Увод

Cloud Computing[1] навлиза неусетно в ежедневието на хора и все повече се превръща в неизменна част от живота ни. Този процес на интегриране започва с появата на Youtube, Facebook, Google Docs, Concur Travel & Expense и други SaaS[2] решения. Повечето потребители на тези усулуги не правят разлика между SaaS[2] и статичен Website[3]. За разлика от тях, разработчиците на облачни приложения ежедневно се сблъскват с предизвикателства и трудности, свързани с процесите на имплементация, интегриране, Application Lifecycle Management[4] и други.

Първото изпитание на което са подложени софтуерните инжeнери е инсталирането на всички необходими инструменти преди да започне процеса на разработване. Примери за това са специфичните за проекта Runtime[5], IDE[6], Build System[7], Framework[8], Server[9], Source Control System[10], SDK[11] и други. Следва цикълът на разработка, писане на тестове и продуктивен код. За да се провери дали текущата имплементацията работи коректно се използва тестова система (най-често локален Server[9]). След като всички тестове се изпълняват успешно и продуктивния код е готов, следва да се изпробва, дали програмата може да работи не само в тестовата среда, но и в условията на реалната система. Преди да се изпробва поведението на програмата в реалната среда, трябва да се мине през процес на Build[12] на приложението и всички необходими артефакти. Следващата стъпка е Deploy[13] в реалната система.

Тези стъпки накратко описват как протича един пълен цикъл от процеса на разработване, тестване, Build[12] и Deploy[13] на дори и най-простото облачно приложение. Най-често само един цикъл не стига за да се завърши програмното задание. Така процесът на работа от еднократен се превръща в итеративен. В повечето случаи за завършването на един пълен цикъл е необходимо значително време, което в комбинация с итеративния подход за решаване на поставената задача, води до драстично понижен Turnaround Time[14] и многократно увеличeн Overhead[15].

Текущата дипломна работа има за цел да представи ползите от използването на съвременни технологии като Cloud Computing[1] и WebIDE[16], в съчетание с добре познати концепции като In-System Development[17] и RAD[18]. Симбиозата на тези технологии и концпеции води до значително улесняването на софтуерните инженери при разработването на облачни приложения, максимален Turnaround Time[14] и минимален Overhead[15].

Мотивация

Ползите от избора на WebIDE[16] пред стандартно Desktop IDE[19] са повече от една. Най-очевидната полза е, че изискванията към компютрите, използвани за разработка, намаляват драстично (скорост на изчисленията, оперативна памет, твърд диск и други). Единственото, от което имат нужда разработчиците, е добра интернет връзка и съвременен Browser[20]. Само това подобрение води до следните резултати - намалени оперативни разходи, разходи за амортизация и морално изхабяване. Друг не толкова очевиден плюс за софтуерните фирми, е липсата на време за първоначална инсталация на всички необходими инструменти и настройки на работната среда. Това води до минимален Overhead[15] преди започване на разработване.

Съчетанието между WebIDE[16] и In-System Development[17] води до значително предимство спрямо стандартния модел на работа. Липсата на времена за Build[12] и Deploy[13] ускорява допълнително работния процес. Програмирането директно върху реална/работна среда елиминира грешките, които се получават при транспортиране на готовия продукт от тестовата система към реалната среда. Всичко това е възможно благодарение на принципите взети от програмирането на микроконтролери и логически схеми работеща върху платка, така наречения In-System Programming[21] модел.

Комбинацията на горните технологии и концепции с RAD[18] техники и принципи като използването на Template[22], Wizard[23] и други, значително намаляват Turnaround Time[14] и ускоряват процеса на имплементация. RAD[18] техниките до голяма степен елиминират писането на Boilerplate Code[24] и спомагат за концентрация на усилията върху съществената бизнес логика.

Цел

Целта на дипломната работа е проектирането и реализирането на модули за RAD[18] инструменти за In-System Development[17] на облачни приложения. Реализираните модули трябва да могат да се използват в процеса на разработката на облачно приложение.

За постигането на поставената цел, в дипломната работа са дефинирани следните задачи:

* Да се направи обзор и сравнителен анализ на съществуващи WebIDE[16].
* Проектиране и интегриране на модули за избраното WebIDE[16].
* Създаване на облачно приложение с избраното WebIDE[16].

Обзор и сравнителен анализ на съществуващи решения

Обявената от Eclipse[25] инициатива за създаване на индустриален стандарт за разрабоване в облака, показва че сме достигнали до нов етап в използването на инструменти за създаване на софтуер. В момента се намираме в началото на преход от използване на Desktop IDE[19] и On-Premises[26] софтуер към WebIDE[16] и On-Demand[27] услуги (Ref. 1).

В настоящата дипломна работа ще бъдат разгледани следните проекти, част от които участват в ECD[28] - инициатива за създаване на индустриален стандарт за разработване в облака:

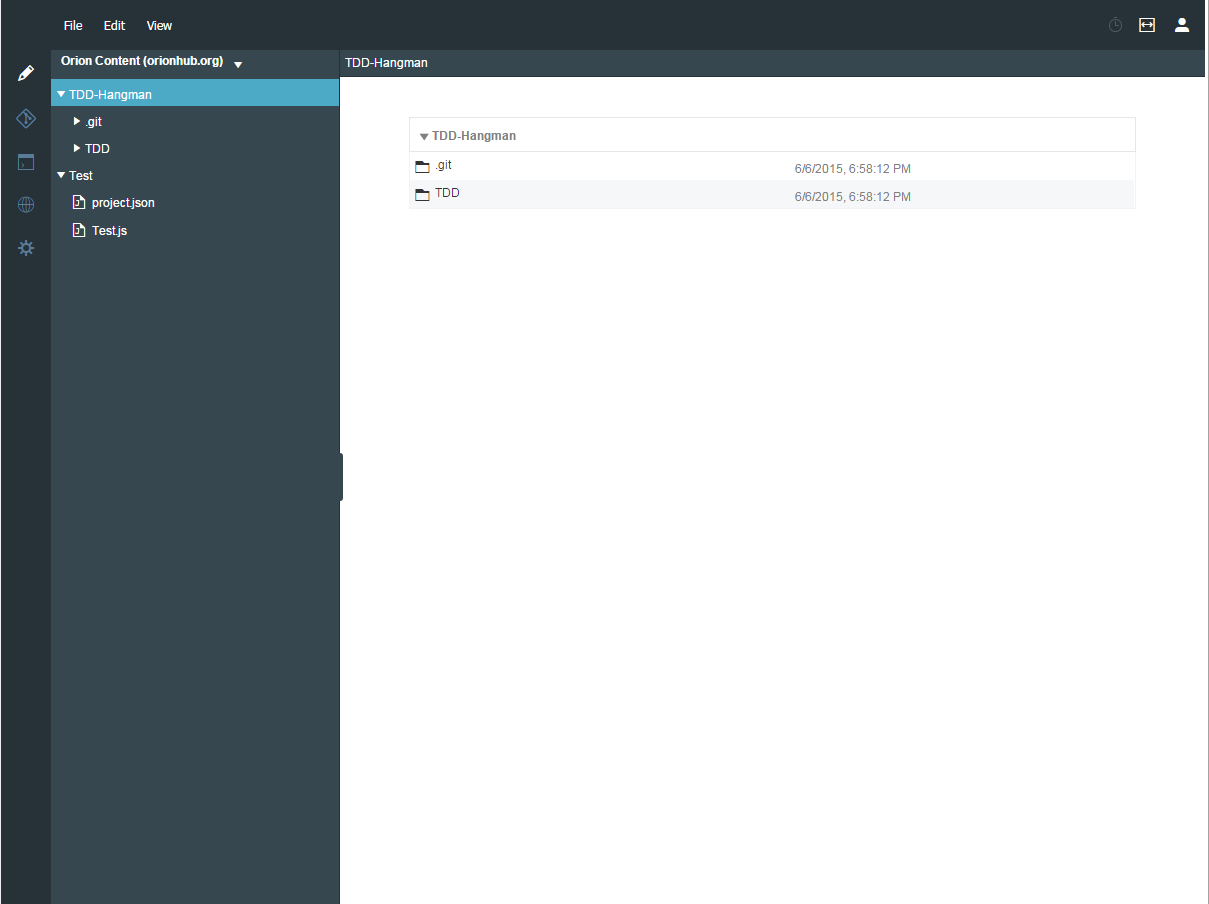
* Eclipse Orion (IBM)
* Eclipse Flux (Pivotal)
* Eclipse Che (Codenvy)
* Eclipse Dirigible (SAP)
* Cloud9 (Cloud9 IDE, Inc)

Eclipse Orion

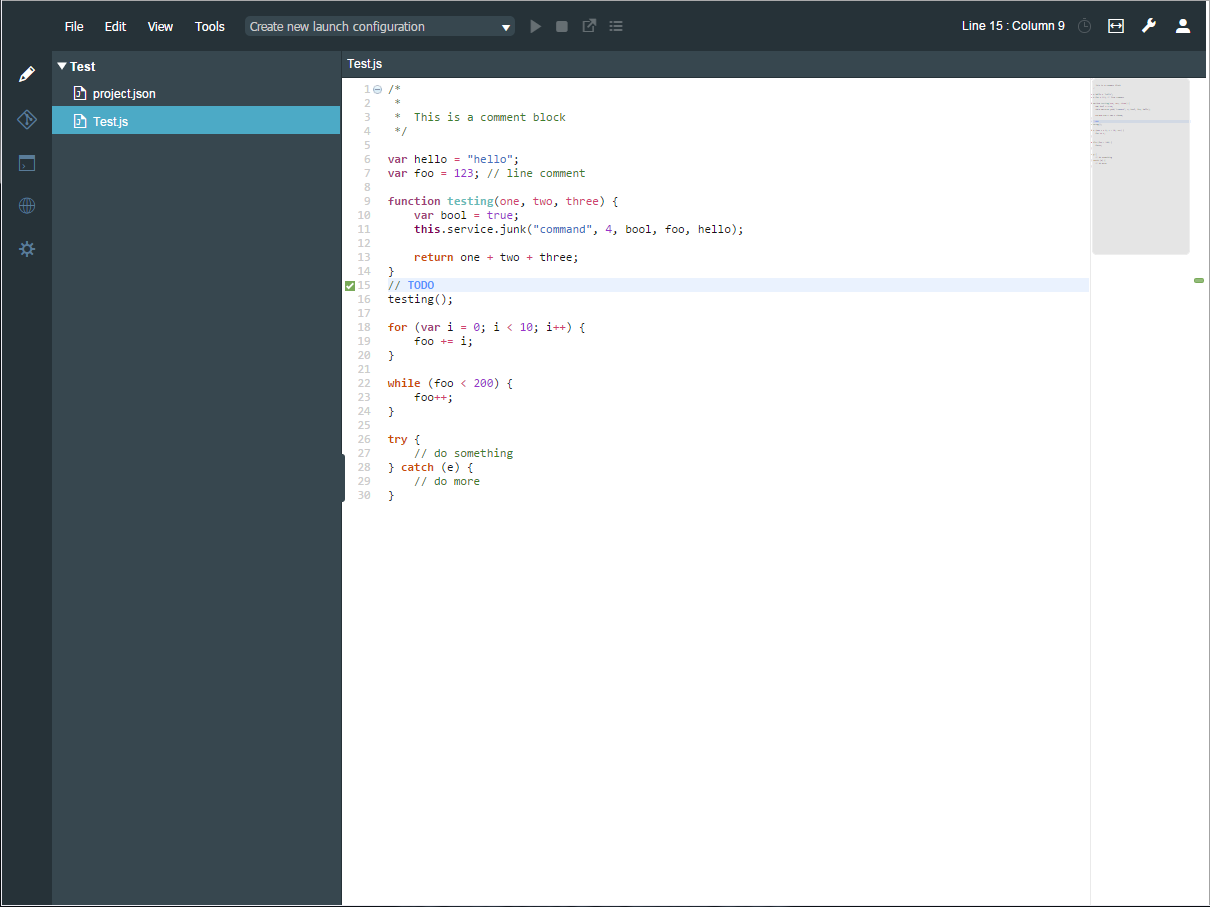
“Eclipse Orion” е Open Source[29] проект предимно разработен от “IBM”, който предлага онлайн редактор за JavaScript и други динамични езици (Ref. 2).

Основните му предимства са, че позволява лесна конфигурация и интеграция на други облачни услуги както и лесно писане и добавяне на Plugins[30] с нови функционалности. Eclipse Orion позволява лесен Cross-Site Workflow[31] и разнообразие от Integration Points[32]. Поради добрата си модуларизация и Loose Coupling[33], части от проекта, като редактора, могат лесно да се преизползват в други проекти.

Най-големите му недостатъци са, че липсват собствени Runtimes[5], APIs[34], Templates[22] и Wizards[23]. Заради липсата на тези основни звена, Eclipse Orion е по-подходящ за обединяваща точка на различни облачни услуги, а не толкова като самостоятелна среда, в която протичат всички стъпки по разработка, настройка, конфигурация и Monitoring[35] на облачни приложения.



Фиг. 1 Работен плот на проекти в Eclipse Orion



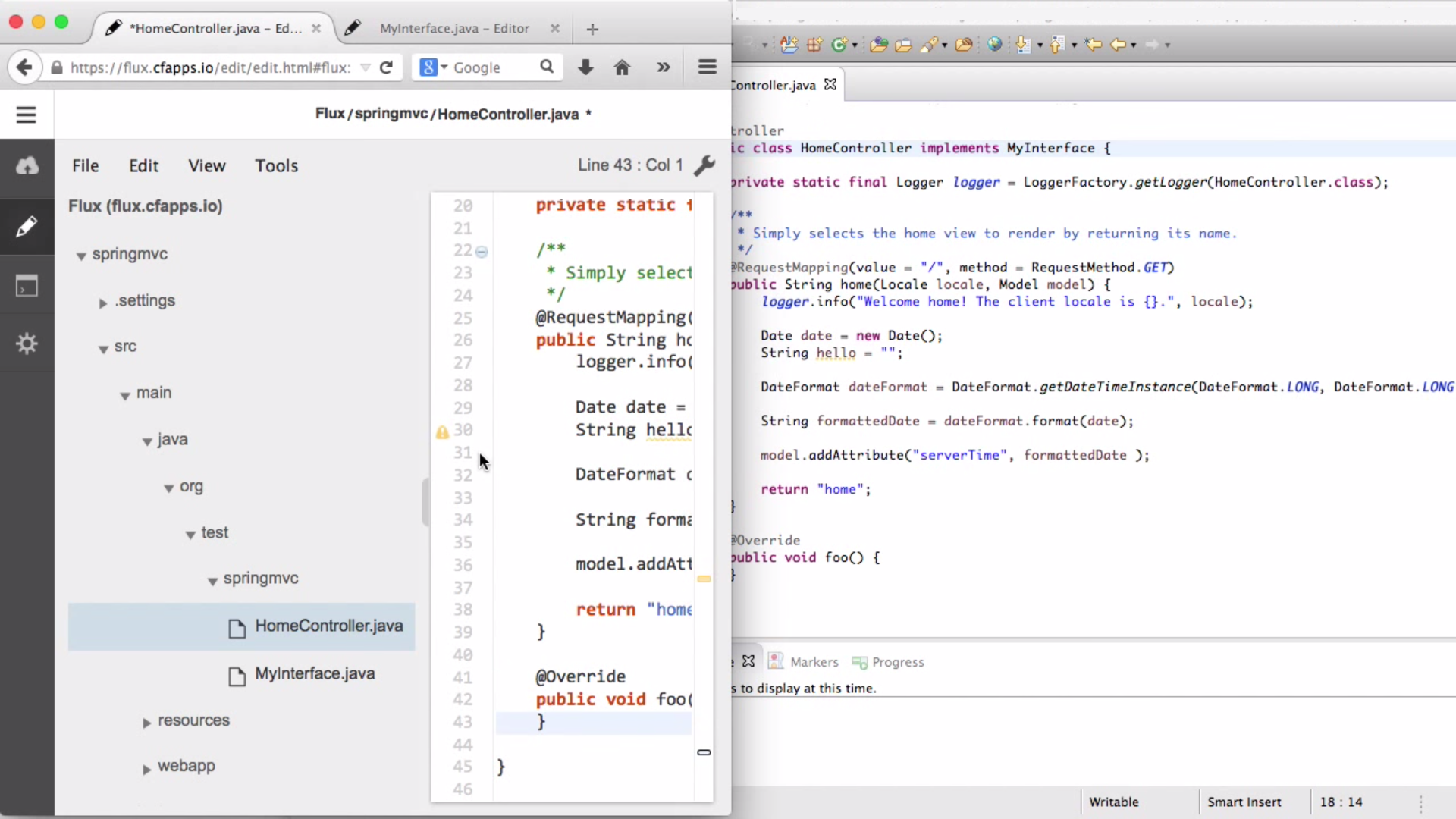
Фиг. 2 Редактор на изходен код на Eclipse Orion

Eclipse Flux

“Pivotal” е компанията, която е с основен принос за създаването на проекта “Eclipse Flux”. Мисията на Open Source[29] проекта е, да позволи плавен преход, за разработчиците, от Desktop IDE[19] към WebIDE[16] (Ref. 3, Ref. 3.1).

Предимството на този проект е, че позволява лесна синхронизация на проекти от Desktop IDE[19], като по този начин софтуерните разработчици могат да изберат, кои неща да правят с Desktop IDE[19] и кои с WebIDE[16]. Тази функционалност е възможна благодарение на архитектура за дистрибутирано изпращане/получаване на съобщения, липса на спицифика относно езика на програмиране и изпращане/получаване на съобщения в дистрибутираната среда в реално време. Други плюсове на проекта са използването на Micro Services[36] и интеграцията с Eclipse Orion (Ref. 3.2, Ref. 3.3).

Съществени недостатъци са, липсата на собствени Runtimes[5], APIs[34], Templates[22], Wizards[23] и други. Целта на проекта е да бъде интегриран в екосистема, която предлага липсващите функционалности.



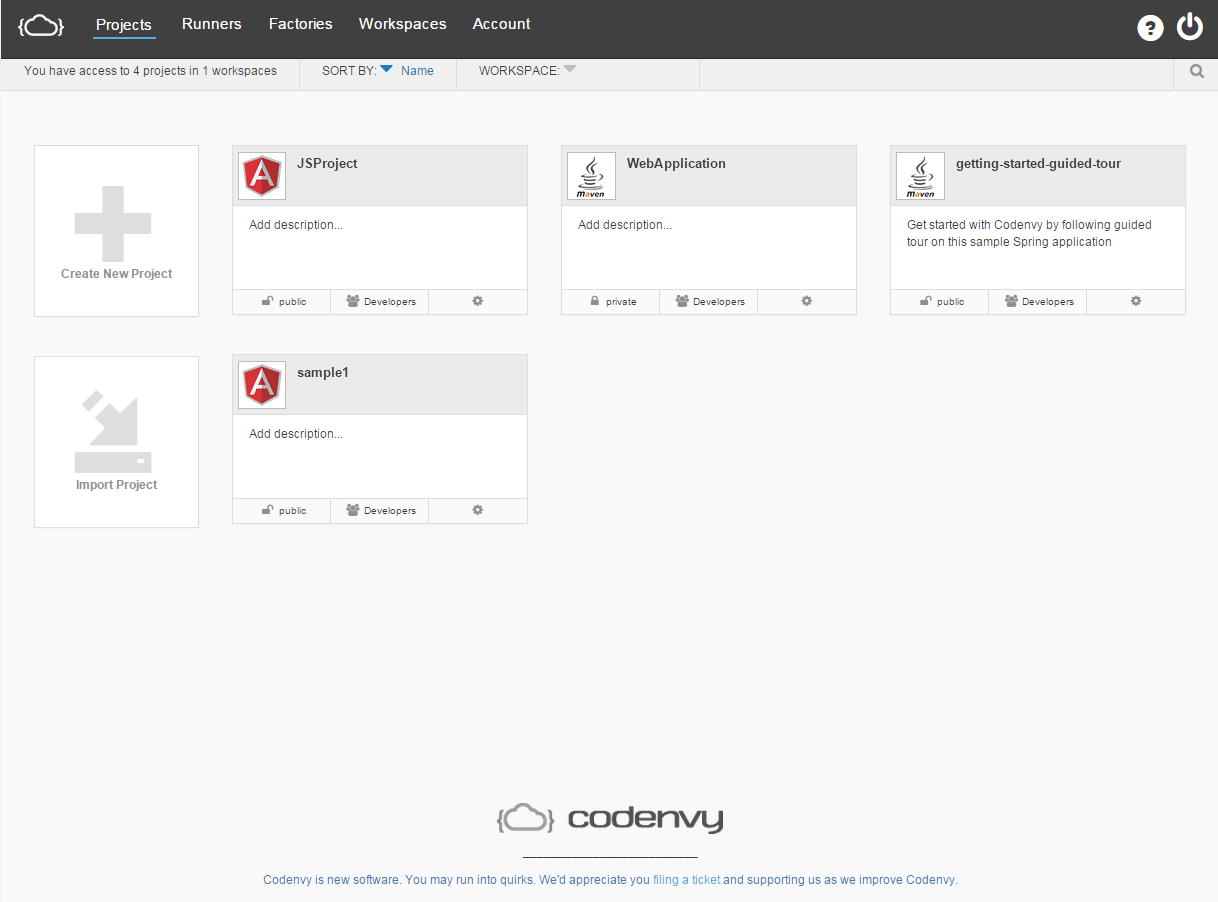
Фиг. 3 Синхронизиране на проект от Eclipse Desktop IDE в Eclipse Flux

Eclipse Che

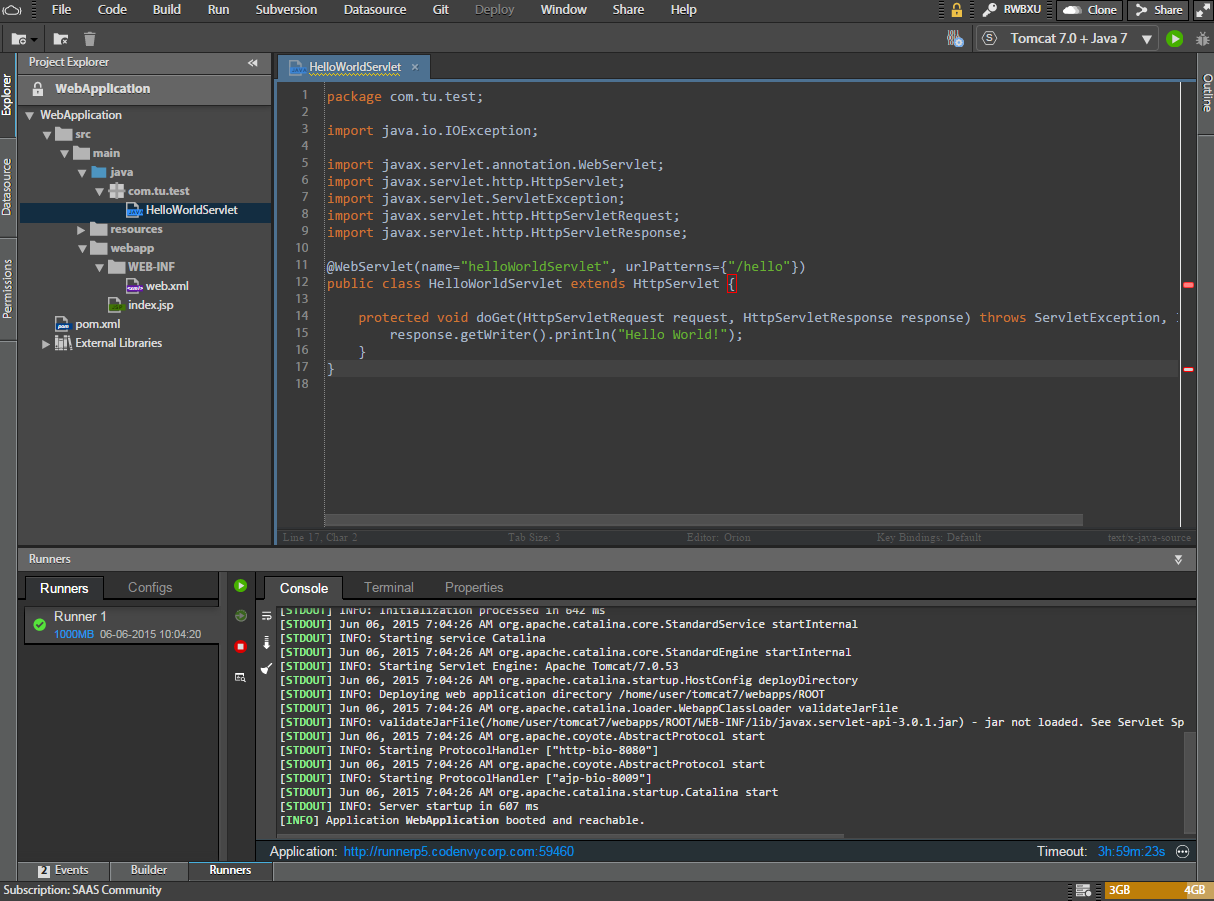
“Eclipse Che” е името на Open Source[29] проекта за “cloud IDE”(WebIDE[16]), предимно разработен от компанията “Codenvy” (Ref. 4, Ref. 4.1).

Голямото разнообразие от Runtimes[5] е освноното му предимство, някои от тях са: Java, Android, C/C++, JavaScript, PHP, Ruby и Go. Освен тях, Eclipse Che предлага и повечето известни Build Systems[7] като Maven[37], Ant[38], Grunt[39] и други. В основата на много добрата контейнерна модулоризация е поставен Docker[40] (Ref. 5). Той позволява на Eclipse Che да предостави много на брой и напълно изолирани един от друг, Runtime[5], Build System[7] и виртуални машини. Благодарение на всичко това, Eclipse Che успява да автоматизира процесите по настройка на работната среда и необходимите инструменти, като ги пренася в облака.

Освновен недостатък на това IDE[6] е, че стандартният модел на разработка е запазен. Той е същият както при On-Premises[26] софтуера, което носи със себе си нисък Turnaround Time[14]. Използването на този модел на работа налага наличието на фази като Build[12], Deploy[13], стартиране на отделна виртуална машина, стартиране на отдалечен сървър и други. Друг голям минус е липсата на Templates[22] и Wizards[23], които биха ускорили процеса на разработване.



Фиг. 4 Начална страница на Eclipse Che



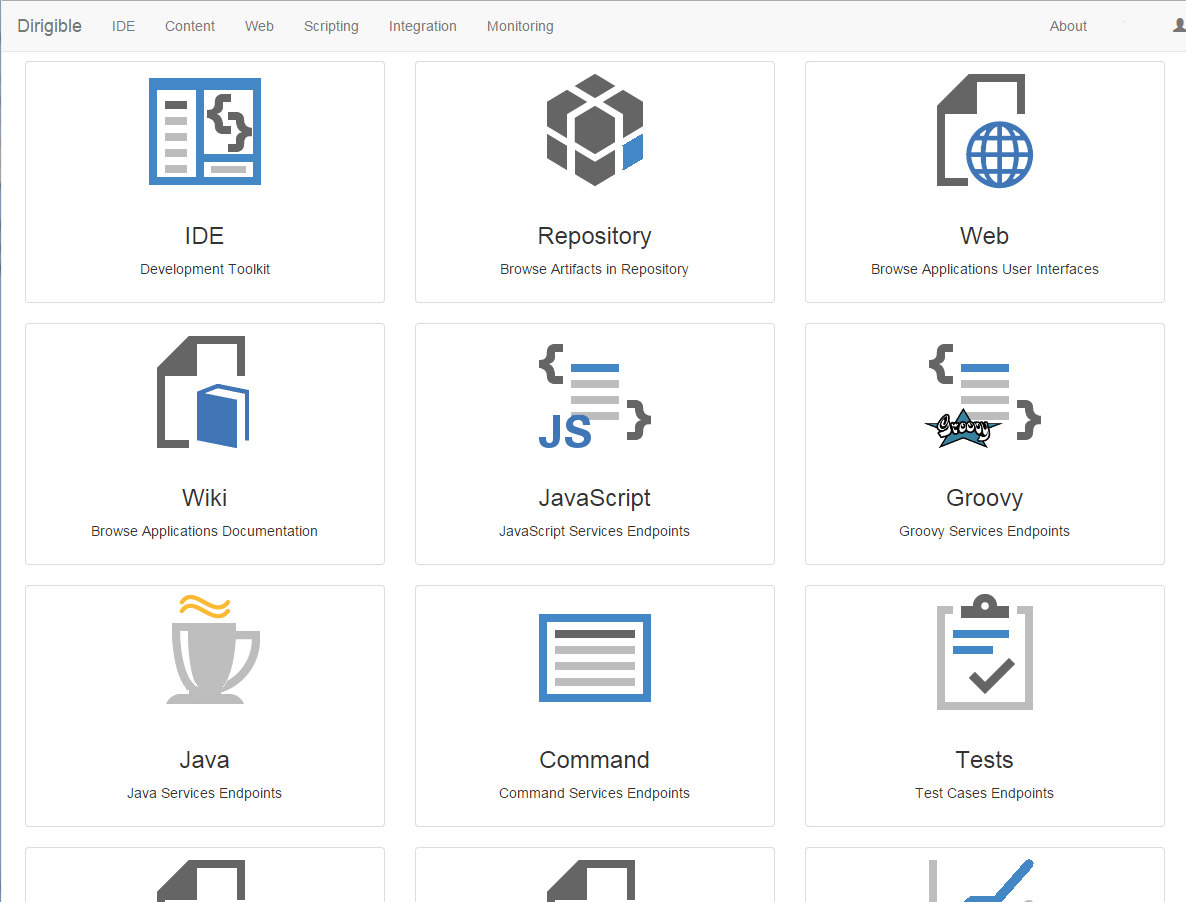
Фиг. 5 Работен плот на проект в Eclipse Che

Eclipse Dirigible

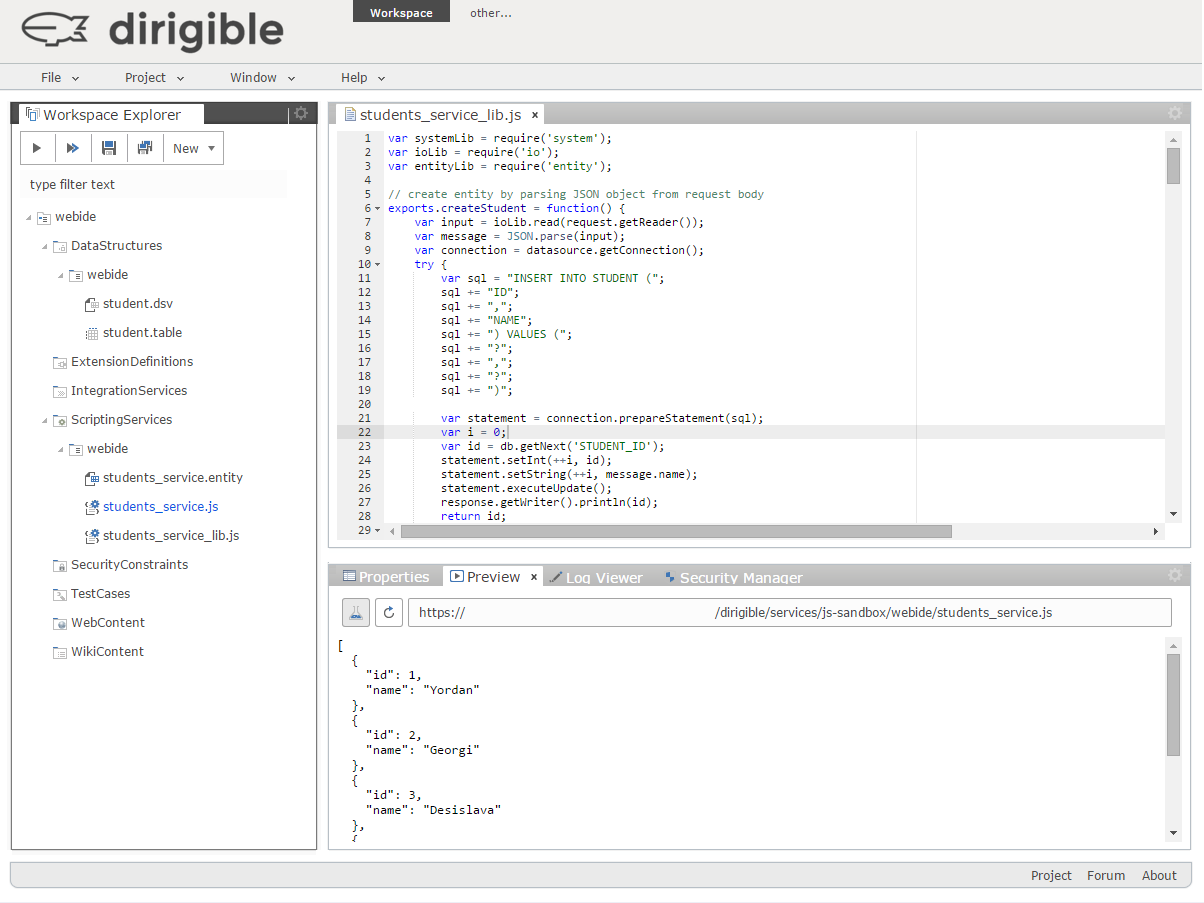
“SAP” е компанията с основен принос за създаването на Open Source[29] проекта “Eclipse Dirigible”. Проектът предлага IDEaaS[41] услуга (WebIDE[16]) за In-System Development[17] на облачни приложения (Ref. 6, Ref. 6.1).

Най-големите предимства на проекта са възможностите за In-System Development[17], RAD[18] чрез Templates[22] и Wizards[23], висок Turnaround Time[14], нисък Overhead[15], липса на време за Build[12] и Deploy[13], Runtime[5] за Java, JavaScript, Groovy, Ruby, както и лесното транспортиране на проекти от една система в друга. Друг плюс е графичния интерфейс, който до голяма степен прилича на Eclipse IDE, това значително улеснява разработчиците, които за първи път използват IDEaaS[41] услугата. Проектът предлага възможности за Offline[42] работа като инсталиранието на Plugins[30] за Eclipse IDE или работата с локален сървър. На последно място, но не и по-важност е лесното Extensibility[55] на проекта, чрез писане на Server-side[54] и Client-side[56] Plugins[30] (Ref. 6.2, Ref. 6.3, Ref. 6.4, Ref. 6.5, Ref. 6.6).

Няколко минуса към оценката на проекта са липсата на добър Code Completion[43] и Single Sign-On[44] опции за вписване чрез социални мрежи като Facebook, Github и други.



Фиг. 6 Начален екран на Eclipse Dirigible



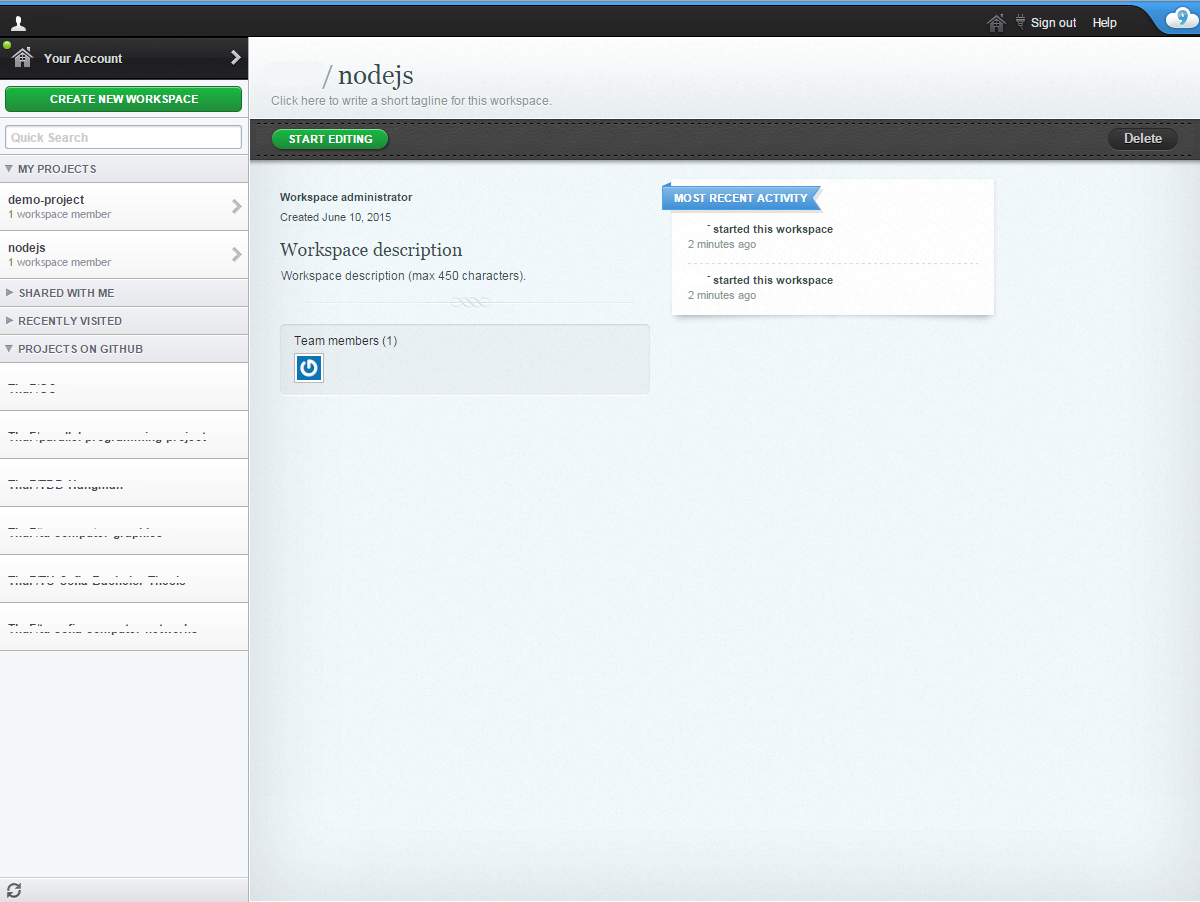
Фиг. 7 Работен плот на Eclipse Dirigible

Cloud9

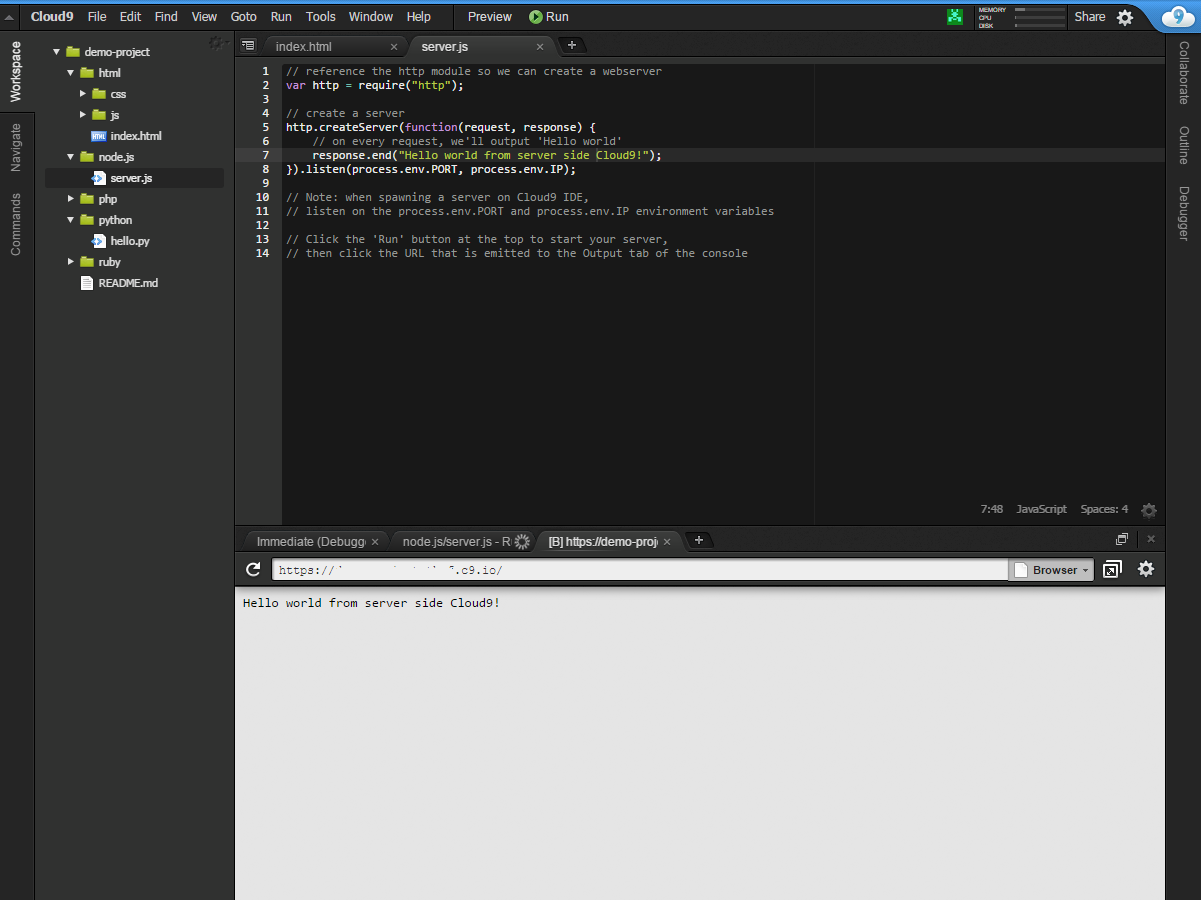
“Cloud9” е Open Source[29] WebIDE[16] проект с Non-Commercial License Agreement[45], голяма част от който е разработенa от компанията “Cloud 9 IDE, Inc” (Ref. 7, Ref. 7.1).

Някои от най-големите му плюсове са поддръжката на множество от Runtimes[5] като PHP, Ruby, Python, JavaScript, Go, възможността за едновременно редактиране на код от няколко разработчика, Templates[22] за повечето възможни артефакти на проекта и голямо разнообразие от опции за споделяне на изходния код на проекта, част от него или на готовото решение.

Съществен минус в оценката на проекта е, че не може да се използва безплатно за търговски цели (Commercial Use[46] (Ref. 7.2)) както и липсата на Wizards[23].



Фиг. 8 Tабло за управление на проекти в Cloud9



Фиг. 9 Работна среда за проект в Cloud9

Архитектура на среда за разработка в облака

За анализ на архитектурата на среда за разработка в облака е избран проектът Eclipse Dirigible. Архитектурата на проекта се състои от три основни компонента - IDE[6], Repository[58] и Runtime[5]. Освновните езиците за програмиране, които са използвани в проекта са: Java, JavaScript, SQL, HTML, JavaScript и CSS. Разделянето на проекти, в Eclipse Dirigible, по модули е постигнато благодарение на OSGi[60] Plugin[30] механизъм.

В IDE[6] частта на архитектурата се намират модулите, които отговарят за потребителския интерфейс. За Framework[8] за графичния интерфейс е избран Eclipse RAP[61]. Благодарение на Eclipse RAP[61] е възможно създаването на интерфейс, който разполага с добре познати компоненти от Eclipse Desktop IDE - папки, файлове, перспективи, Wizards[23] и други (Ref. 8). Това позволява създаването на UI[62], който е много познат за разработчиците ползващи Eclipse Desktop IDE. Някои основни модули са:

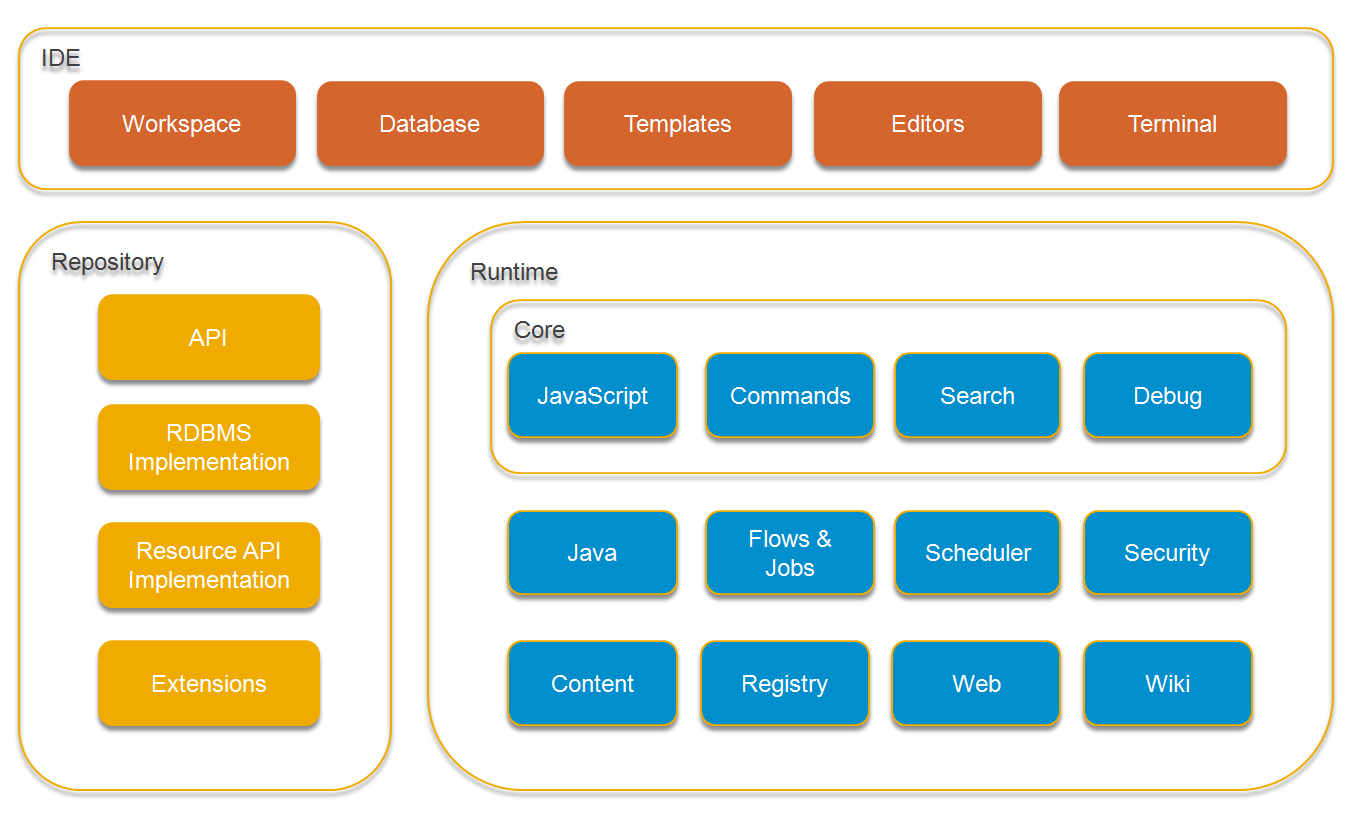
* Workspace - там се намират класовете, отговорни за графичното изобразяване на артефакти от работния плот.
* Database - в този модул се намира логиката за графична репрезентация на една от основните перспективи - “Database”. В перспективата се използват следните Views[64]: “Database View” и “SQL Console”.
* Templates - там се съдържат основни класове, Templates[22] и Wizards[23], които са отговорни за генерирането на изходен код на базата на определени правила. Този модул е базов за имплементирането на RAD[18] техниките в Eclipse Dirigible.
* Editors - модулът осигурява поддръжка на редактори за изходен код. Редакторите, които се поддържат към момента на писане на дипломната работа са: Ace[59], Eclipse Orion и WYSIWYG[63].

Repository е компонент от архитектурата, който служи за връзка между IDE[6] и Runtime[5]. Някой от основните му модули са:

* API[34] - в този модул са дефинирани интерфейсите за работа с Repository[65] обекта. Някои от тях са: IRepository, ICollection, IEntity, IResource и други.
* RDBMS[65] Implementation - там се намира имплементацията на Repository[65] API[34] за използване на база данни. В този модул се намират и SQL заявките към базата данни. Някои от класовете са: DBRepository, DBCollection, DBEntity, DBResource и други.
* Extensions - в този модул се съдържат общите за IDE[6] и Runtime[5] обекти както и поддръжката на диалекти за различни видове бази данни. Някои от класовете са: DebugSessionMetadata, DBUtils, DerbyDBSpecifier, PostgreSQLDBSpecifier, HANADBSpecifier, ConfigurationStore и други.

В Runtime[5] компонента се съдържат модулите, благодарение на които се изпълнява Server-side[54] логиката. Основен за този компонент е Mozilla Rhino Engine, който позволява изпълняването на Server-side[54] JavaScript логика (Ref. 14, Ref. 14.1). Някои основни модули за компонента са:

* JavaScript - обработва Server-side[54] логиката написана на JavaScript.
* Commands - обработва Server-side[54] логиката за изпълняване на команди към операционната система на виртуалната машина.
* Debug - Server-side[54] логика, която позволява Debugging[50] на Server-side[54] JavaScript.

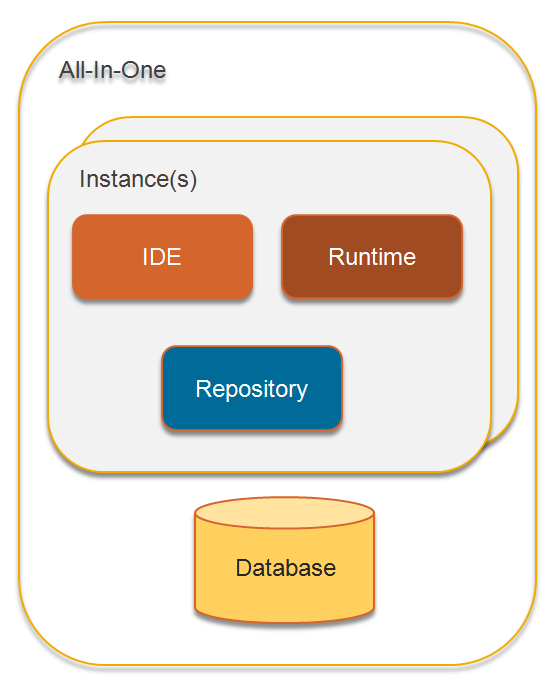
Фиг. 10 Основни компоненти на архитектурата на Eclipse Dirigible

В зависимост от конкретните нужди съществуват следните начини за Deploy[13] на Eclipse Dirigible:

* All-In-One
* Production
* RCP
* Multi-tenant

Deploy “All-In-One”

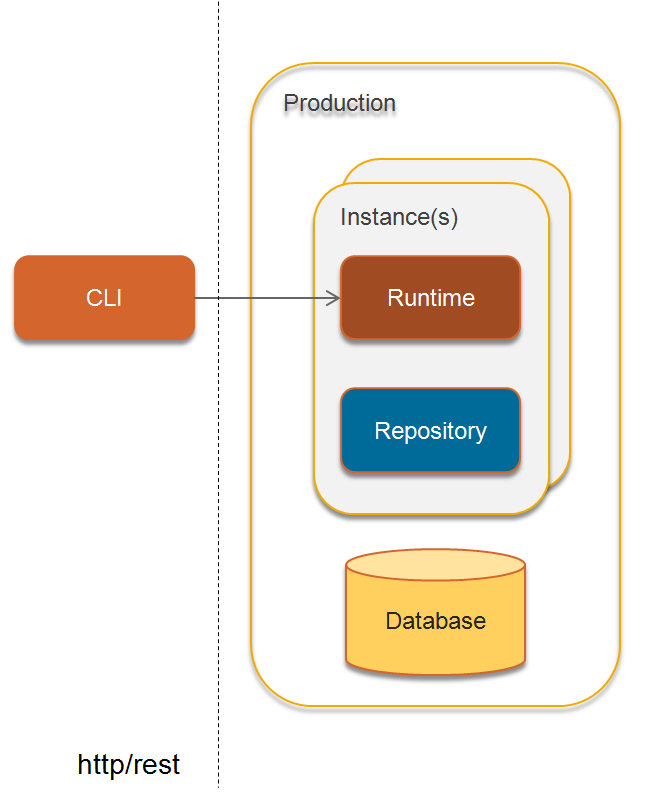
Варианта за Deploy[13] “All-In-One” предлага възможност във всяка инстанция (виртуална машина) да се съдържат трите основни компонента - IDE, Runtime и Repository. По този начин компонентите са свързани към една и съща база данни. Този модел на Deploy[13] е подходящ както за тестови Landscape[66], така и за Landscape[66] за разразботване.



Фиг. 11 Eclipse Dirigible - опция за Deploy “All-In-One”

Deploy “Production”

“Production” е опция за Deploy[13], в която разполагаме само с два от архитектурните компонента - Runtime и Repository. Този сценарий е подходящ за продуктивни Landscapes[66] - такива в които нямаме нужда от IDE, защото не се очаква в тези системи да пишем нов код. Новите версии на облачните приложения се транспортират чрез CLI[67] команди или чрез потребителския интерфейс на Registry[68].

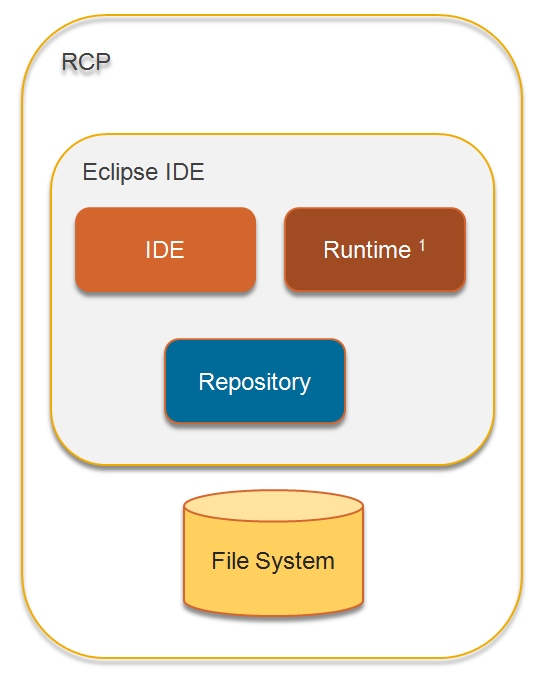


Фиг. 12 Eclipse Dirigible - опция за Deploy “Production”

Deploy “RCP”

Deploy[13] “RCP” е единият от вариантите за Local Development[57] с Eclipse Dirigible. Така основните архитектурните модули - IDE, Repository и Runtime1, стават част от Eclipse Desktop IDE. Този подход позволява преизползването на следните функционалности от Eclipse:

* JDT[69] - позволява използването на Code Completion[43] и Refactoring[49] за Java.
* Repository компонента използва стандартното Resource API на Eclipse Desktop IDE за работа с файловата система, вместо имплементацията на Resource API за работа с база данни.
* Runtime1 компонента има ограничени функции поради липсата на сървър.

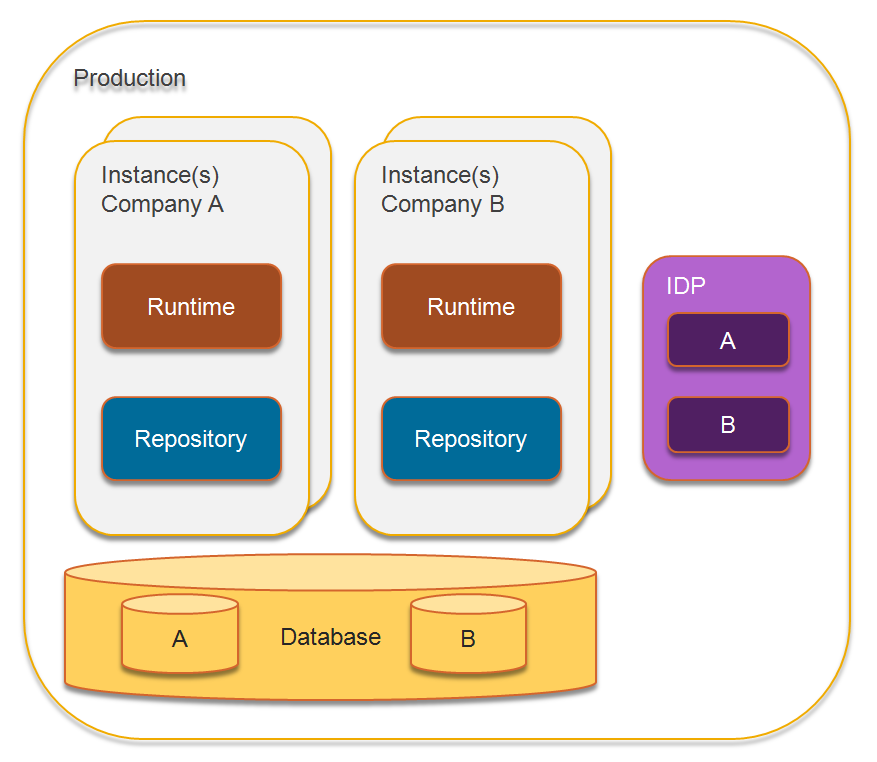


Фиг. 13 Eclipse Dirigible - опция за Deploy “RCP”

Deployment “Multi-tenant”

“Multi-tenant” е последният вариант за Deploy[13], която се допълва варианта “All-In-One” със следните неща:

* Могат да съществуват няколко инстанции едновременно.
* Всяка инстанция може да бъде конфигурирана, така че да използва собствен Identity Provider[70].
* Всяка инстанция може да бъде настроена, така че да използва отделна/изолирана схема в базата данни.



Фиг. 14 Eclipse Dirigible - опция за Deploy “Multi-tenant”

Извод

Таблица 1: Критерийно оценяване на Orion, Flux, Che, Dirigible и Cloud9

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Критерии | Orion | Flux | Che | Dirigible | Cloud9 |
| Single Sign-On[44] with social networks | ✔ | ╳ | ✔ | ╳ | ✔ |
| Code Completion[48] | ╳ | ✔ | ✔ | ╳ | ✔ |
| Refactoring[49] | ╳ | ╳ | ╳ | ╳ | ╳ |
| Debugging[50] | ╳ | ╳ | ✔ | ✔ | ✔ |
| Import/Export[51] | ✔ | ╳ | ✔ | ✔ | ✔ |
| Static Content based Code Completion[47] | ✔ | ✔ | ✔ | ✔ | ✔ |
| In-System Development Lifecycle[17] | ╳ | ╳ | ╳ | ✔ | ╳ |
| Samples[52] | ╳ | ╳ | ✔ | ✔ | ✔ |
| Templates[22] | ╳ | ╳ | ╳ | ✔ | ✔ |
| Wizards[23] | ╳ | ╳ | ╳ | ✔ | ╳ |
| Terminal[53] | ╳ | ╳ | ✔ | ✔ | ✔ |
| Server-side[54] JavaScript Runtime[5] | ╳ | ╳ | ✔ | ✔ | ✔ |
| Server-side[54] Java Runtime[5] | ╳ | ╳ | ✔ | ✔ | ✔ |
| Other Runtimes[5] | ╳ | ╳ | ✔ | ✔ | ✔ |
| Source Control System[10] | ✔ | ╳ | ✔ | ✔ | ✔ |
| Server-side[54] Extensibility[55] | ╳ | ✔ | ✔ | ✔ | ✔ |
| Client-side[56] Extensibility[55] | ✔ | ╳ | ✔ | ✔ | ✔ |
| Micro Services[36] | ╳ | ✔ | ╳ | ╳ | ╳ |
| Local Development[57] | ✔ | ✔ | ✔ | ✔ | ✔ |
| Commercial Use[46] | ✔ | ✔ | ✔ | ✔ | ╳ |
| Общо | 7 | 6 | 15 | 16 | 15 |

От направения сравнителен анализ на съществуващите решения и от таблицата за критерийно оценяване, можем да заключим, че най-подходящото решение за целите на дипломната работа е Eclipse Dirigible. Проектът събира най-много точки като отговаря на 16 от общо 20 изисквания. Някой от най-съществените предимства на Eclipse Dirigible пред останалите проекти са In-System Development[17], RAD[18] чрез Samples[54], Templates[22] и Wizards[23].

Проектиране и реализиране на модул за “In-memory Dynamic Java Compilation”

От версия 1.6 на Java (JDK 6) се въвеждат инструменти за динамична компилация на код. Товa се постига благодарение на пакета javax.tools. В него се намират необходимите интерфейси и класове за компилиране на код като JavaCompiler, JavaFileManager, ToolProvider и други (Ref. 9).

Съчетанието между динамичното компилиране и In-System Development[16] парадигмата, отключва възможността за “In-memory Dynamic Java Compilation” (Ref. 10). Ако към тях добавим и идеята за WebIDE[16], резултатът ще бъде възможност за динамично компилиране, зареждане и изпълнение на Java код, директно върху отдалечения сървър и виртуалната машина. Този модел на разработване се отличава значително от стандартния модел за разработване на Java базирани уеб приложения. Предимствата на новия модел пред стария са: липса на времена за Build[12], Deploy[13] и рестартиране на Server[9]. Всичко това води до значително увеличаване на Turnaround Time[14] за писане на Java уеб приложения. “In-memory Dynamic Java Compilation” модула е част от архитектурния компонент Runtime[5] на Eclipse Dirigible. Кодът на целият модул може да бъде намерен на: <https://github.com/eclipse/dirigible/tree/master/org.eclipse.dirigible/org.eclipse.dirigible.parent/runtime/org.eclipse.dirigible.runtime.java>

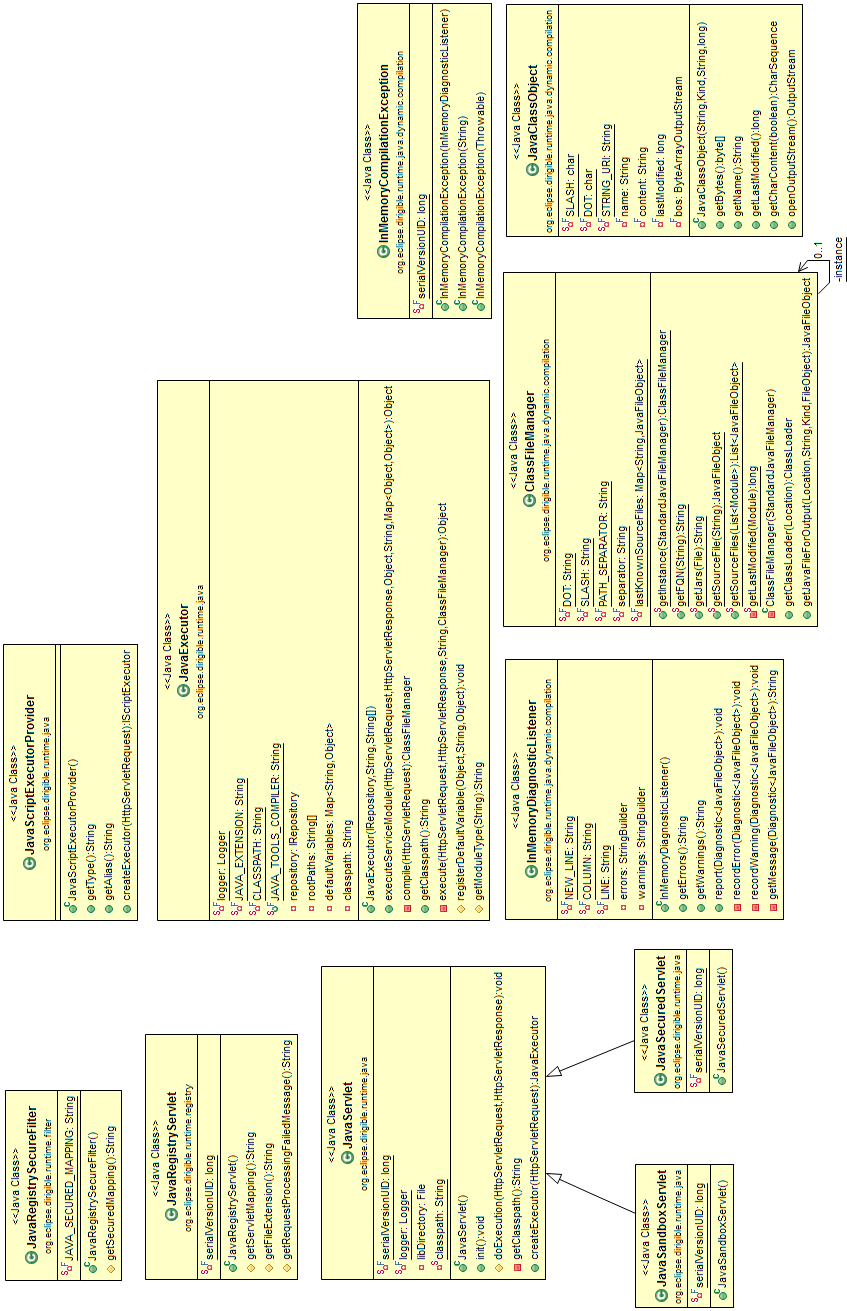
OSGi[60] модула за “In-memory Dynamic Java Compilation” е интегриран с другите части от Eclipse Dirigible чрез следните Extensions[71]:

Списък на Extensions[71] към Extension Point[72] “org.eclipse.equinox.http.registry.servlets”:

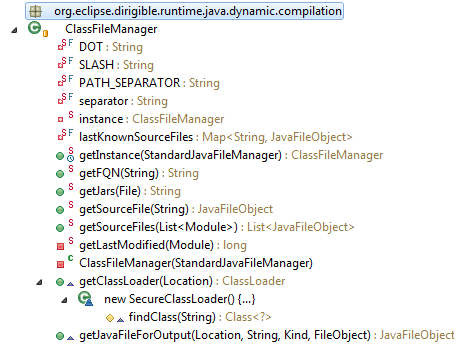
* “JavaServlet” за уеб ресурси в “/java”
* “JavaSecuredServlet” за уеб ресурси в “/java-secured”
* “JavaSandboxServlet” за уеб ресурси в “/java-sandbox”
* “JavaRegistryServlet” за уеб ресурси в “/scripting/java”

Списък на Extensions[71] към Extension Point[72] “org.eclipse.equinox.http.registry.filters”:

* “RegistrySecureRolesFilter” за уеб ресурси в “/java-secured/\*”
* “RegistrySecureRolesFilter” за уеб ресурси в “/java-sandbox/\*”
* “RegistrySecureRolesFilter” за уеб ресурси в “/java/\*”



Фиг. 15 ООП модел на модул за “In-memory Dynamic Java Compilation” в Eclipse Dirigible

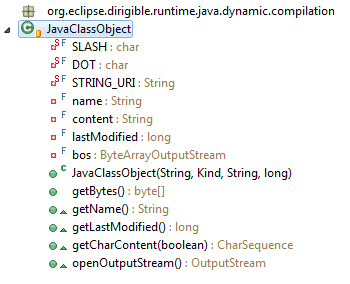


Фиг. 16 Структура на класа “ClassFileManager”

Класът “ClassFileManager” е основен за модула. В него се извършва логиката за прочитане и зареждане на Java библиотеки, файлове с изходен код и класове от оперативната памет. Някои от основните методи са:

* getJars() - статичен метод, който открива абсолютния път до Java библиотеки в дадена папка. Резултатът от метода е String съдържащ пътищата разделени със запетайки.
* getSourceFiles() - статичен метод, който връща списък със записаните в памметта Java файлове. Търсенето на файлове става по модул, от който е дошло запитването. Ако не се открие такъв файл или файлът е бил обновен, спрямо последното извикване на метода, методът зарежда наново съдържанието на файла в оперативната памет и го добавя в списъка с последно известните Java файлове.
* getClassLoader() - статичен метод, който създава ClassLoader, който създава инстанции на класовете заредени в RAM паметта.

Част от изходния код на класа “ClassFileManager” може да бъде видян в “Приложение” - App. 1.

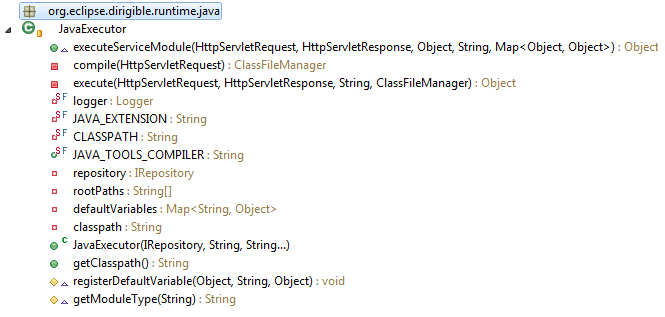


Фиг. 17 Структура на класа “JavaClassObject”

Друг ключов клас е “JavaClassObject”. В него се намира имплементацията за представянето на компилиран Java файл в паметта. Някои от основните методи са:

* getBytes() - връща масив от байтове, които представляват съдържанието на компилирания клас.
* getName() - връща името на файла.
* openOutputStream() - отворя поток за запис във файла. Използва се за да се запишат байтовете на компилирания клас в паметта.

Изходния код на класа може да бъде видян в “Приложение” - App. 2.

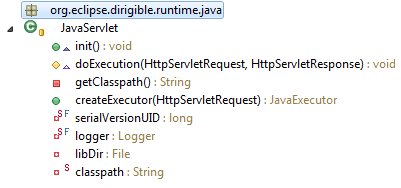


Фиг. 18 Структура на класа “JavaExecutor”

В класа “JavaExecutor” се намира логиката, която компилира и изпълнява записаните в паметта Java файлове. Някои от основните методи са:

* compile() - методът се грижи за компилирането на файловете с изходен код написан на Java. Списъкът с файловете се извлича чрез класа ClassFileManager. Следва компилиране на подадените файлове и проверка на резултата от компилацията.
* execute() - методът изпълнява компилираните класове като извиква методът “service” на инстанция на обект от този клас. По този начин изпълнимите Java файлове, записани в оперативната памет, до голяма степен приличат на Servlets (Ref. 11, Ref. 11.1). Методът “service”, на компилирания изпълним Java файл, получава три аргумента от следните типове - javax.servlet.http.HttpServletRequest, javax.servlet.http.HttpServletResponse и java.util.Map.
* executeServiceModule() - първата стъпка от изпълнението на метода е регистрирането в контекста на изпълнимия Java файл, чрез обект от тип java.util.Map, API[34] на Eclipse Dirigible (Ref. 12). Следват изиквания към методите compile() и execute().

Част от изходния код на класа може да бъде видян в “Приложение” - App. 3.



Фиг. 19 Структура на класа “JavaServlet”

Класа “JavaServlet” обработва заявки за изпълнение на Java файлове. Някои от основните методи са:

* init() - в този метод се определя къде се намира папката, във виртуалната машина, с необходимите Java библиотеки.
* getClasspath() - този метод връща резултата от извикването на статичния метод getJars(), с параметър местоположението на папката с необходимите Java библиотеки, на класа “ClassFileManager”.
* doExecution() - методът създава инстанция на класа “JavaExecutor” и извиква метода му executeServiceModule().

Част от изходния код на класа може да бъде видян в “Приложение” - App. 4.

Проектиране и реализиране на модул за “Git Integration”

Git е Source Control System[10], която се използва в компании и проекти като Google, Facebook, Microsoft, Twitter, Apache, Eclipse, Linux, Android и други (Ref. 13). JGit е библиотека написана на Java, която предлага Java API[34] за повечето функционалности на Git (Ref. 13.1). JGit се използва в проекти като EGit (Plugin[30] за Eclipse Desktop IDE), Gerrit Code Review, NBGit (Plugin[30] за NetBeans) и други. “Git Integration” модула е част от IDE[6] компонента на архутектурата на Eclipse Dirigible. Кодът на целият модул може да бъде намерен на: <https://github.com/eclipse/dirigible/tree/master/org.eclipse.dirigible/org.eclipse.dirigible.parent/ide/org.eclipse.dirigible.ide.jgit.connector>

OSGi[60] модула за “Git Integration” е интегриран с другите части от Eclipse Dirigible чрез следните Extensions[71]:

Списък на Extensions[71] към Extension Point[72] “org.eclipse.ui.commands”:

* Регистрация на командата “Clone” с име “org.eclipse.dirigible.ide.jgit.command.CloneCommand”.
* Регистрация на командата “Push” с име “org.eclipse.dirigible.ide.jgit.command.PushCommand”.
* Регистрация на командата “Pull” с име “org.eclipse.dirigible.ide.jgit.command.PullCommand”.
* Регистрация на командата “Reset” с име “org.eclipse.dirigible.ide.jgit.command.ResetCommand”.
* Регистрация на командата “Share” с име “org.eclipse.dirigible.ide.jgit.command.ShareCommand”.

Списък на Extensions[71] към Extension Point[72] “org.eclipse.ui.menus”:

* Надпис “Clone” в менюто “org.eclipse.dirigible.ide.workspace.ui.team” с команда “org.eclipse.dirigible.ide.jgit.command.CloneCommand”.
* Надпис “Push” в менюто “org.eclipse.dirigible.ide.workspace.ui.team” с команда “org.eclipse.dirigible.ide.jgit.command.PushCommand”.
* Надпис “Pull” в менюто “org.eclipse.dirigible.ide.workspace.ui.team” с команда “org.eclipse.dirigible.ide.jgit.command.PullCommand”.
* Надпис “Reset” в менюто “org.eclipse.dirigible.ide.workspace.ui.team” с команда “org.eclipse.dirigible.ide.jgit.command.ResetCommand”.
* Надпис “Share” в менюто “org.eclipse.dirigible.ide.workspace.ui.team” с команда “org.eclipse.dirigible.ide.jgit.command.ShareCommand”.

Списък на Extensions[71] към Extension Point[72] “org.eclipse.ui.commandImages”:

* Регистрирано изображение, за командата “org.eclipse.dirigible.ide.jgit.command.CloneCommand”, с относителен път “resources/icons/clone.png” в модула “org.eclipse.dirigible.ide.jgit.connector”.
* Регистрирано изображение, за командата “org.eclipse.dirigible.ide.jgit.command.PushCommand”, с относителен път “resources/icons/push.png” в модула “org.eclipse.dirigible.ide.jgit.connector”.
* Регистрирано изображение, за командата “org.eclipse.dirigible.ide.jgit.command.PullCommand”, с относителен път “resources/icons/pull.png”в модула “org.eclipse.dirigible.ide.jgit.connector”.
* Регистрирано изображение, за командата “org.eclipse.dirigible.ide.jgit.command.ResetCommand”, с относителен път “resources/icons/reset.png” в модула “org.eclipse.dirigible.ide.jgit.connector”.
* Регистрирано изображение, за командата “org.eclipse.dirigible.ide.jgit.command.ShareCommand”, с относителен път “resources/icons/share.png” в модула “org.eclipse.dirigible.ide.jgit.connector”.

Списък на Extensions[71] към Extension Point[72] “org.eclipse.ui.handlers”:

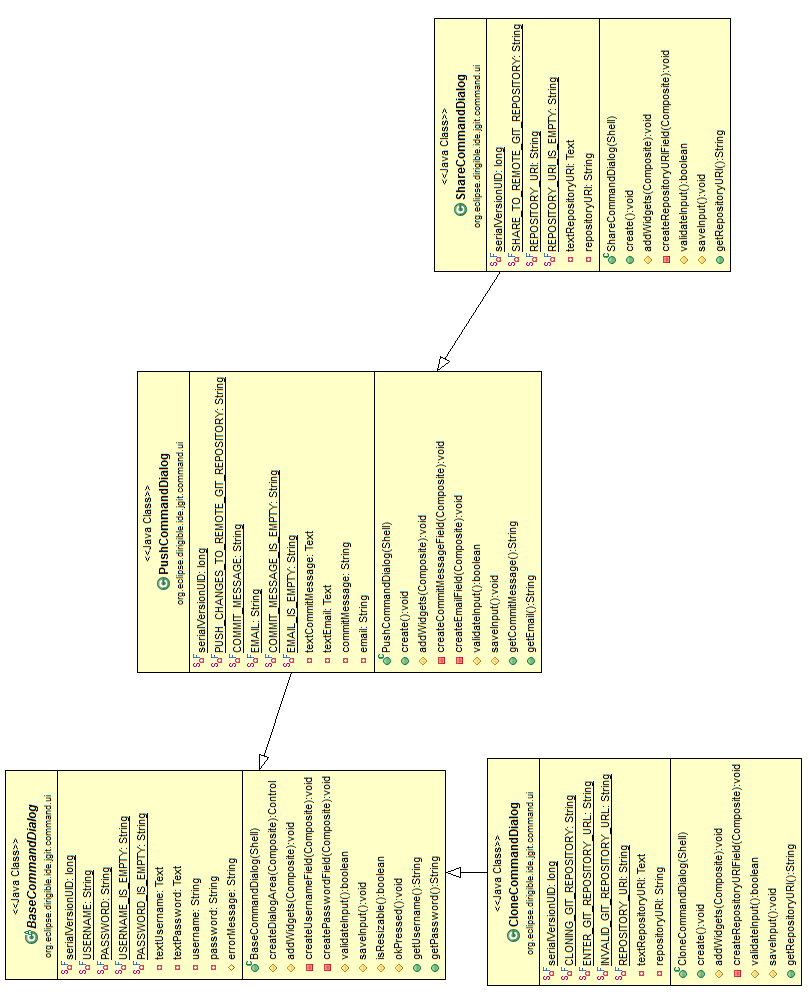
* “CloneCommandHandler” - класът който имплементира командата за “Clone” - “org.eclipse.dirigible.ide.jgit.command.CloneCommand”.
* “PushCommandHandler” - класът който имплементира командата за “Push” - “org.eclipse.dirigible.ide.jgit.command.PushCommand”.
* “PullCommandHandler” - класът който имплементира командата за “Pull” - “org.eclipse.dirigible.ide.jgit.command.PullCommand”.
* “ResetCommandHandler” - класът който имплементира командата за “Reset” - “org.eclipse.dirigible.ide.jgit.command.ResetCommand”.
* “ShareCommandHandler” - класът който имплементира командата за “Share” - “org.eclipse.dirigible.ide.jgit.command.ShareCommand”.

Списък на Extensions[71] към Extension Point[72] “org.eclipse.core.expressions.propertyTesters”:

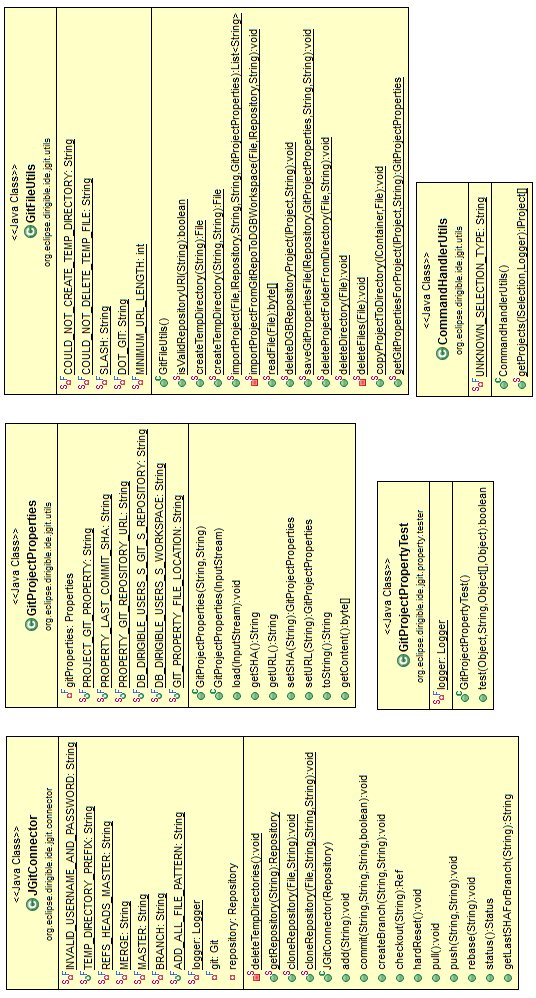
* “GitProjectPropertyTest” - класът който имплементира логиката за кои команди (Clone, Push, Pull, Reset и Share) е валиден ресурс от тип “org.eclipse.core.resources.IProject”.



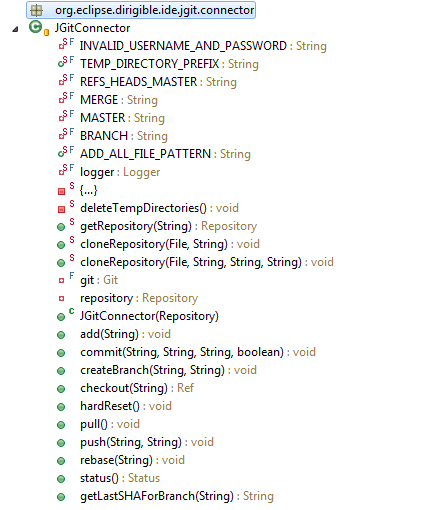
Фиг. 20 ООП модел на модул за “Git Integration” в Eclipse Dirigible част 1



Фиг. 21 ООП модел на модул за “Git Integration” в Eclipse Dirigible част 2



Фиг. 22 ООП модел на модул за “Git Integration” в Eclipse Dirigible част 3



Фиг. 23 Структура на класа “JGitConnector”

Класът “JGitConnector” е основен за “Git Integration” модула. В него се съдържа логиката за работа с Git Repository[58] и изпълняване на основни команди като “Commit”, “Push”, “Pull”, “Rebase” и други. Някои от основните методи са:

* deleteTempDirectories() - методът изтрива папките в “%temp%” директорията на операционната система, използвани по време на работа с JGitConnector класа.
* cloneRepository() - клонира Git Repository[58] върху файловата система на виртуалната машина (Ref. 13.2).
* commit() - записва направените промени, върху следените файлове, в Git Repository[58] (Ref. 13.3).
* pull() - изтегля и интегрира в локално Git Repository[58], промени направени в отдалечено Git Repository[58] (Ref. 13.4).
* push() - изпраща на направените и запазените промени от локално Git Repository[58] към отдалечено Git Repository[58] (Ref. 13.5).

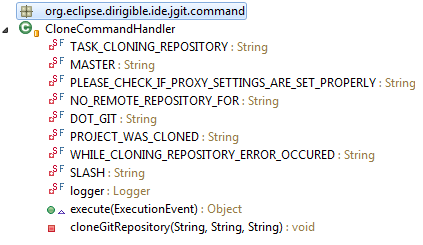
Част от изходния код на класа “JGitConnector” може да бъде видян в “Приложение” - App. 5.

|  |  |
| --- | --- |
| BaseCommandDialog - Outline.png  Фиг. 24 Структура на класа “BaseCommandDialog” | CloneCommandDialog - UI.png  Фиг. 25 “CloneCommandDialog” - графичен интерфейс |

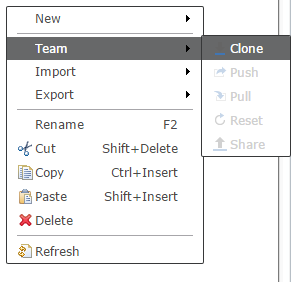
“BaseCommandDialog” е базовият клас за диалог с потребителите на Git командите. За създаване на UI[62] е използван Eclipse RAP[61]. Наследници на “BaseCommandDialog” класа са: “CloneCommandDialog”, “PushCommandDialog” и “ShareCommandDialog”. Някои от основните методи са:

* createDialogArea() - създава потребителския интефейс на прозореца за диалог. Обръща се към метода addWidgets() за добавяне на елементи в прозореца.
* addWidgets() - добавя надписи и текстови полета за въвеждане на потребителско име и парола.
* validateInput() - методът проверява дали въведените от потребителя данни, в полетата за потребителско име и парола, са валидни и връща Boolean резултат.
* okPressed() - извиква се при натискане на бутона “OK” от потребителя. Методът се обръща към validateInput() за валидиране на входните данни. Ако резултатът от валидацията е положителен, данните се записват в полетата за потребителско име и парола. Ако въведените данни не са валидни се изписва съобщение за грешка.

Част от изходния код на класа “BaseCommandDialog” може да бъде видян в “Приложение” - App. 6.



Фиг. 26 Структура на класа “CloneCommandHandler”



Фиг.27 Събитие от потребителския интерфейс, което задейства “CloneCommandHandler”

Кодът на класа “CloneCommandHandler” обработва събитието, което настъпва при натискане на опцията от контекстното меню за Team->Clone. Основни методи са:

* execute() - методът се изпълнява при постъпване на събитието активиращо класа. Методът създава нова инстанция от класа “CloneCommandDialog” (диалогов прозорец за Git командата “Clone” (Ref. 13.2)) и го показва на потребителя.
* cloneGitRepository() - в този метод се намира логиката за изпълняване na Git командата “Clone” (Ref. 13.2). Методът използва класа “JGitConnector” за да изпълни операциите по създаване на временна директория на операционната система, създаване на локално Repository[58] и обновяване на потребителския интерфейс след края на опрацията.

Част от изходния код на класа “CloneCommandDialog” може да бъде видян в “Приложение” - App. 7.

*Проектиране и реализиране на модул за “Debugger”*

Eclipse Dirigible използва Mozilla Rhino Engine за Server-side[54] JavaScript (Ref. 14, Ref. 14.1, Ref. 14.2). Rhino е написан на Java, което позволява лесното интегриране в Runtime[5] компонента на архитектурата на Eclipse Dirigible. Този Server-side[54] JavaScript Framework[8] позволява зареждането, интерпетирането, изпълнението, и Debugging[50] на JavaScript файлове (Ref. 14.3). На база на Debugging[50] функционалността, предлагана от Mozilla Rhino, е създаден “Debugger” модул в Eclipse Dirigible. Части от този модул се съдържат в трите архитектурни компоненти (IDE[6], Repository[58] и Runtime[5]), което позволява връзката между потребителския интерфейс и Debugging[50]. Кодът на частите на модулa могат да бъде намерен на:

* <https://github.com/eclipse/dirigible/tree/master/org.eclipse.dirigible/org.eclipse.dirigible.parent/ide/org.eclipse.dirigible.ide.debug.model>
* <https://github.com/eclipse/dirigible/tree/master/org.eclipse.dirigible/org.eclipse.dirigible.parent/ide/org.eclipse.dirigible.ide.debug.ui>
* <https://github.com/eclipse/dirigible/tree/master/org.eclipse.dirigible/org.eclipse.dirigible.parent/repository/org.eclipse.dirigible.repository.ext/src/org/eclipse/dirigible/repository/ext/debug>
* <https://github.com/eclipse/dirigible/tree/master/org.eclipse.dirigible/org.eclipse.dirigible.parent/runtime/org.eclipse.dirigible.runtime.core/src/org/eclipse/dirigible/runtime/js/debug>

OSGi[60] модулите за “Debugger” са интегрирани с другите части от Eclipse Dirigible чрез следните Extensions[71]:

Extension[71] към Extension Point[72] “org.eclipse.ui.perspectives”:

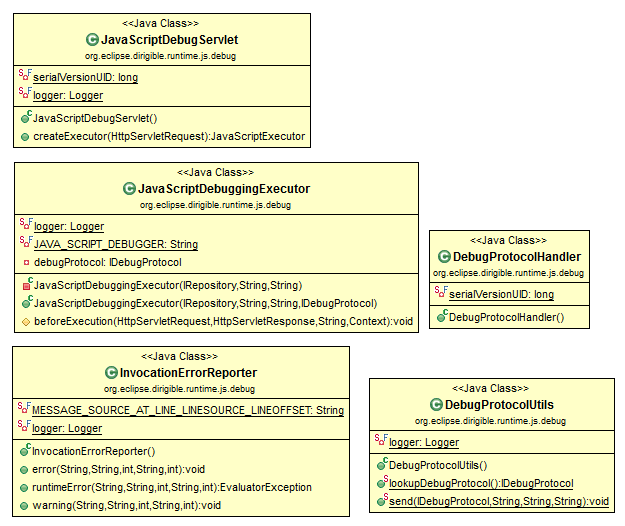
* Debug персепктива, която е имплементирана в класа “org.eclipse.dirigible.ide.debug.ui.DebugPerspective”

Extension[71] към Extension Point[72] “org.eclipse.ui.views”:

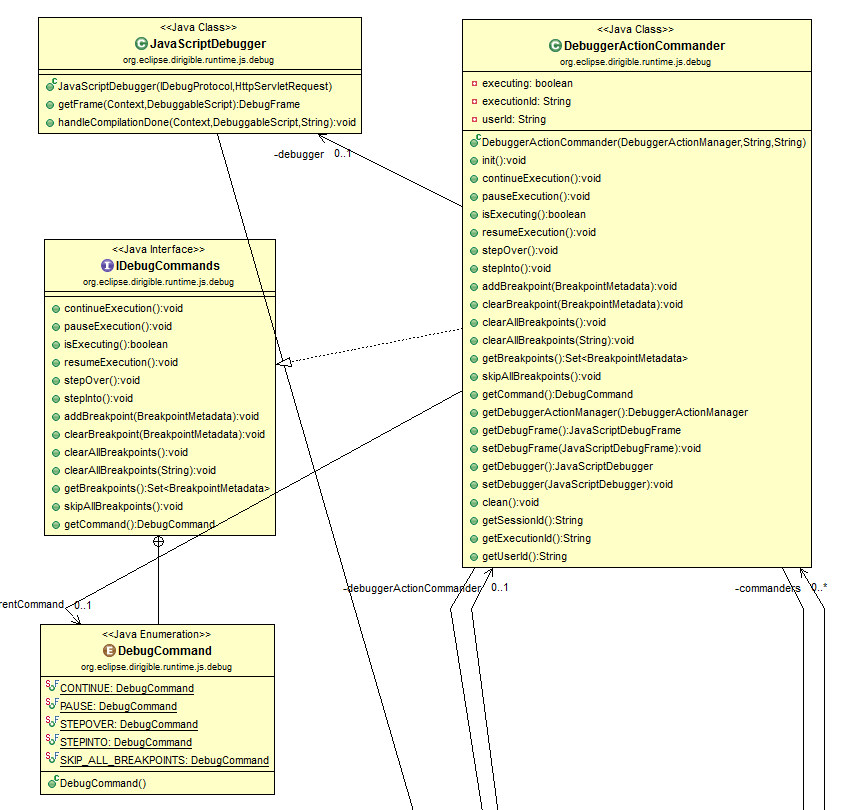
* Debug View[64], което е имплементирано в класа “org.eclipse.dirigible.ide.debug.ui.DebugView”

Extension[71] към Extension Point[72] “org.eclipse.equinox.http.registry.servlets”:

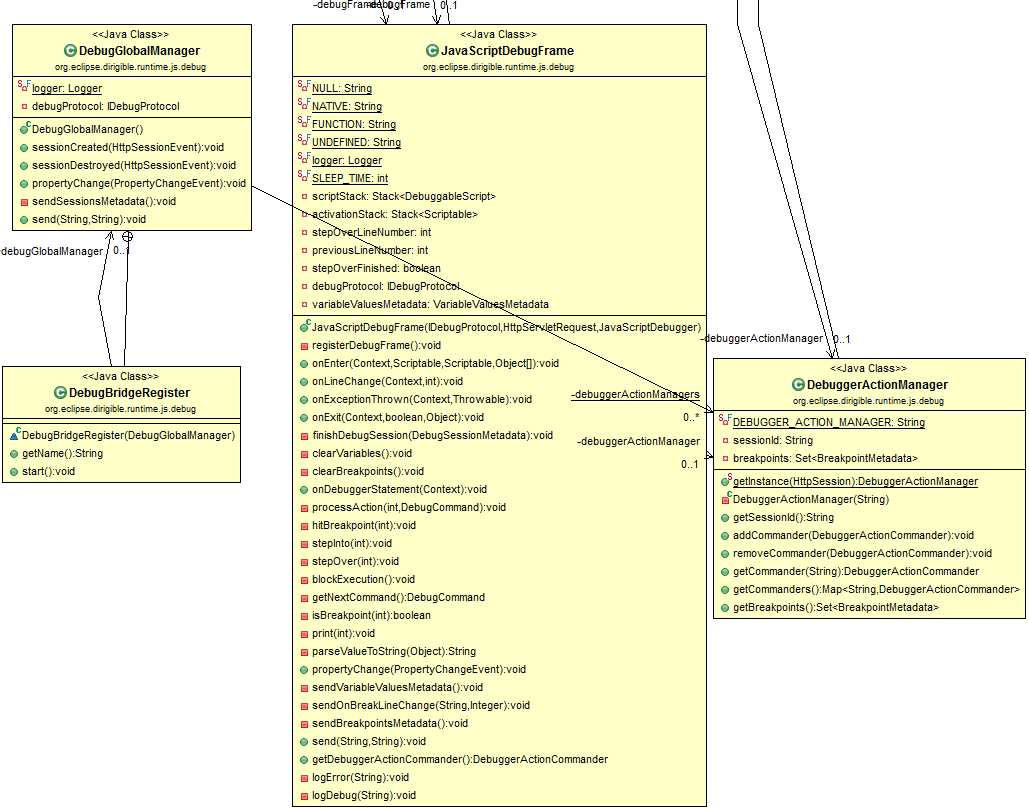
* “JavaScriptDebugServlet” за уеб ресурси в “/js-debug”



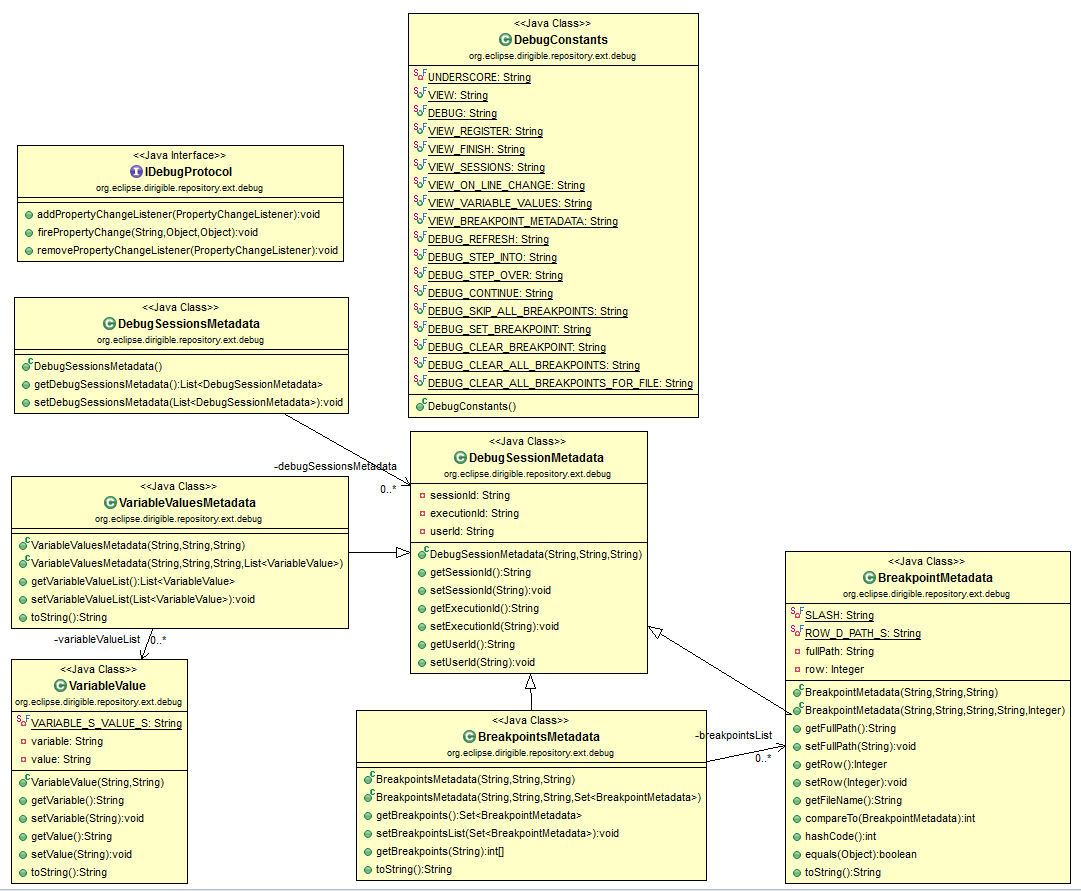
Фиг. 28 ООП модел на модул за “Debugger” (Runtime) в Eclipse Dirigible част 1



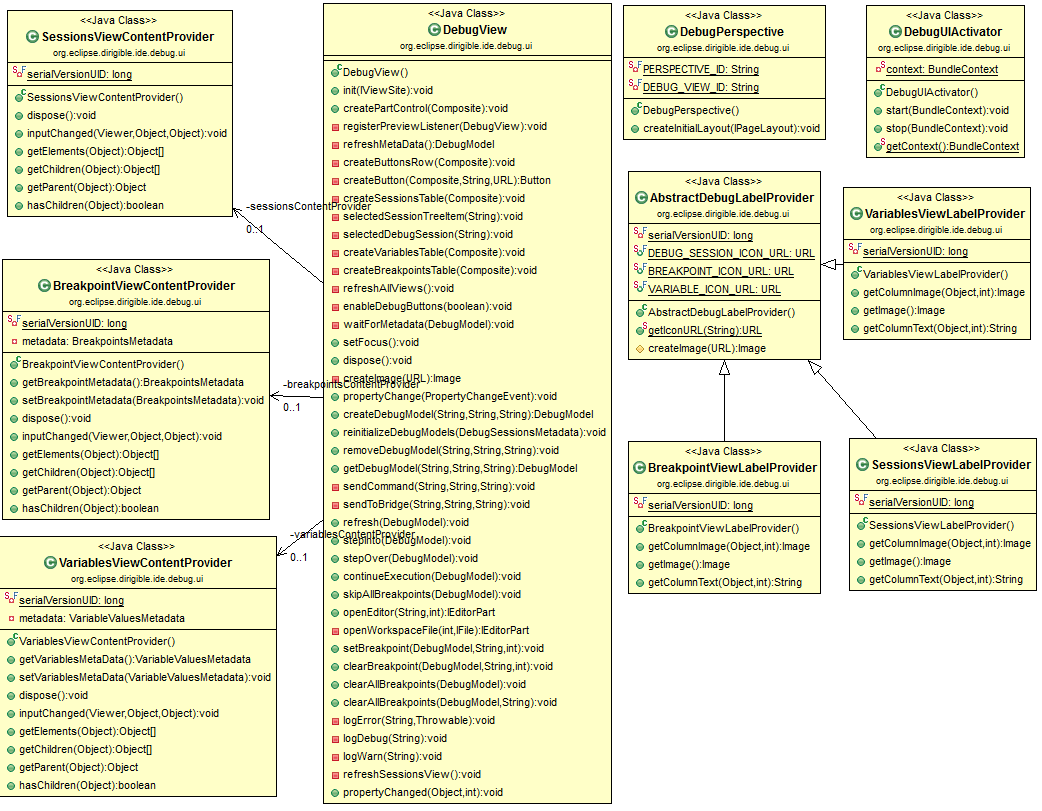
Фиг. 29 ООП модел на модул за “Debugger” (Runtime) в Eclipse Dirigible част 2



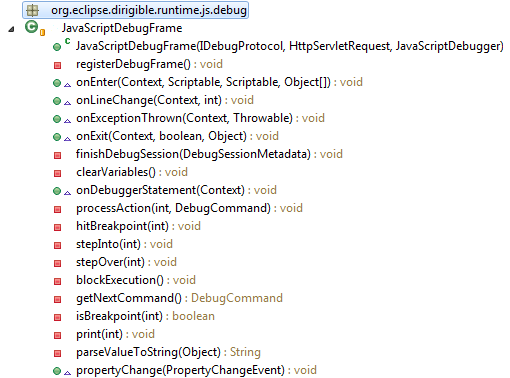
Фиг. 30 ООП модел на модул за “Debugger” (Runtime) в Eclipse Dirigible част 3



Фиг. 31 ООП модел на модул за “Debugger” (Repository) в Eclipse Dirigible



Фиг. 32 ООП модел на модул за “Debugger” (IDE) в Eclipse Dirigible

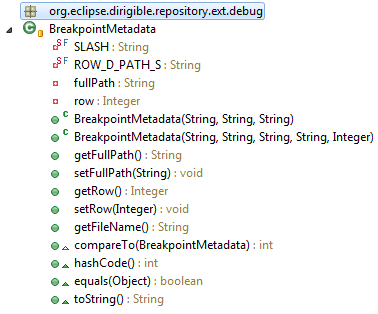


Фиг. 33 Структура на класа “JavaScriptDebugFrame”

Класът “JavaScriptDebugFrame” е основен за “Debugger” модула в Runtime[5] компонента. В него се намира логиката за управляване на Debugging[50] процеса. Някои от основните методи са:

* onEnter() - методът се изпълнява при начало на нова Debugging[50] сесия. В него се записват стойностите на JavaScript променливите и имената на фунцкиите.
* processAction() - методът определя кое ще е следващото състояние, в което ще премине Debugger. Методът действа като краен автомат (машина на състоянията).
* blockExecution() - методът блокира изпълнението на следващи опреации, докато не получи съобщение за промяна или не е в състояние, в което не трябва да се блокира Debugging[50] процеса.

Част от изходния код на класа “JavaScriptDebugFrame” може да бъде видян в “Приложение” - App. 8.



Фиг. 34 Структура на класа “BreakpointMetadata”

Класът “BreakpointMetadata” се намира в Repository[58] компонента от архитектурата на Eclipse Dirigible. Така той може да се използва от IDE[6] и Runtime[5] частите на архитектурата. Класът служи за представяне на данните за Breakpoint (Ref. 15). Някои от основните методи са:

* getFullPath() - връща пътя до файла, в който е поставен Breakpoint.
* getFileName() - връща името на файла, в който е поставен Breakpoint.
* getRow() - връща реда, на който е поставен Breakpoint.

Част от изходния код на класа “BreakpointMetadata” може да бъде видян в “Приложение” - App. 9.

|  |  |
| --- | --- |
| DebugView - Outline.png  Фиг. 35 Структура на класа “DebugView” | DebugView - View.png  Фиг. 36 “DebugView” - графичен интерфейс |

Класът “DebugViеw” е част от архитектурния компонент IDE[6] на Eclipse Dirigible. “DebugView” се използва в “Debug” перспективата за показване на текущите Debugging[50] сеси, списък на променливите и стойностите им, както и списък на активните Breakpoints. Някои от основните методи са:

* createPartControl() - създава UI[62] за работа с “Debugger” модула - бутони (“Step Into”, “Step Over”, “Continue” и други), изгледи за активните Debugging[50] сесии, стойностите на променливите и Breakpoints.
* sendCommand() - изпраща, към “Runtime” частта на “Debugger” модула, командата на потребителя. Такива команди са за “Step Into”, “Step Over”, “Continue” и други.
* refreshAllViews() - обновява съдържанието на UI[62] с последните данни получени от “Runtime” частта.

Част от изходния код на класа “DebugView” може да бъде видян в “Приложение” - App. 10.