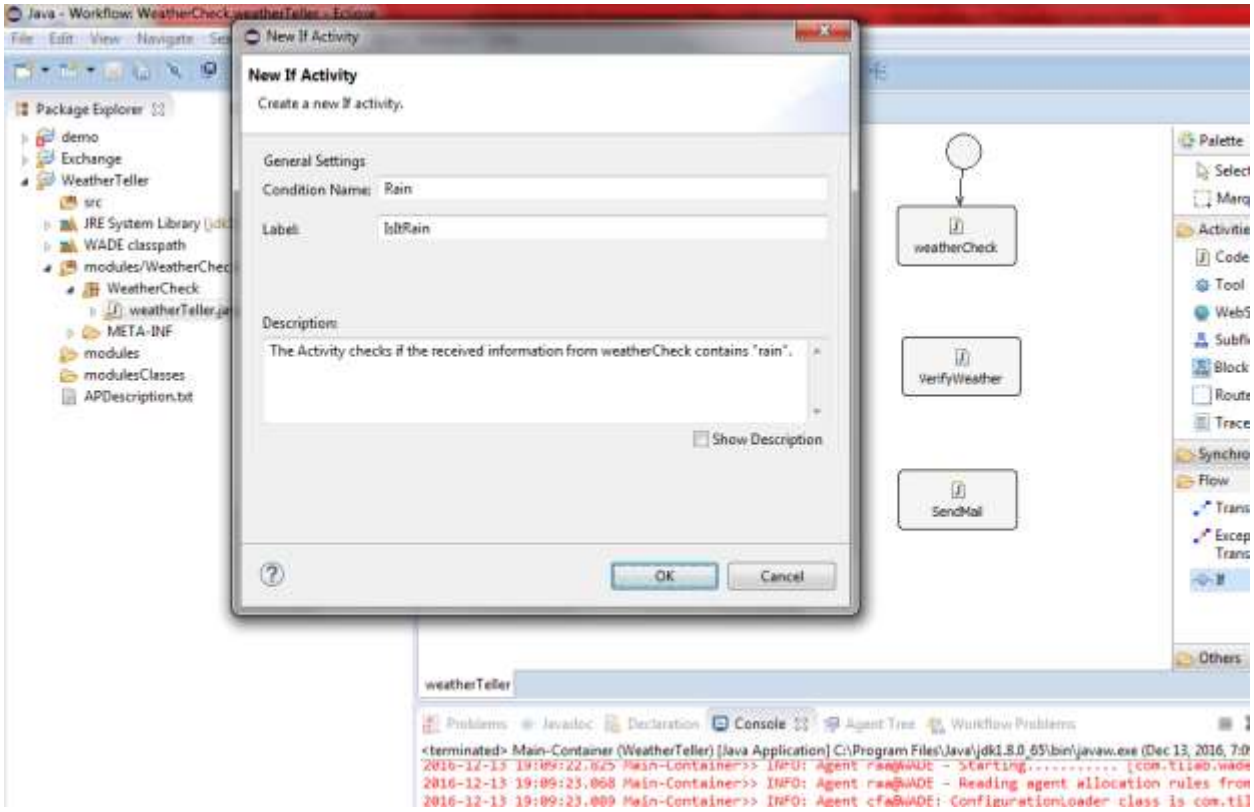


Фиг 22.

Изобразените кръгове над weatherCheck и TerminateProcess обуславят вида на посочените дейности при конфигурацията им. В случая weatherCheck е първична (Initial), тъй като е първата от всички дейности, която се изпълнява при стартиране на процеса, TerminateProcess е крайна (Final) дейност, с която процеса приключва.

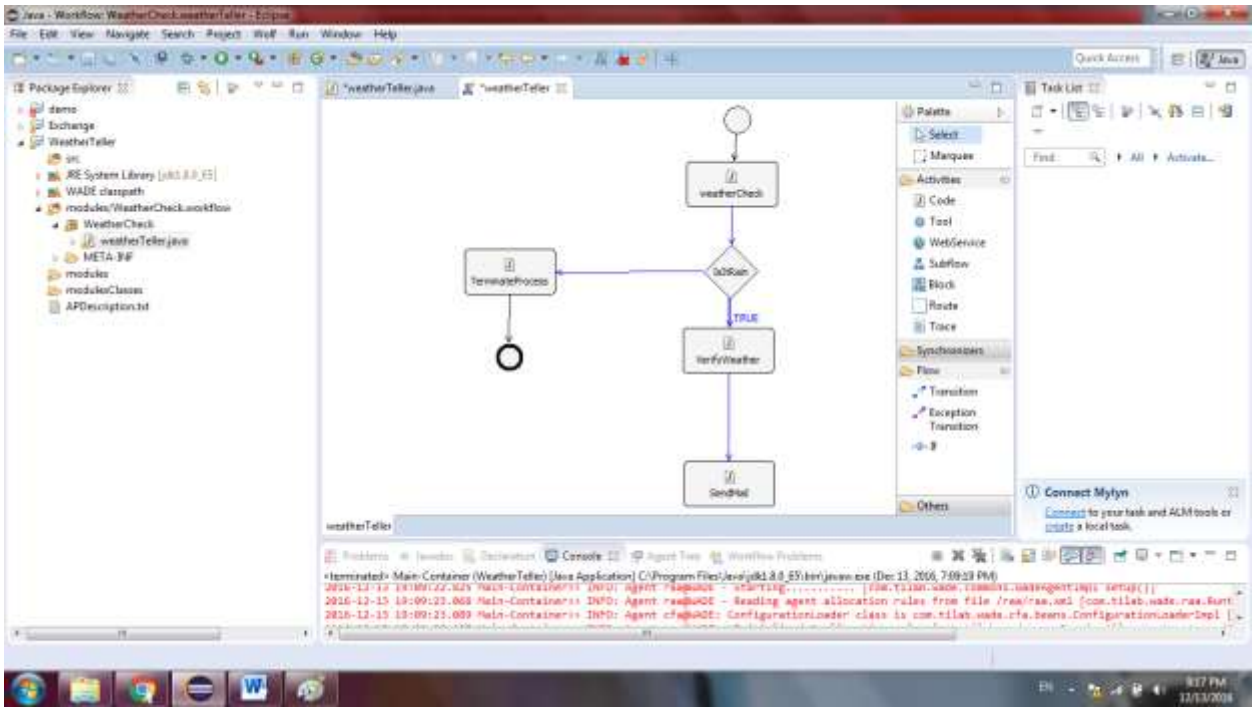
След създаването на дейностите е необходимо да се дефинират и връзките между тях, връзката между първата и втората дейност ще

бъде от вида if transition, със конфиурация за проверка на наличието на rain „дъжд“ , в получената от webService-а информация. Създаването на този вид дейности и дефинирането на условието е поаказано на фигура 23.



Фиг.23

При получаване на информация различна от „дъжд“, ще бъде изпращана входяща информация до дейноста по прекъсване на процеса. При изпращане на информация съдържаща „дъжд“ дейноста verifyMail ще подготви информацията с имейла, съдържаща дата и стаус на времето и ще се свърже с дейноста SendMail, която ще изпрати имейл съдържащ подготвената информация до крайният получател. Условно концепцията на проекта е приложението да се задейства всеки работен ден в 16:45 и да изпрати имейл до човекът X ако в града в който живее вали дъжд.



Фиг. 24

На фигура 24 е представен графично, работният процес на проекта, създаен посредством опциите за графично представяне на процеси на WADE.

Кодът на създаденият работен процес не съдържа конкретните действия на всяка една от дейностите, а само представя тяхното създаване, дефиниране и конфигурации, както и ситуационните им връзки и изглежда по следният начин:

```
package WeatherCheck;

import com.tilab.wade.performer.RouteActivityBehaviour;
import com.tilab.wade.performer.layout.TransitionLayout;
import com.tilab.wade.performer.Transition;
import com.tilab.wade.performer.layout.MarkerLayout;
import com.tilab.wade.performer.layout.ActivityLayout;
import com.tilab.wade.performer.layout.WorkflowLayout;
import com.tilab.wade.performer.CodeExecutionBehaviour;
import com.tilab.wade.performer.WorkflowBehaviour;

@WorkflowLayout(exitPoints = { @MarkerLayout(position = "(189,265)", activityName =
"weatherTellerCodeActivity3") }, transitions = {@TransitionLayout(labelPosition =
"START", labelVisible = false, label = "verifyMail", to =
"weatherTellerCodeActivity4", from = "weatherTellerCodeActivity2"),
@TransitionLayout(labelPosition = "START", labelVisible = false, label = "FALSE", to
= "weatherTellerCodeActivity3", from = "weatherTellerRouteActivity1"),
@TransitionLayout(labelPosition = "START", labelVisible = true, label = "TRUE", to =
```

```
"weatherTellerCodeActivity2", from = "weatherTellerRouteActivity1"),
@TransitionLayout(labelPosition = "START", labelVisible = false, label = "D", to =
"weatherTellerRouteActivity1", from = "weatherTellerCodeActivity1") }, entryPoint =
@MarkerLayout(position = "(439,6)", activityName = "weatherTellerCodeActivity1"),
activities = {
            @ActivityLayout(conditionName = "Rain", label = "IsItRain", position =
"(417,157)", name = "weatherTellerRouteActivity1"), @ActivityLayout(label =
"SendMail", position = "(395,394)", name = "weatherTellerCodeActivity4"),
@ActivityLayout(label = "TerminateProcess", position = "(155,162)", name =
"weatherTellerCodeActivity3"), @ActivityLayout(label = "VerifyWeather", position =
"(394,248)", name = "weatherTellerCodeActivity2"), @ActivityLayout(label =
"weatherCheck", position = "(398,66)", name = "weatherTellerCodeActivity1") })
public class weatherTeller extends WorkflowBehaviour {

      public static final String WEATHERTELLERROUTEACTIVITY1_ACTIVITY =
"weatherTellerRouteActivity1";
      public static final String RAIN_CONDITION = "Rain";
      public static final String WEATHERTELLERCODEACTIVITY4_ACTIVITY =
"weatherTellerCodeActivity4";
      public static final String WEATHERTELLERCODEACTIVITY3_ACTIVITY =
"weatherTellerCodeActivity3";
      public static final String WEATHERTELLERCODEACTIVITY2_ACTIVITY =
"weatherTellerCodeActivity2";
      public static final String WEATHERTELLERCODEACTIVITY1_ACTIVITY =
"weatherTellerCodeActivity1";

      public weatherTeller() {
      }

      public weatherTeller(String activityName) {
            super(activityName);
      }

      public weatherTeller(String activityName, boolean component) {
            super(activityName);
      }

      private void defineActivities() {
            CodeExecutionBehaviour weatherTellerCodeActivity1Activity = new
CodeExecutionBehaviour(
                        WEATHERTELLERCODEACTIVITY1_ACTIVITY, this);
            registerActivity(weatherTellerCodeActivity1Activity, INITIAL);
            CodeExecutionBehaviour weatherTellerCodeActivity2Activity = new
CodeExecutionBehaviour(
                        WEATHERTELLERCODEACTIVITY2_ACTIVITY, this);
```

```
            registerActivity(weatherTellerCodeActivity2Activity);
            CodeExecutionBehaviour weatherTellerCodeActivity3Activity = new
CodeExecutionBehaviour(
                        WEATHERTELLERCODEACTIVITY3_ACTIVITY, this);
            registerActivity(weatherTellerCodeActivity3Activity, FINAL);
            CodeExecutionBehaviour weatherTellerCodeActivity4Activity = new
CodeExecutionBehaviour(
                        WEATHERTELLERCODEACTIVITY4_ACTIVITY, this);
            registerActivity(weatherTellerCodeActivity4Activity);
            RouteActivityBehaviour weatherTellerRouteActivity1Activity = new
RouteActivityBehaviour(
                        WEATHERTELLERROUTEACTIVITY1_ACTIVITY, this);
            registerActivity(weatherTellerRouteActivity1Activity);
      }

      /**
       * <br>
       */
      protected void executeweatherTellerCodeActivity1() throws Exception {
      }

      /**
       */
      protected void executeweatherTellerCodeActivity2() throws Exception {
      }

      private void defineTransitions() {
            registerTransition(new Transition(),
WEATHERTELLERCODEACTIVITY1_ACTIVITY, WEATHERTELLERROUTEACTIVITY1_ACTIVITY);
            registerTransition(new Transition(RAIN_CONDITION, this),
WEATHERTELLERROUTEACTIVITY1_ACTIVITY,
                        WEATHERTELLERCODEACTIVITY2_ACTIVITY);
            registerTransition(new Transition(),
WEATHERTELLERROUTEACTIVITY1_ACTIVITY, WEATHERTELLERCODEACTIVITY3_ACTIVITY);
            registerTransition(new Transition(),
WEATHERTELLERCODEACTIVITY2_ACTIVITY, WEATHERTELLERCODEACTIVITY4_ACTIVITY);
      }

      /**
       */
      protected void executeweatherTellerCodeActivity3() throws Exception {
      }

      /**
       */
```

```
	protected void executeweatherTellerCodeActivity4() throws Exception {
	}

	/**
	 * The Activity checks if the received information from weatherCheck contains
"rain".<br>
	 */
	protected boolean checkRain() throws Exception {
		return true;
	}
}
```

WADE е изграден на основата на JADE и сам по себе си представлява разширение на JADE. Всяко WADE базирано приложение е също и JADE базирано приложение и като такова е структурирано като определен брой контейнери (разпространени в набор от хостове), всеки от които съдържа един или няколко агента. Както и при JADE , главният контейнер в WADE, който съдържа AMS и DF агентите, трябва да бъде активиран първи. Всички останали контейнери се регистрират към главният контейнер при стартирането си. Основната разлика между двата софтуера е възможността за създване, наблюдение и управление на задачи и действия в смисъла на работни процеси. В описаните базови експерименти с двата софтуера, е видно, че при създаването на агентите в JADE е необходимо да бъде конфигуриран всеки един от агентите, посредством избран конкретен метод, действие, вид поведение, видове на изпращаните или получени съобщения. Онтологиите при JADE са по тежки за изпълнение, тъй като са насочени към конкретни съобщения и тълкуването на съдържанието им в детайли. При

описанието на онтологиите в JADE е необходимо те да се дефинират за комуникацията между два или повече агента. Онтологиите, които се използват в WADE са по леки за употреба, защото те се отнасят за абстрактно тълкуване на процеси. Например моментното състояние на един процес може да бъде запаменетно и да бъде дефинирано, посредством онтология като целево състояние. Така в случай, че процеса се разпадне или спре по някаква причина и приложението спре и се рестартира, разпространението до агентите и платформите няма да спре. От друга страна поради същността на работните процеси и възможността при промени те автоматично да бъдат дистрибутирани до всички агенти и контейнери, употребата на онотолоии при някой работен процес обхваща и всички агенти и контейнери участващи в изпълнението му. Графичният редактор на процеси е другото голямо предимство на WADE пред JADE. Той позволява създаване на процеси посредством готови модули, агенти и връзките между тях, без нуждата от добри познания по програмиране и позволява употребата му, не само от програмисти. При употребата на графичният едитор на процеси, при добаввяне на нови дейности, WADE автоматинчо създава голяма част от кода на всеки елемент, като задава рамките, действията, връзките и условията във кода на елементите. По този начин се спестява много време писане на код, а това от своя страна позволява по задълбочен поглед над концепциите на процесите и тяхното управление.

С оглед на изложеното, може да се заключи, че многоагентните системи тепърва навлизат в управлението на работните процеси и имат широко поле за развитие в тази сфера. Фактът, че компаниите и организациите започват да поглеждат повече към качествтото, печалбата и ефективността при изпълнение на работните процеси в тях, и вече не се задоволяват с факта че процесите просто се изпълняват, е ясен знак за потенциала, които предлагат агентно-базираните решения за управление на бизнес процеси. Проведените и описани експерименти разкриват само базовите концепции за разработване на много-агенттни системи. Една от основните причини агентите да навлизат все повече в бизнеса е възможността за автономност при извършването на определени процеси и изключването на необходимостта от външно наблюдение над изпълненото.Тъй като при имплементиране на агени за изпълнението на конкретни задачи, нуждата от човешка намеса в тези процеси престава да съществува, може да се твърди, че освен автоматизирането на процесите, агентите намаляват влиянието на човешкият фактор и възможностите за допускане на грешки.

4. АВТОМАТИЗИРАНЕ И УПРАВЛЕНИЕ НА БИЗНЕС ПРОЦЕСИ

ИЗПОЛЗВАЙКИ GOOGLE.

С оглед на масовата употреба на платформи за електронна поща и по конкретно Google , автоматизиране на бизнес процеси може да бъде извършено и с помощта на продуктите на компанията съпътстващи приложението за електронна поща Gmail. Според изявление на Дънкан Исаксен , публикувано в платформата Quora на 29 декември 2015 г, Google се използва от около 5 милиона компании:

„ Като част от Условията за обслужване и поверителност, Google не публикува списък със всичките си клиенти, нито разкрива точни цифри относно тях.

Спред тях те имат грубо около 5 милиона бизнеса по цял свят и за някои от тях може да бъде прочетено в Google Apps for Work – Customers. “

В голяма част от компаниите все още се използва до голяма степен екселски документи за съхранение представяне и обработка на данни. Алтернатнивата представена от Google в лицето на Google sheets, впряга пълният потенциал не версията на Microsoft, като надгражда фунцкионалностите до степен на възможност за взаимодействие с

конкретни скриптове написани в Google Apps Script (GAS), с цел решаване на конкретни задачи и проблеми. За разлика от многоагентните системи действието на автоматизирането посредством програмните продукти на Google е ограничено до самите програмни продукти и съответно до процесите които са свързвани или могат да бъдат свързани с тях.

В следващите точки ще бъде описан конкретен процес свързан с администрирането на заявки за ремонти и неговото автоматизиране използвайки продукти на Google. Изпълнението на автоматизацията се осъществява посредством Google Forms, Google sheets и Google Apps Script.

4.1. Администриране на процеса по заявяване и одобрение на ремонти.

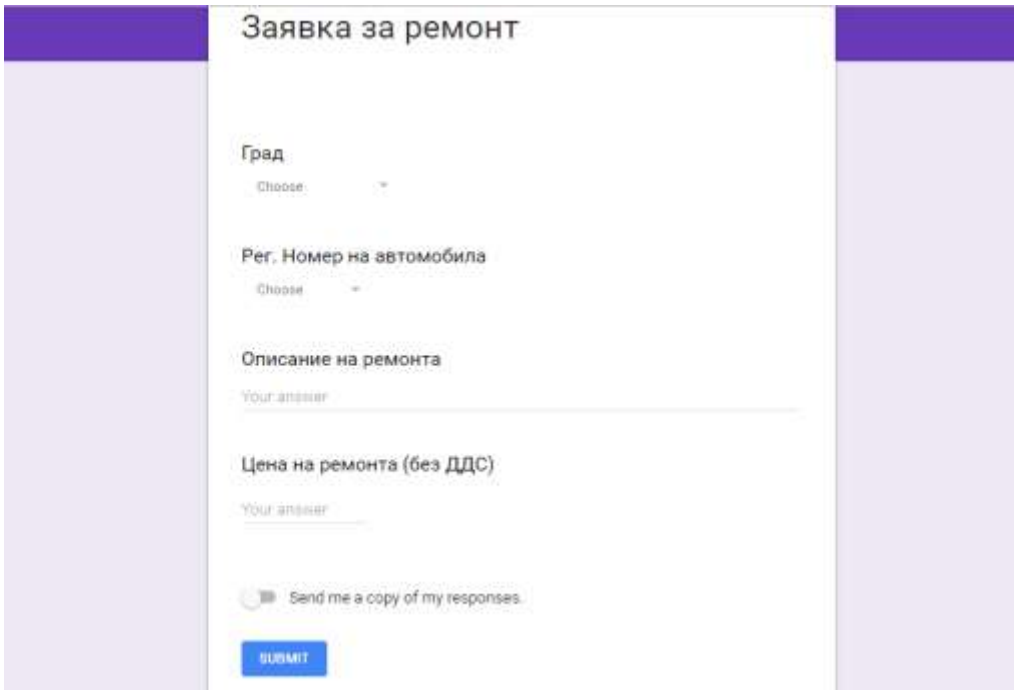
Процеса по заявяване на ремонти за автомобили в една куриерска фирма, представлява последователност от подаване на заявка за ремонта, прегледа на заявката от отговорника за автопарка, нейното и одобряване/отхвърляне. Заявките се разпечатват и подписват от служителят отговорен за съответният регион и се изпращат физически до централният офис на фирмата базиран в София. Тъй като фирмата разполага с 15 офиса в страната, един служители от централният офис е натоварен със задачата да консолидира получените заявки, да ги предава за одобрение на отговорника на автопарка и след това да уведомява за решението подалият заявката служител. Поради множеството заявки и бавното транспортиране на хартиените носители проследимостта и възможността за анализ на разходите, съответно за тяхното контролиране е изключително трудна и неточна. В следващата точка ще бъде описан начина по който този процес е заменен от програмните продукти на Google.

4.2. Автоматизиране на процеса посредством създаване на форма Google Form за изпращане на заявки.

За целта на процеса е необходимо да бъде въведена алтернатива на хартиените заявки, с която те да бъдат изпращани до централният офис електронно,но същевременно тя трябва да запазва идентификацията на изпращача. Опцията на Google forms да събират

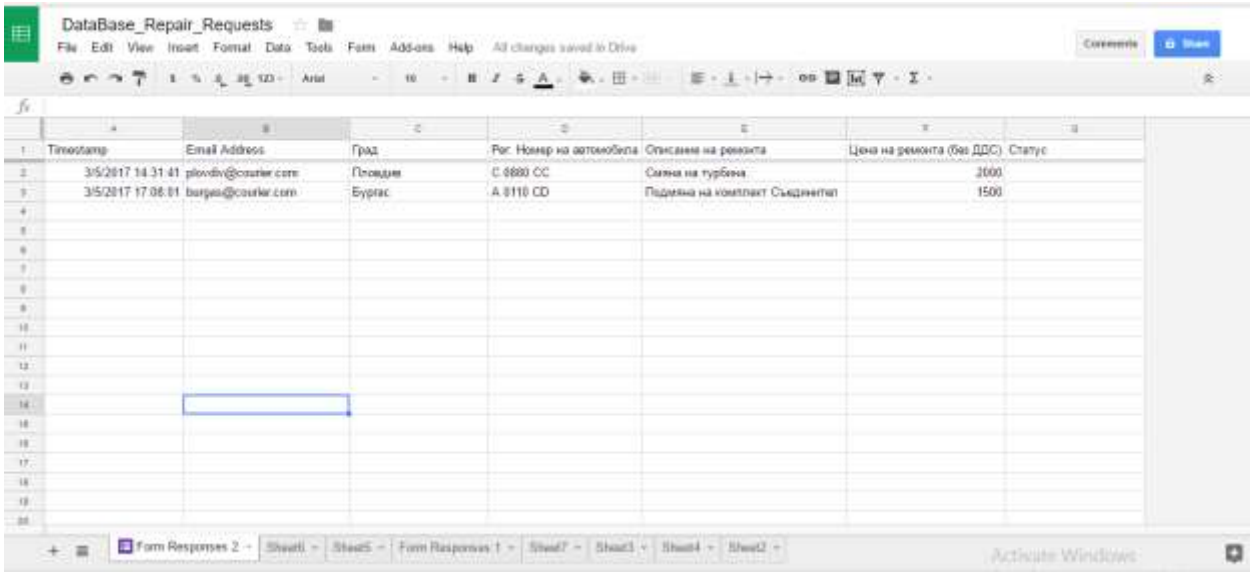
данните от отговорите на потребителите в Google sheet е подходяща за консолидирано съхраняване на информацията във вид на таблица, дефинирана посредством колони и редове.

Първата стъпка е дефинирането на инфорамцията, която трябва да съдържа всяка една заявка. В новата форма се определят полетата които трябва да се попълнят при нова заявка, както и видът информация, която те трябва да съдържат (текст, цифра и др.). Завършеният вид на създаената форма е показан на фигура 25.



Фиг. 25

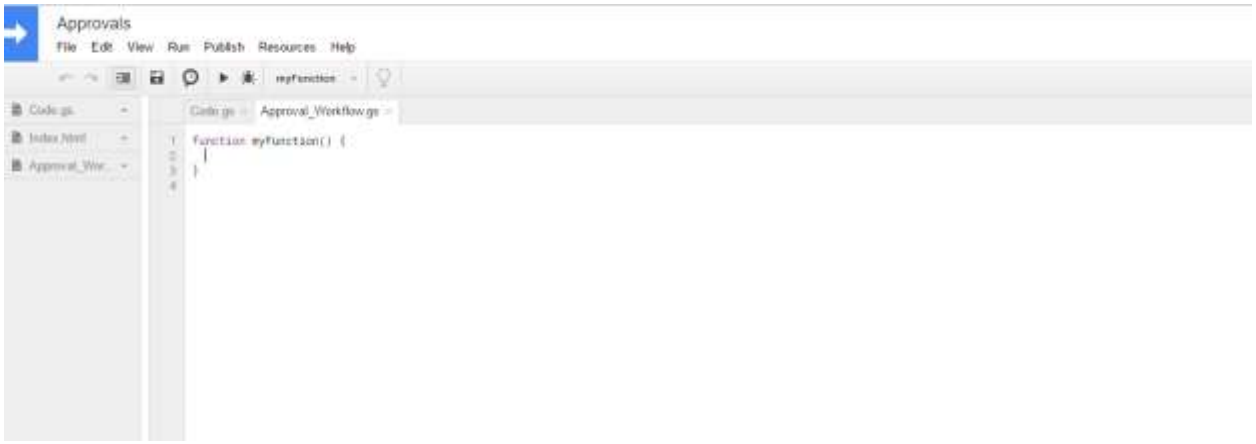
Постъпилата информация от всяка една заявка се зпазва в Google Spreadsheet, който ще представлява базата данни, на процеса, както и конзола за одобрение на заявките. При създаването си и при записване на първата заявка, формата създава нов файл в който именува клетките от първият ред с имента на полетата, в които се записва информацията. След последната автоматично създадена колона във файла, се добаввя допълнителна колона с име „Статус“, която ще бъде показателя за одобряване. Името на файла ще бъде DataBase_Repair_Requests. Видът на файла след добавянето на колона Статус е показан на фигура 26.



Фиг. 26

След като са конфигурирани входните величини е необходимо да

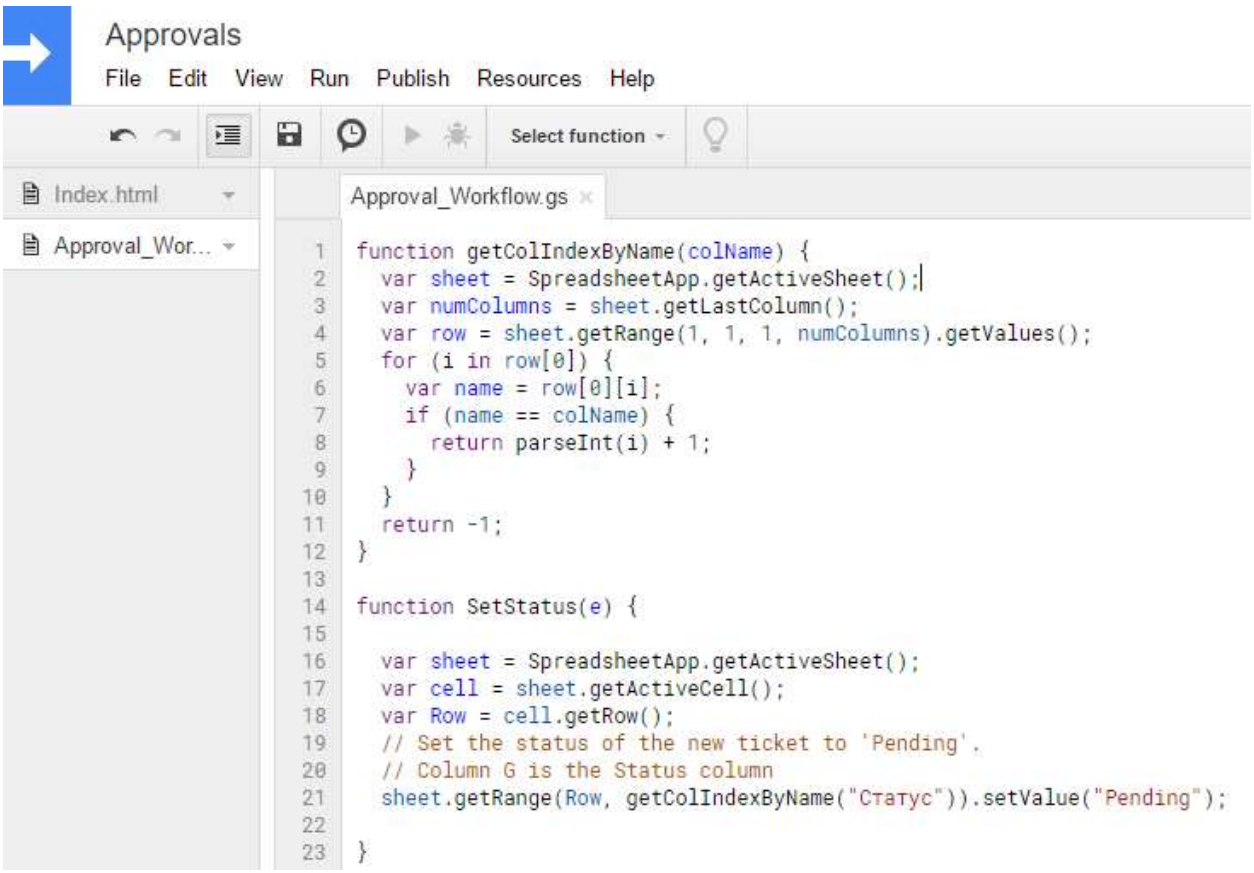
бъде добаввен кодът в GAS , с който ще бъде автоматизиран файлът DataBase_Repair_Request. Добавянето на скрипт към spreadsheet файловете се извършва посредствм менюто Tools❼ Script Editor. Името на скрипта е Approval_Workflow и видът му преди изписването на кода е показан на фигура 27.



Фиг. 27

4.3. Автоматизиране на Google Spreadsheet посредством употреба на Google Apps Script.

Google Aps Script е базиран на Java език, представляващ множество класове и подкласове за управление, редактиране и автоматизиране на всички видове Google приложения. Като част от глобалният проект на компанията G-suite (Google for Work) GAS може да се използва в комбинация и комбинирайки всички приложения съдържащи се в пакета (Gmail, Google Drive, Google Docs, Goole Calendar). В случая GAS ще бъде използван за автоматизиране на документ, който ще изпраща имейли на база получените заявки и последващото им одобрение/отказ. Употребата на скриптове за управление на документи в Google представлява дефинирнае посредством код на отделни функции, всяка от които извършва конкретно действие, и е конфигурирана да се активира по определен начин. За целта на проекта по одобрение на заявките е необходимо да бъде конфигурирано, автоматичното задаване на статус на всяка нова заявка. Тъй като всяка нова заявка се записва на нов ред, статуса на всяка заявка ще представлява информацията от клетката в колона G на редът корепондиращ със заявката. Колона G е със въведен Data Validation rule , който позволява само три вида запис: “Pending” , “Approved” “Rejected”. Функцията която ще отговаря за задавнето на статус “pending” (чакаща) се нарича SetStatus () и в скрипт едитора изглежда както е показано на фигура 27.



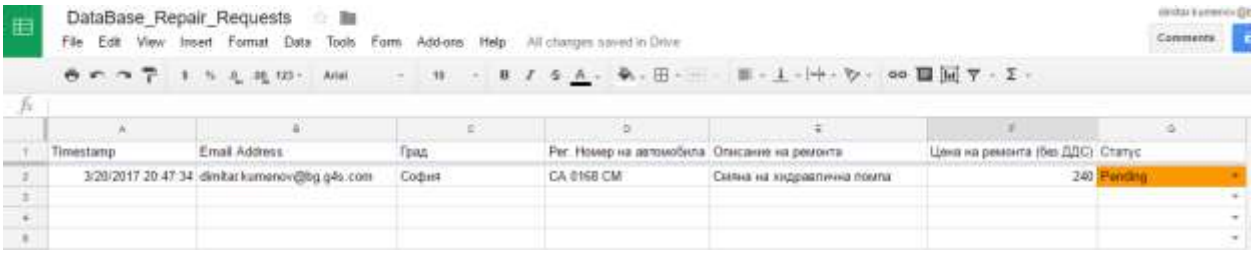
Фиг. 27

За да могат да бъдат използвани имената на колоните като идентификатори, във функциите, които се записват в GAS, е необходимо първо да бъде създадена фукцията getColIndexByName (), както е показано на Фиг. 27.

Една от опциите на ScriptEditor в Google Docs е възможноста за задавне на Triggers за изпълнение на функциите, които се записват посредством Java базиран код в него. Тези „Спусъци“ определят кога ще се задейсва всяка една от функциите. SetStatus () очаква нова заявка и в момента в който бъде въведен нов запис посредством формата, записва “Pending” в колона „Статус“. Trigger-а който е избран е показан на фигура 28.

Фиг. 28

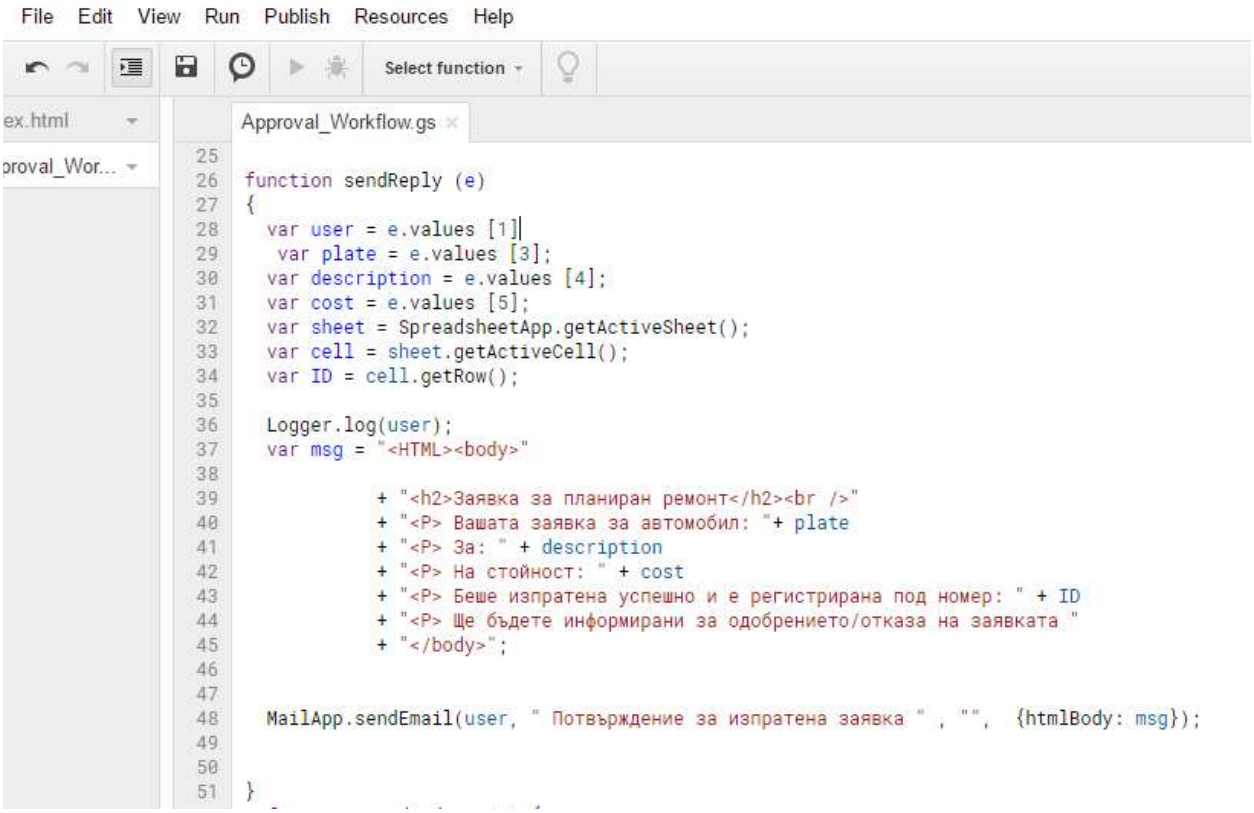
За да се тества функцията е изпратена тестова заявка посредством формата и записа във файла, с добавен статус както е показано на фигура 30.



Фиг 30.

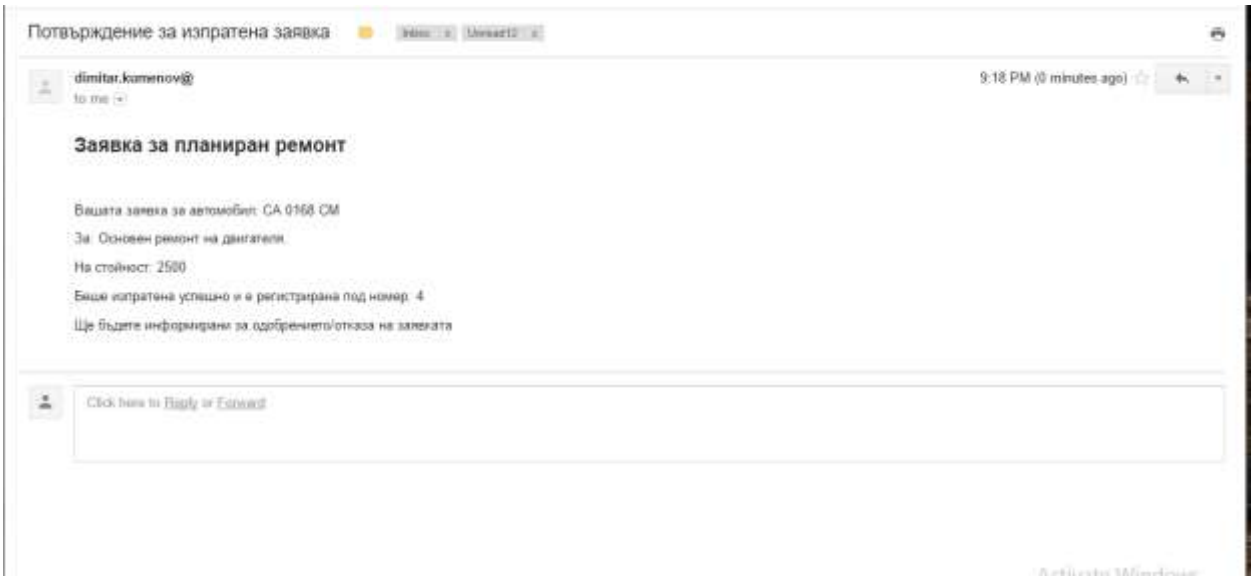
Следващата функция ще изпраща обратна връзка към подателя на заявката непосредствено след изпращането и. Информацията ще съдържа и номер на заявката. Функцията е с име sendReply () и тя взима определена информация от записа за всяка заявка, формирайки по този начин информативно съобщение с добавен номер и статус. За всеки вид информация, която кодът трябва да формира като съобщение, се

представя колоната и редът от който трябва да бъде информацията във вид на Variable “Променлива“. По този начин в изготвянето на съобщението могат да се ползват директно тези променливи, а не да се изписва отново исканият ред/колона/клетка. Готовото съобщение се изпраща до изпращача на заявката използвайки имейла адреса му, който се записва при попълване на формата. В скрипт едитора, кодът на функцията изглежда както е показано на фигура 31.



Фиг. 31

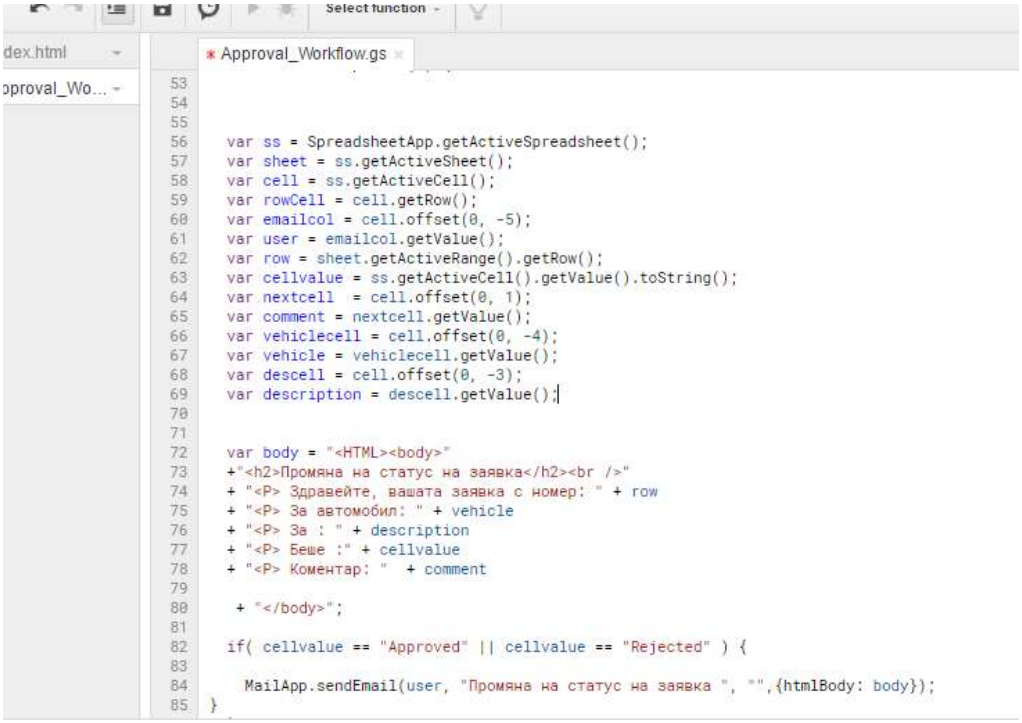
Trigger за тази функция ще бъде същият като на функцията setStatus () (On Form Submit). След задаването му, е изпратена тестова заявка с цел тестване на кодът. Полученият обратно имейл е показан на фигура 32.



Фиг 32.

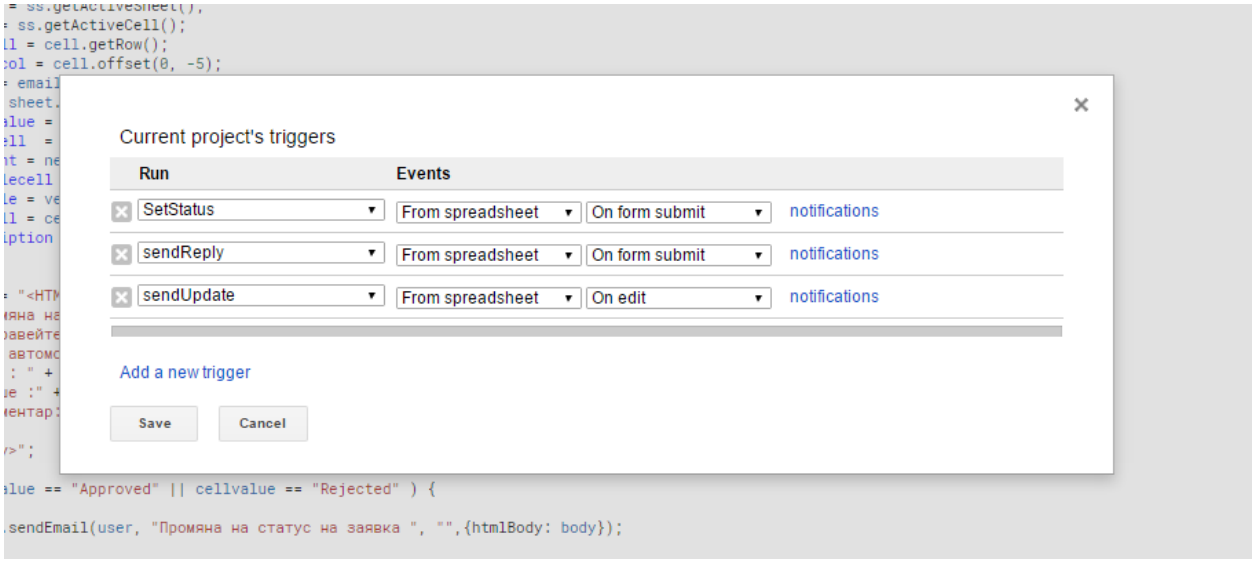
Процесът по одобряване/отказване на заявките се извършва

посредством промяна на статуса на всяка една от заявките в колона G от файла DataBase_Repair_Requests. Одобряващият има възможност да разгледа всички чакащи заявки и да избере кои от тях да одобри. За да може подателя на заявката да бъде информиран при нейното одобрение/отказ е създадена функцията sendUpdate (), която при промяна на статуса на някоя заявка, изпраща информативен имейл до подателя на заявката с променениият статус на заявката. Функцията ще бъде с Trigger – From Spreadsheet ❼ On Edit , който ще я активира когато информацията в някоя клетка във файлът бъде променена. Тъй като промяна на информацията има само в колона „Статус“ промяната на статуса, на някоя заявка ще активира функцията. В началото функцията, дефинира информацията, която ще е необходима за да се изпрати съобщението. Условието за изпращане на имейл е състоянието на променената клетка (тъй като така написан кода в комбинация с този Trigger е валиден за всички клетки във файла) е “Approved” или „Rejected”. Кодът на функцията sendUpdate () е показан на фигура 37



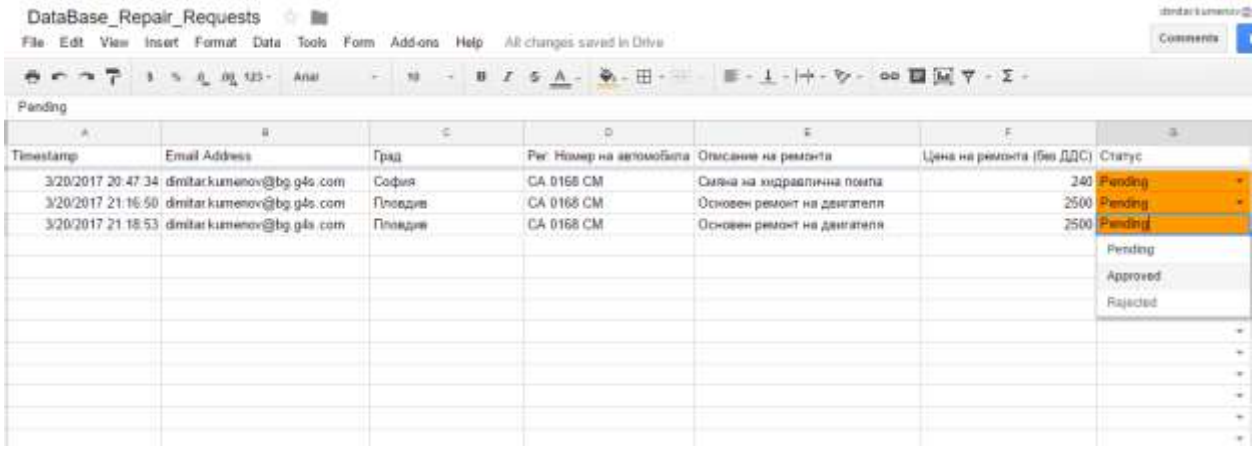
Фиг. 37

След конфигуриране на Trigger и на тази функция, зададените до момента в приложението „спусъци“ изглежда както е показано на фигура 38



Фиг. 38

За да бъде тествана функцията е променен статуса на една от заявките както е показано на Фиг 39. и съответно полученият имейл, който е показан на фигура 40.


DataBase_Repair_Requests
File Edit View Insert Format Data Tools Form Add-ons Help All changes saved in Drive
Comments
Pending
Timestamp
Email Address
Град
Рег. Номер на автомобила
Описание на ремонта
Цена на ремонта (без ДДС)
Статус
3/20/2017 20:47:34 dimitar.kumenov@bg.g4s.com София CA 0168 CM Смяна на хидравлична помпа 240 Pending
3/20/2017 21:16:50 dimitar.kumenov@bg.g4s.com Пловдив CA 0168 CM Основен ремонт на двигателя 2500 Pending
3/20/2017 21:18:53 dimitar.kumenov@bg.g4s.com Пловдив CA 0168 CM Основен ремонт на двигателя 2500 Pending
Pending
Approved
Rejected

Фиг 39.


More
Промяна на статус на заявка
Inbox
Unread 12
dimitar.kumenov@
to me
10:19 PM (0 minutes ago)
Промяна на статус на заявка
Здравейте, вашата заявка с номер: 4
За автомобил: Пловдив
За : CA 0168 CM
Беше :Approved
Коментар
Click here to Reply or Forward

Фиг 40.

Употребата на GAS съвместно с другите приложения на Google дават множество възможности за оптимизиране, подобрение, управление и автоматизиране на процеси. Крайните продукти могат да бъдат пригодени за спецификите на всеки бизнес, благодарние на възможността за адаптивност на Java базираният код. Употребата на Google Sheets е удобна за създаване на автоматизирани системи за мониторинг и информиране от вида “ticket system” , който позволяват лесно управление на информацията, изготвяне на анализи, идентифициране на трендове и пр. Употребата им дава много възможности на мениджърите да изпълняват своята функция дори мобилно, позволявайки им по този начин да се фокусират върху по-важни задачи от одобряването на заявки. Основното предимство е, че за да бъде създадено приложение с помощта на GAS не са необходими познания по програмиране. В мрежата съществуват множество готови кодове с подробно описание на това какво точно извършват и са достъпни от всеки. Конкретната употребта на GAS за управление и оптимизиране на процеса по одобрение на заявки, е въведена успешно в експлоатация и работи безотказно.

Заключение

Употребата на многоагентните системи в управлението на бизнес процеси е навлизащ и все по-предпочитан метод в корпорациите, във връзка с управлението на бизнес процеси. Предимствата на възможността за автономно изпълнение на рутинни и административни дейности са много, поради повишаването на ефективността и оптимизиране на част от процесите, изпълнявани в организацията, както и поради възможностите за фокусиране на вниманието на мениджъри и служители към по-важни и печеливши процеси и проблеми. Собственият ми опит в международна компания, потвърждава нуждата от оптимизиране на процесите в големите компании, поради загубите които търпят те от „тромави“ бизнес процеси, липса на стриктност при изпълнението на процеси, както и от ангажиране на служители с рутинни дейности. Проведените експерименти с многоагентният софтуер Jade и Wade описани в тази магистърска теза са извършени лично от мен и разкриват само малка част от възможностите на употребата на агенти за изпълнението и управлението на определени процеси. Създаването

на описаният автоматизиран процес за одобрение на заявки за ремонти бе създаден от мен, поради нуждата от оптимизиране на трудоемкият и бавен процес по одобрение на заявките на хартия. Създаването на работещата платформа бе съвкупност от имплементиране на единични действия, посредством Google Apps Script , с цел изграждане на цялостен процес, който да се извършва по начин, определен от политиките на компанията. Възможността на Java базираните системи за управление на бизнес процеси да се адаптират към средата в която трябва да работят е ключова и определяща при избора на подход за автоматизиране и повишаване на ефективността на продесите във всяка организация. Според мен следващата стъпка в развитието и еволюцията на съвременният бизнес е въвеждането на интелигентни агенти и системи за управление и изпълнение на бизнес процесите. Подхода за уптотребата на агенти е сравнително нов и непознат за голяма част от мениджърите в световетн мащаб, но резултатите ще доведат засилване на избора му в следващите години.

Използвани източници:

1. http://www.unob.cz/eam/Documents/Archiv/EaM_3_2011/%C5%A0PERKA.pdf
2. http://www.bptrends.com/publicationfiles/01-07-ART-MakingtheCaseforBPM-BenefitsChecklist-Rudden.pdf
3. http://users.ecs.soton.ac.uk/nrj/download-files/aaij991.pdf
4. http://jade.tilab.com/doc/tutorials/JADEAdmin/startJade.html
5. http://jade.tilab.com/wade/doc/tutorial/WADE-Tutorial.pdf
6. http://2011.eclipse-it.org/ProcEclipse-IT11/Technical/1712.pdf
7. https://developers.google.com/apps-script/