СОФИЙСКИ УНИВЕРСИТЕТ “СВ. КЛИМЕНТ ОХРИДСКИ”
ФАКУЛТЕТ ПО МАТЕМАТИКА И ИНФОРМАТИКА
Катедра “Компютърна Информатика”

ДИПЛОМНА РАБОТА

за получаване на образователно-квалификационна степен “магистър”

на Невен Николов Николов, фак. № М-21485

студент от магистърска програма “Разпределени системи и мобилни технологии”

тема:

“Резидентен модул за управление на възлите в йерархичен грид”

Научен ръководител:

доц. д-р Васил Георгиев

София, 2007

Съдържание

Съдържание 2
Увод 4
1. Обзор 6
1.1. Грид-системи 6
Концепция за грид 6
Архитектура на грид-системите 7
Грид приложения - класове 9
1.2. Лек грид – характеристика и примери (обзор) 10
1.3. Лек грид - технологии 15
1.4. Модели за управление на възлите в йерархичен грид 17
Изисквания към управлението на компютърен възел в грид 18
Autopilot 19
GridICE 20
OurGrid 21
2. Модел на управлението на компютърен възел в GrOSD 22
2.1. Системна архитектура на GrOSD 22
Грид портал и клъстърни портали 26
Услуга за управление на ресурсите (RMS) 26
Информационна услуга (IS) 27
Услуга за наблюдение и контрол (MS) 27
Комуникационна услуга (CS) 28
Услуга за управление на възлите (NS) 28
2.2. Сценарии на използване на GrOSD 29
Примери за изпълнение на потребителско задание 29
Използване на услуга 30
2.3. Технологии използвани при реализацията на "Резидентен модул за управление на възлите в йерархичен грид" 31
Java – основно използваният програмен език 32
C++ - програмният език за който няма ограничения 32
Платформата Java 32
Платформата J2EE 33
Enterprise JavaBeans (EJB) 34
RMI(Remote Method Invocation) 35
JNI(Java Native Interface) 36
Инструменти за разработка на Java частта от модула 36

Инструменти за разработка на C++ частта 37
3. Проектиране и реализация на модул за управление на възлите в GrOSD 37
3.1. Проектиране 38
Представяне на високо ниво 38
Изисквания 40
Реализация на системната услуга за управление на компютърен възел 42
Класове и пакети написани на Java 42
Класове и пакети написани на C++ 46
4. Инсталиране и интеграция в системата 48
4.1 Инсталиране 48
4.1.1 Стартиране на услугата при компютърният възел 48
4.1.2 Стартиране на услугата при компютри, на които са инсталирани и работят системни услуги 48
4.2. Интеграция 49
Заключение 50
Възможни бъдещи разширения 51
Библиография 54

Увод

Настоящата дипломна работа представлява част от проекта GrOSD за разработка на лек грид мидълуеър. Изследванията и разработката на грид платформата GrOSD (Grid-aware Open Service Directory) се извършват по проекта SUGrid на Факултета по Математика и Информатика към СУ ”Свети Климент Охридски”, както и в партньорство с европейския изследователски проект CoreGRID (European Network of Excellence CoreGRID). Основната цел е изграждането на лека грид инфраструктура, като приоритет са опростената архитектура и реализация.

Темата на настоящата дипломна работа е Резидентен модул за управление на възлите в йерархичен грид. Цел на дипломната работа е създаване на софтуерен модул реализиращ системна услуга за управление на компютърен възел, това включва определяне на хардуерни и софтуерни параметри на компютърният възел, определелянето на натовареността на различните хардуерни компоненти.Друга функционалност, която е необходима е стартиране или спиране на задачи, комуникация с останалите системни услуги в грид-системата.

Основни характеристики, които трябва да притежава дипломната работа са:

а) трябва да съдържа обзор на грид и анализ на използваните в аналогични гридове системи за управление на компютърни възли

б) реализираният резидентен модул за управление на възлите в йерархичен грид, трябва да прави това, което е поставено като негова цел.

в) системната услуга трябва да е написана по начин позволяващ лесната и промяна и разширяване – това включва използването на конвенция за писане на софтуерен код, писане на коментари и прочие. Добрият дизайн и документация са изискване, за да е изпълнено горното условие.

г) поради факта, че крайният продукт(GrOSD) част от който е и модула за управление на компютърен възел се очаква да работи върху много компютри за които нетрябва да се заплащат допълнителни лицензни такси, то реализацията на модула трябва да използване само нелицензионни програмни средства(среди

за разработка, за документиране и пр.) както и да позволява инсталирането и използването на модулите на платформи(операционни системи, приложни сървъри и прочие) базирани на софтуер с отворен код.

Дипломната работа притежава стандартна организация, разделена е на четири основни части.

Глава 1. Обяснение на понятието грид. Обзор на леките гридове.

Целта на тази глава е да се обясни понятието грид, да се дадат основни понятия, които се използват в грид, както и да се направи кратко описание на съществуващите грид архитектури и използваните в грид системите технологии.

Глава 2. Представяне на грид системата GrOSD. Детайлно описание на резидентен модул за управление на възлите в GrOSD.

Представя се модела на управление на компютърни възли в грид-системата GrOSD.

Глава 3. Описва подробно програмната реализацията на резидентен модул за управление на възлите в йерархичен грид.

Описват се използваните технологии и се обяснявава защо те са предпочетени за реализацията на софтуерния модул

Глава 4. Разглежда дейностите, свързани с внедряване на резидентен модул за управление на възлите като инсталиране и интеграция в системата.

Обяснява се начина на инсталация на резидентния модул за управление на възлите, както на обикновените машините на които ще работят грид заданията, така и на компютри на които ще работят системните грид услуги.

1. Обзор

1.1. Грид-системи

Грид изчисленията са едно от най-новите направления в развитието на разпределените системи. Тази дисциплина изучава най-общо свързването на потенциално неограничен брой всевъзможни изчислителни(но не само, може да са и устройства за запаметяване на данни, именно това е една от разликите между грид и интернет, в грид може да се споделят всички хардуерни ресурси) устройства посредством мрежови връзки и услуги в една “мрежа” или “решетка” (grid). Грид-системите могат да добавят неограничен брой изчислителни(съхраняващи информация) устройства във всяка грид-среда, като по този начин увеличават изчислителните(съхраняващите) възможности на грид-средата.

За грид започва широко да се говори в края на 90те години на миналия век. Преди това опитите за координирано използване на широко разпределени ресурси са известни като метакомпютинг. През този етап се свързват суперкомпютри в изследователски центрове с цел решаване на конкретни научни проблеми. От този период датират проекти като FAFNER, който изчислява възможността за откриване на защитния код на RSA криптиращия алгоритъм, и I-WAY, при който се свързват с висикоскоростни мрежи паралелни компютри в няколко научни центъра в САЩ. Тези ранни проекти могат да се считат за предшественици на грид.

Концепция за грид

Според една от първите дефиниции, изчислителен грид е апаратно-програмна инфраструктура, предлагаща контролиран, постоянен, прозрачен и широкоразпространен достъп до система от високопроизводителни изчислителни ресурси, дадена е от Фостър и Кеселман в [1]. Надеждността е много важна за потребителите на грид – те трябва да са сигурни, че ще получат предсказуема, постоянна, а често и висока производителност от грид-системата. Системата трябва да предлага консистентни услуги, в смисъл, че услугите трябва да са стандартни, достъпни през стандартни интерфейси, използващи стандартни параметри. Само така може да се постигне широко възприемане на новата технология. Широката разпространеност гарантира на потребителите, че

грид услугите са достъпни винаги, в каквато и да е среда – както е достъпна електроенергията. Тази инфраструктура трябва да предлага евтин достъп, защото в противен случай не би могла да намери широко възприемане.

Тази дефиниция набляга основно на изчислителните възможности на грид. По-късно авторите я променят, като наблягат на идеята, че основната задача на грид е споделянето на ресурси, и по-конкретно – "координираното споделяне на ресурси и решаване на задачи в динамични, мултиинституционални виртуални организации" [2]. Споделянето се отнася до директен достъп до компютри, програмно осигуряване, данни и други ресурси. То е стриктно контролирано, като доставчиците и потребителите на ресурси ясно дефинират какво точно се споделя, кой има право да споделя и при какви условия се извършва споделяне. Група индивиди и/или институции, обединени от подобни правила за споделяне, съставляват виртуална организация (ВО). Освен координираното споделяне на ресурси и формирането на виртуални организации, от съществено значение за грид-системата е наличието на отворени стандарти. Последните осигуряват средства за взаимодействие и интеграция, и трябва да се използват при откриването, достъпа и координацията на ресурсите.

Архитектура на грид-системите

От съществено значение за грид архитектурата е тя да се базира на общи отворени протоколи, дефиниращи основните механизми чрез които членовете на една ВО и ресурсите договарят, установяват и извършват споделянето. Отворената архитектура, базирана на стандарти, улеснява разширяемостта, взаимодействието, преносимостта и споделянето на код, докато общите протоколи позволяват дефинирането на стандартни услуги.

Фостър и Кеселман предлагат в [2] слоен модел на грид архитектура, който се състои от няколко слоя и следва принципа на "пясъчния часовник" - в тясната част на "часовника" се дефинират малък брой основни абстракции и протоколи, които предлагат функционалност на множество услуги от по-високите слоеве, и от своя страна се базират на множество технологии, които се намират в по-ниските слоеве.

В предложената архитектура най-ниското ниво предоставя ресурсите, до които грид протоколите осигуряват достъп. Това могат да са изчислителни ресурси,

ресурси за съхранение на данни, мрежови ресурси, сензори. Това ниво се нарича "тъкан" (Fabric) и представлява апаратната инфраструктура на грид-системата. Слоят над него има два подслоя и съставя тясната част на "часовника". Единият подслой се нарича съобщителен (connectivity) и дефинира основни протоколи за комуникация и идентификация (authentication). Комуникационните протоколи позволяват пренос на данни между ресурсите, изграждащи грид инфраструктурата. Идентификационните протоколи надграждат комуникационните услуги и осигуряват криптографски сигурни механизми за установяване идентичността на потребители и ресурси. Ресурсният подслой се базира на протоколите на съобщителния подслой и дефинира протоколи за сигурно договаряне, наблюдение, контрол, счетоводство и заплащане за операциите върху ресурсите. Тези протоколи са насочени напълно към индивидуални ресурси. Следващият слой предлага протоколи, услуги и приложни програмни интерфейси (API), които не са свързани с конкретен ресурс, а са глобални и реализират взаимодействието между ресурсите в инфраструктурата. Затова този слой се нарича колективен (collective). Той увеличава разнообразието от услуги като комбинира малкия брой протоколи от долния слой. Последният слой от тази архитектура е приложният – тук се изпълняват грид приложенията. Те използват услугите, предоставени от всеки от долните слоеве.

Така описаната архитектура намира пълна реализация в Globus Toolkit [3]. Той предоставя програмна инфраструктура за свързване на хетерогенни възли във виртуален компютър. Globus Toolkit поддържа групи от услуги, като комуникация, защита, регистрация и управление на ресурсите, като услугите са със строго дефинирани интерфейси. Пакетът от инструменти поддържа съвместно ползване на ресурси от множество организации.

С нарастване на интереса към грид системите се увеличава необходимостта от стандарти, които да се използват в грид архитектурата. Един от най-важните стандарти в това отношение е Архитектурата за отворени грид услуги (Open Grid Services Architecture – OGSA) [4], [5]. Той е издаден от Global Grid Forum (GGF) и представлява спецификация, която дефинира обща, стандартна и отворена архитектура за грид приложения. Тази спецификация цели да стандартизира почти всички услуги, които грид приложенията могат да

използват. OGSA специфицира архитектура, ориентирана към услугите (Service-Oriented Architecture – SOA) за грид, която създава модел на изчислителна система, състоящ се от множество разпределени изчислителни шаблони. Като технология за реализация на последните се предвиждат уеб услугите. OGSA стандартът дефинира интерфейси на услугите и определя протоколите за използване на тези услуги, но не предоставя конкретна реализация.

Грид приложения - класове

Грид-системите могат да се използват за решаване на разнообразни задачи. Все пак, обособяват се пет основни класа от грид приложения [2], въз основа на изискванията и предназначението на приложенията.

- Разпределен суперкомпютър – този тип приложения използват услугите на грид-системата, за да получат достъп до голям брой изчислителни ресурси, с цел решаване на задача, която изисква интензивни изчисления и не може да се реши от единствена система. Примери за такива приложения са военни симулации, субатомни, атомни и молекулярни задачи, задачи от небесната механика и др.
- Високоефективна фонова обработка (High throughput computing) – в този случай грид-системата насрочва и изпълнява голям брой слабо свързани или независими задания, с цел да се оползотворят неизползвани ресурси. Подобни приложения се използват за дизайн на чипове, решаване на криптографски задачи и др.
- Обработка по заявка (On-demand computing) – тези приложения използват възможностите на грид, за да посрещнат кратковременна нужда от ресурси, които не би било финансово ефективно да се подържат локално. Такива ресурси могат да са изчислителни ресурси, програми, хранилища за данни, специализирани сензори и др. Пример за подобни грид приложения е обработката на данни, генерирани от сложна медицинска апаратура.
- Обработка на големи обеми от данни (data intensive computing) – при тези приложения фокусът се поставя върху синтезиране на информация от данни, намиращи се в географски разпределени хранилища, цифрови библиотеки и бази от данни. Често тази обработка е съпроводена с

интензивни изчисления и комуникации. Такива приложения са нужни при обработка на огромното количество данни, натрупано при експерименти от физика на високите енергии, от телескопи, или метеорологични спътници.

- Съвместна обработка (Collaborative computing) – този клас приложения са основно ориентирани към улесняване на взаимодействието между хората и изграждане на сложни колективи за решаване на определени задачи. Могат да се използват за симулации в реално време, интерактивни игри, и др.

Като цяло, грид представлява сравнително ново и многообещаващо направление в развитието на разпределените системи. Развитието на грид и широкото му разпространение могат да променят начина, по който понастоящем се използват изчислителните ресурси. Към момента грид-системите се използват предимно в научните и инженерни среди. Основна задача пред грид в бъдеще е повсеместното му разпространение сред всички групи потребители.

1.2. Лек грид – характеристика и примери (обзор)

Много от проектите в областта на грид са ориентирани към определена приложна област, или дори към изпълнение на специфични приложения – пример за това е проектът EU DataGrid, който е насочен към създаване на грид инфраструктура за анализ на големи обеми от данни, натрупани при научни експерименти и формиране на научни екипи за сътрудничество. От друга страна, повечето платформи, които предлагат грид мидълуеър, като например Globus Toolkit, са твърде сложни – те имат много богата функционалност и голям брой инструменти, което ги прави трудни за инсталация и администрация. Като следствие от това възниква необходимостта от грид платформи, които са опростени и общи – не са ориентирани към конкретна задача и предлагат базова функционалност, която може да се разширява при необходимост. Тези грид платформи са известни като “леки” (lightweight) грид-системи.

По своята архитектура, реализация, предлагани протоколи и услуги леките грид-системи се различават значително помежду си. Основно ги обединява това,

че те са проектирани за бързо внедряване и минимални разходи за експлоатация и поддръжка. Те са подходящи за използване от (и дори предимно са насочени към) индивидуални потребители или по-малки организации. Този клас потребители не се нуждае от всички услуги и протоколи, които сложните грид платформи предлагат и за тях лекият грид предлага базова функционалност, която е достатъчна за изпълнение на техните задачи, като при това могат да се използват предимствата на грид изчисленията като споделяне и прозрачно използване на ресурси. Обикновено се предвижда такива грид-системи да се ползват за решаване на задачи с по-скромни мащаби, но това не е задължително.

За разлика от сложните грид системи, леките грид платформи са лесни за инсталиране и администриране – повечето дори не изискват специални администраторски права или задълбочени познания. Простотата на използване ги прави подходящи за разнообразни потребители и организации. Освен това те обикновено не са насочени към решаване на конкретна задача, и затова са подходящи както за научни цели, така и за приложение в корпоративна среда, или дори сред масови потребители.

Характерно за този тип грид-системи е динамичната промяна на инфраструктурата, лежаща под мидълуеъра – динамичното включване и отпадане на ресурси и потребители. Това позволява подобни системи да се използват в динамична среда, като например сред индивидуални потребители, които се включват и изключват спорадично в системата и използват и предоставят временно ресурси. Подобна динамика е присъща и на самия мидълуеър – той предлага само най-базовата функционалност, а ако са необходими допълнителни услуги, те се добавят динамично. Обикновено в леките грид-системи управлението на потребителите и на ресурсите е разпределено, няма напълно централизиран контрол, което допринася за увеличаване на динамиката на системата.

В [6] са разгледани основните изисквания, на които трябва да отговаря един лек грид с общо предназначение и компонентна архитектура. Следват някои от тези изисквания:

- Лек и с архитектура, която не е ориентирана към конкретна задача – повечето съществуващи грид-системи разчитат на множество ресурси и

инструменти, което им пречи да бъдат достатъчно общи. Общата грид платформа трябва да е “лека”, с минимална, но съществена функционалност, като може да бъде разширявана с допълнителни характеристики. Това би позволило грид технологиите да се използват от всякакви потребителски устройства.

- Статични и динамични метаданни – статичните метаданни са особено важни в компонентна среда, тъй като те дават информация за компонентите, която е нужна за правилното им използване и комбиниране, като например версия, производителност, проблеми със съвместимостта и др. Динамичните метаданни предоставят информация за динамичните свойства на компонентите и са важни например за удовлетворяване на качеството на услугите, оптимизация, възстановяване след грешки и др.
- Динамично внедряване на компоненти – системата трябва да може динамично да включва компоненти, например като реакция на промени или нараснали нужди.
- Преконфигуриране и адаптиране – системата трябва да е в състояние да се адаптира към променящата се среда, като по този начин се гарантира отказоустойчивост.
- Поддръжка на клиент/сървър и P2P споделяне на ресурси – при клиент/сървър модела на споделяне съответствието между заявки и ресурси се прави централизирано от брокер, докато при P2P доставчиците и ползвателите на услуги комуникират без посредник. При първия подход могат да се наложат конкретни политики на употреба на ресурсите, докато при втория се намалява времето за отговор и се увеличава производителността. Затова и двата модела трябва да се поддържат.
- Предоставяне на услуги при нужда – платформата трябва да поддържа създаване на услуги при поискване (on demand), когато са нужни на потребителите.
- Минимален но достатъчен модел на сигурност – високото ниво на сигурност, високата производителност и простотата на платформата са

взаимно противоречащи си цели. Затова между тях трябва да се намери правилното съотношение. Подходящо е да се реализира минимално ниво на сигурност, но да се предвиди поддръжка на допълнителни механизми за сигурност, които да се използват при нужда.

- Разпределено управление – разпределеното и същевременно координирано управление е в основата на функционирането на платформата.

Следват няколко примера за леки грид платформи.

ALiCE (Adaptive and scalable Internet-based Computing Engine) [7], [8] представлява лек грид мидълуеър за създаване и внедряване на общи грид приложения. ALiCE е завършена грид-система. Тя улеснява натрупването и виртуализацията на ресурси в интранет и използването на свободни ресурси в Интернет. Платформата се състои от няколко ясно обособени слоя. В най-ниския слой се намират базовите услуги, предлагани от системата. Те включват откриване, заделяне и управление на ресурси, управление на данни, управление на сигурността, комуникация между обектите, счетоводство и мониторинг. Следващият слой поддържа разработката и внедряването на грид приложения. Той скрива детайлите на паралелното програмиране като предлага библиотека с шаблони за изграждане на грид приложения, които улесняват разработката. Най-горният слой съдържа инструменти и приложения, които се използват от потребителите на системата.

Заданията в грид-системата се подават от компютър, наричан “консуматор”, и се изпълняват на свободни компютри, които се наричат “производители”. Заданията се предават от консуматорите на производителите посредством ресурсен брокер, който управлява процесите и ресурсите. Брокерът съдържа и планировчик (scheduler), който се грижи за правилното разпределяне на заданията по производители. На всеки консуматор има потребителски интерфейс, които улеснява подаването на заданията, докато на производителите работи модул, управляващ изпълнението на заданията.

Друг проект реализиращ лека грид платформа е H2O [9], [10]. Той е ориентиран към по-малки организации и индивидуални потребители. В H2O се набляга на силната разпределеност на ресурсите, преконфигурирането и адаптивността.

Платформата е компонентна и ориентирана към услуги. Базира се на идеята всеки ресурс да се представи като софтуеърен компонент, който предоставя услуги чрез ясно дефинирани отдалечени интерфейси. Собствениците на ресурсите предоставят среда за изпълнение под формата на контейнер за компоненти, който се изпълнява върху ресурса. Тези контейнери се наричат ядра (kernels). В тези контейнери могат да се стартират услуги, също под формата на компоненти. Тези услуги се наричат плъглети (pluglets). Докато при повечето компонентни платформи собствениците на контейнерите са също и тези, които предоставят услугите, които се изпълняват в контейнерите, при модела на H2O това не е задължително. Възможно е трета страна, която няма нищо общо със собственика на контейнера, ако има необходимите права, да стартира услуга в контейнера. Този модел на споделяне прехвърля тежестта на внедряване на услугите в контейнерите от собствениците на ресурси към доставчиците на услуги, което може да насърчи повече потребители да споделят ресурси. Платформата осигурява разнообразие от протоколи за комуникация между компонентите, както и различни механизми за сигурност.

OurGrid [11], [12] е лека грид-система, разработвана от Федералния университет на Кампина Гранде, Бразилия, със съдействието на Hewlett Pacard. Тази платформа, за разлика от разгледаните дотук, не е съвсем обща – тя е предназначена за решаване на асинхронни паралелни задачи (т.е., такива паралелни задачи, които могат да се разделят на независими подзадания). Въпреки, че този тип задачи са по-прости за програмиране, те намират голямо приложение, например при Монте Карло симулации, фрактални изчисления, изчисления в биологията, компютърната обработка на изображения и др. Основните компоненти на системата са три:

- MyGrid – представлява централна точка на грида. При изпълнение на задания той координира обработката, като извършва планиране на заданията, трансфер на данни, наблюдение на изпълнението. Чрез него се подават задания в грида. Този модул се изпълнява на “home” машина. Всички останали машини в грида се наричат грид машини.
- Peer – изпълнява се на “peer” машина. Неговата задача е да организира и предоставя грид машините в същия административен домейн. Той се използва от MyGrid модула за осигуряване на грид машини.

- Потребителски агент (User Agent) – работи на всяка грид машина, като осигурява достъп до нея за изпълнение на подадените в грида задачи.

Потребителите на тази грид платформа се обединяват в P2P общност, като всеки в общността използва свободните ресурси на останалите. При този сценарий е възможно потребители само да използват ресурси, без да споделят. За да се избегне това, OurGrid реализира механизъм за заделяне на ресурси, който гарантира предимство на потребителите, които споделят своите ресурси, като по този начин стимулира споделянето.

В заключение може да се каже, че леките гридове са тип грид- системи, които биха могли да притежават различни характеристики, но основните им качества са опростената, олекотена архитектура, общото им предназначение и наличието само на най-основни услуги. Тези техни черти биха могли да допринесат за разпространението им сред по-широки групи потребители.

1.3. Лек грид - технологии

За реализацията на леките грид платформи се прилагат разнообразни технологии. Повечето като език за програмиране използват Java, от където следват и технологиите, поддържани от платформата Java. Използването на този език гарантира платформена независимост на лекия грид, което е от съществена важност, тъй като се предполага, че грид мидълуеърът ще работи на машини с различни архитектури и операционни системи. Не на последно място стои и фактът, че платформата е безплатна. Леките грид-системи използват и разнообразни протоколи, среди и др.

ALiCE е реализирана на Java и прилага Java технологии като Jini, JavaSpaces, GigaSpaces и JNI.

Jini [13] представлява мрежова технология, която улеснява динамичното свързване и взаимодействие на услуги и устройства в мрежа. Обединението на услуги и устройства се нарича Jini федерация. Услугите могат свободно да се присъединяват към федерацията и да я напускат. Във всяка федерация има поне една директорийна услуга, в която се регистрират останалите услуги. Когато една услуга се присъединява към федерация, тя се регистрира в директорийната услуга като изпраща информация за себе си и свой посредник (proxy) – обект, чрез който е достъпна съответната услуга. Клиентите могат да търсят услуги по

техните описания. Ако клиент намери услуга, която иска да използва, директорийната услуга му изпраща съответния обект посредник, чрез който клиентът може директно да комуникира с услугата.

Jini използва JavaRMI за предаване на обекти между Java виртуални машини. Това позволява на Jini устройствата да пренасят изпълним код из мрежата, към която са свързани, като по този начин се създават мрежи от устройства, които не изискват предварително планиране, инсталиране и човешка намеса.

JavaSpaces [14], която е част от Jini, представлява проста, но мощна технология за изграждане на разпределени приложения. В приложение, използващо тази технология, всички процеси са свързани посредством мрежа, като комуникират и синхронизират действията си посредством хранилище за обекти, наречено пространство (space). Всички комуникации в ALiCE се извършват посредством JavaSpaces.

Като алтернатива на JavaSpaces в ALiCE може да се използва технологията GigaSpaces [15]. Тя представлява платформа за синхронизация и комуникация, осигуряваща програмна инфраструктура за информационно взаимодействие на корпоративни разпределени приложения и уеб услуги. Основната разлика между JavaSpaces и GigaSpaces е, че първата технология осигурява логическа разпределена обща памет, докато втората реализира разпределена обща памет като обединява няколко пространства на различни машини.

ALiCE използва JNI за да направи възможно извикване на не-Java код по време на изпълнение.

H2O също е реализиран на Java, като при това използва технологиите, предлагани от Java платформата. За комуникация се използва модифициран вариант на RMI – RMIX [16]. Той добавя към стандартния RMI възможност за асинхронни извиквания, както и еднократни извиквания, които могат да се използват за реализация на предаване на съобщения. Всъщност, RMIX представлява обща платформа и семантика над множество протоколи. Различни протоколи се поддържат чрез добавянето на съответните модули , което може да стане дори по време на изпълнение. Понастоящем са реализирани два протоколни модула – RMIX-JRMPX, който е базиран на JavaRMI и Java сериализация, и RMIX-XSOAP, базиран на XSOAP и използващ SOAP като

протокол за пренос на данни. Въпреки, че RMIX е протоколът по подразбиране за H2O компонентите, то е напълно възможно компонентите да използват други средства за комуникация.

Други технологии, които се използват в H2O, са JNI, която се използва за да се стартират услуги, които не са писани на Java, в Java контейнери, както и уеб услугите – възможно е интерфейсът на приложна услуга, работеща в контейнер, да се опише посредством WSDL.

Разнообразието от технологии за разработката на леки грид-системи дава голяма свобода за тяхната реализация. От направения преглед се вижда, че основно се използват Java платформата и различни Java технологии, което се обуславя от платформената независимост на езика и множеството средства, които платформата предлага.

1.4. Модели за управление на възлите в йерархичен грид

Популярността на грид-системите все повече се увеличава, а заедно с това се увеличава и разнообразието на компонентите на грид-системата и броя на потребителите и. Това ги прави уязвими към грешки, прекъсване и претоварване. Необходимо е да се наблюдават компонентите, тяхната употреба, откриване на причини и състояния, които могат да доведат до критични точки, отказ или провал. Сигурното(отказоустойчиво) управление на компютърните възли в грид-системата е важно за осигуряването на надеждна, сигурна и ефективна работа.

Целта на управлението на компютърният възел в грид е да се управляват задачите, които се изпълняват както и за наблюдение на основните хардуерни и софтуерни параметри на компютърната система. За да може управлението да е ефективно, то трябва най-общо да управлява(контролира) правилно софтуера и хардуера. Управлението на хардуера включва определяне на точна информация относно вида, количеството и натовареността на използвания хардуер(процесор , рам памет, външна памет, мрежова карта, принтер, твърди дискови устройства и др.).Управлението на софтуера включва определянето на операционната система, контрол на вече работещите или определени за изпълнение задания.

Информацията, която се получава за параметрите на хардуера и софтуера на един компютърен възел е много важна за цялостното функциониране на грид системата, защото ако тези параметри не са правилно определени, потребителските задания няма да могат да бъдат изпълнявани или ще бъдат изпълнявани значително по-дълго от нормалното време(т.е. системата няма да работи така както очаква крайният потребител.По този начин тя би изгубила доверието на крайният потребител.).

Изисквания към управлението на компютърен възел в грид

Някои от стандартните аспекти за управление на компютърни възли, които задължително трябва да се срещат и в услугите за управление на компютърни възли са следните:

- Adaptability (адаптируемост) – способността на системната услуга да позволява лесно разширяване на функционалността и или лесната и адаптация според нуждите на съответната грид-система, в която се използва (т.е. лесна конфигурация на системната услуга, така че тя да бъде настройвана споредe конкретните изисквания и да може да бъде преизползвана в различни грид-системи).

- Fault tolerance(отказоустойчивост) – възможността на системната услуга да продължи да функционира пълноценно, въпреки възникването на някакви грешки в нея

  (Съществуват и различни видове грешки.Конкретен пример за грешка може да е временна невъзможност на услугата за комуникация с останалите системни услуги.Това не трябва да нарушава основния процес на работа на услугата за управление на компютърния възел, разбира се тя трябва периодично да прави опити за подновяване на връзката с другите системни услуги.

- Да позволява лесна разширяемост на функционалността, както и да е изградена на модулен принцип, който да позволява лесно да се добавят модули за нови операционни системи, на които системната услуга ще работи(виж забележката за Java. Налага се написването на специфични подмодули за всяка нова операционна система, на която ще работи системната услуга. Това е цената, която се заплаща заради

преносимостта на Java програмите. Все пак добрия дизайн на системния модул позволява да се сведе до минимум обхвата на специфичните подмодули, а основната логика да се преизползва за всички операционни системи).

- Да е написана с легален софтуер, за да не се нарушават авторските права и интелектуалната собственост.
- Да има ясни и добре определени интерфейси за комуникация с останалите системни услуги като не се нарушава принципа за loose coupling – т.е. да е минимално обвързана с имплиментацията им

Основни компоненти на "Резидентен модул за управление на възлите в йерархичен грид "(Node Service):

- **Information service** (информационна услуга) –

  осигурява(предоставя на заинтересуваните услуги) информация за хардуерни и софтуерни параметри на компютърните възли, както и информация за работещите потребителски задачи

- **Изпълнение на потребителски задачи –** включва взимане на параметри на задачата, която ще се изпълнява, самото изпълнение на потребителска задача, както и връщането на резултата от потребителската задача на RMS(услугата за управление на ресурси).

Примери за реализация на модели за управление на компютърни възли в грид

За илюстрация на различни механизми на управление на компютърни възли в грид следва описание на конкретни реализации в няколко грид-системи.

Autopilot

Autopilot [17] е инфраструктура за контрол в реално време на паралелни и разпространени компютърни ресурси. Целта на Autopilot е да осигури среда за разпределени приложения с контрол в реално време, така че приложенията да могат автоматично да избират, конфигурират и управляват ресурсите, базирани на наблюдението на системата. Autopilot е разработена от Pablo Research Group – Университет Илиноис, намира приложение в много проекти, включително и в Grid Application Development Software Project (GrADS – проект за софтуерно

разработване на грид приложение). Архитектурата на Autopilot е основана на стандартната и използва Globus Toolkit за комуникация между своите компоненти, а за защита на информацията се използват защитните механизми на Глобус(Globus). Основните компоненти, от които се състои Autopilot са:

- Sensors – съответстват на потребителите, следят работата на системата и приложенията
- Actuators – съответстват на производителите, осигуряват механизми за управление на отдалечени приложения и контролират дейността на сензорите
- AM (Autopilot Management) – съответства на справочната услуга, поддържа искания за регистрация от отдалечени sensors и actuators, както и осигурява механизъм за клиентите, сами да намерят информация за ресурс.

Така описаната архитектура на системата отговаря на изискванията за скалируемост и отказоустойчивост тъй като системната услуга за управление на един възел не зависи от функционирането на другите възли.Също така разширяемостта на системата е лесно тъй като за всеки нов компютърен възел се стартира и неговият Actuator.

Autopilot е наличен за свободно изтегляне за обучение, изследвания и некомерсиални цели. Софтуерната преносимост е ограничена за UNIX – базирани платформи. Системата зависи от Globus Toolkit 2.2 и включва библиотеки на Pablo SDDF.

GridICE

GridICE[18] е разбработена за наблюдение, анализ поведението и представянето на грид ресурси. Целта на проекта е да осигури на клиентите отчети на грешки, събития, предизвикани от потребителя или системни нарушения. GridICE е планирана за интегриране с Grid Information Service (GIS) и използва Globus MDS за откриване на нови ресурси. Web – базирания интерфейс осигурява изглед на ресурсите, основани на виртуални организации, грид сайт и изискванията на потребителите. Разбработен е от INFN-Grid и European Data TAG, използва се от LHC Computing Grid и INFN Production Grid.

Архитектурата на GridICE се състои от няколко слоя:

- Management Service
- Publisher Service
- Data Collector Service
- Detection and notification service, Data Analyzer
- Presentation Service

Така описаната архитектура отговаря на изискванията за скалируемост и отказоустойчивост. Много потребители могат едновременно да използват web интерфейса на GridICE, за да получат информация за статуса на ресурсите. Тъй като GridICE е централизирана система, за осигуряване на скалируемост и отказоустойчивост е необходимо да се използват няколко GridICE сървъра за наблюдение на различни части и всеки от сървърите да отчита данните на отделен MDS2s.(Monitoring and Discovery Service). Тъй като GridICE няма механизъм за нотификация за наличие на нови ресурси в MDS2, баланс може да бъде постигнат между честотата на опитите, скоростта, с която ресурсите се добавят в Grid и времето, за което новите ресурси се показват на потребителите.

GridICE не разполага с механизми за защита и сигурност, цялата информация е общодостъпна за анонимни потребители на MDS2 и Web интерфейса на GridICE.

GridICE е софтуер с отворен код за свободно ползване.

OurGrid

OurGrid е предназначен за решаване на асинхронни паралелни задачи, безплатен е за ползване и има голяма известност сред испано езичната грид общност.

Основните компоненти на системата са три:

- MyGrid – представлява централна точка на грида.Играе ролята на RMS.
- Peer – задачата му е да организира групите от един и същ административен домейн
- Потребителски агент (User Agent), изграе ролята на модул за управление на компютърен възел

Така описаната архитектура отговаря на изискванията за скалируемост и отказоустойчивост. OurGrid е една много добра безплатна грид-система, в която се използва принцип от т.н. “Economical Grid”, а именно който дава повече ресурси на системата, да може да ползва повече и с предимство.Чрез този принцип се стимулират грид потребителите да предоставят колкото може повече ресурси, когато на тях не са им нужни, за да могат да използват повече когато им трябват.Благодарение на този подход, те си осигуряват винаги повече налични отколкото използвани ресурси, с което се осигурява безотказност на системата.

От анализа направен на 3-те грид системи се вижда, че архитектурите, които са използвани отговарят на изискването за скалируемост и отказоустойчивост. Такъв извод автоматично не може да се прави и за имплиментираните в тях аналози на “Резидентен модул за управление на възлите в йерархичен грид”, освен ако не се предоставят статистически таблици за отказоустойчивост на компютърните възли.

2. Модел на управлението на компютърен възел в GrOSD

Изграждането на “резидентен модул за управление на възлите в йерархичен грид”, която е обект на настоящата дипломна работа, е осъществено в контекста на грид платформата GrOSD. С цел по-ясното представяне на дизайна и функционалността на услугата за управление на компютърния възел, в тази глава се разглежда цялостната архитектура на платформата GrOSD, след което се описва модула за управление на компютърни възли проектиран за GrOSD.

2.1. Системна архитектура на GrOSD

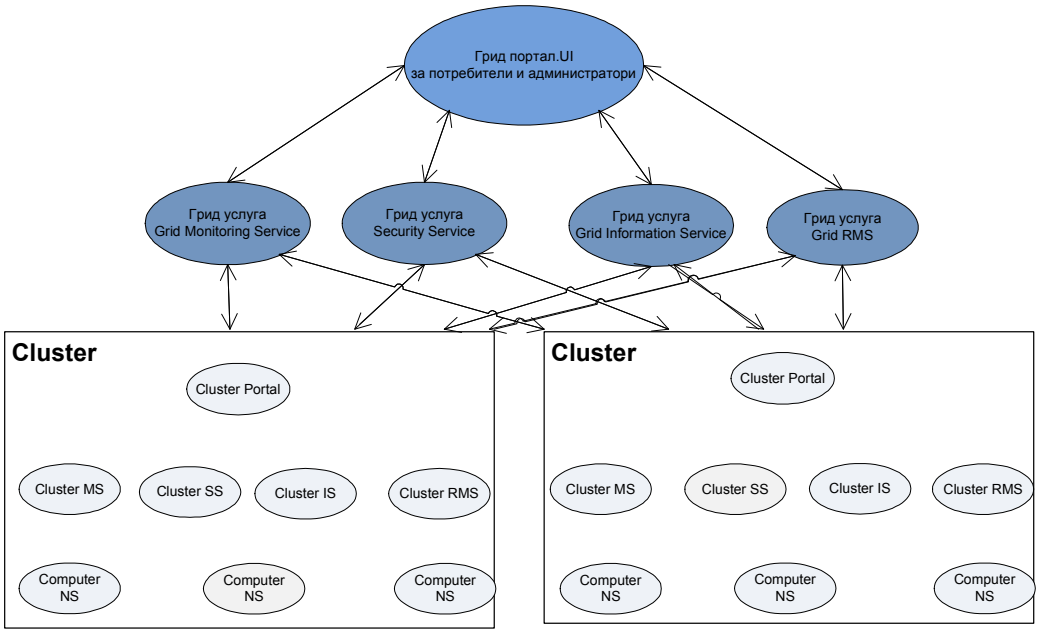
GrOSD се отличава с йерархична многослойна архитектура [22], [23], [24]. Тази грид-система е изградена от свързани “виртуални клъстъри” от компютри. Под виртуален клъстър следва да се разбира свързани чрез мрежа компютри, принадлежащи на обща административна област. На тези компютри работи грид мидълуеър(посредник), който предлага хомогенни системни услуги и обединява машините във “виртуалния клъстър”. Последният е единица за

организиране на възли, ресурси и потребители като в действителност представлява по-скоро логическа единица, отколкото физическа. В GrOSD не е нужно компютрите, които са част от един “виртуален клъстър” да се намират физически на едно и също място. До края на дипломната работа за краткост “виртуалния клъстър” ще се нарича само клъстър. Архитектурата е йерархична и има три нива. Първото от тях е локалното ниво. На това ниво се намират различните клъстъри, които съставят системата. Второто ниво се изгражда чрез свързване на клъстъри и представя самия грид. Последното ниво е “междугридово” – на него е възможна връзка с други грид-системи. Всеки клъстър разполага със собствени системни услуги, като на грид нивото има системни услуги, общи за целия грид.

Една от целите на проекта е платформата да бъде изградена от прости и ефективни компоненти. Архитектурата на GrOSD е базирана на модела на архитектура, ориентирана към услугите (Service Oriented Architecture). Входната точка към системата е грид порталът. Грид порталът и клъстърните портали предлагат сходна функционалност за потребителите. Системните услуги, на които се базира платформата са следните:

- Услуга за наблюдение и контрол (Monitoring Service, MS).
- Услуга за сигурността и управление на потребителите (Security Service, SS).
- Услуга за управление на ресурсите (Resource Management Service, RMS).
- Информационна услуга (Information Service, IS).
- Комуникационна услуга (Communication Service, CS).
- Услуга за управление на възлите (Node Service, NS).

Всяка услуга има както клъстърна, така и глобална версия.Изключение от това правило е Node Service(услугата за управление на компютърен възел), която работи върху клиентските компютри и не се изисква да има горепосочените два варианта.



Фигура 1. Обща архитектура на грид-системата GrOSD

Фигура 1 илюстрира общата архитектура на системата, както и услугите на клъстърно и глобално ниво. Всяка услуга изпълнява своите функции локално в клъстъра, или глобално за грида, в зависимост от вида си. Клъстърните системни услуги взаимодействат помежду си в рамките на клъстъра и могат да комуникират със съответните услуги от грид нивото. Глобалните грид системни услуги взаимодействат помежду си, както и с клъстърните услуги, реализирайки по този начин грид инфраструктурата.

Основните функции, които предоставя GrOSD платформата са следните:

- библиотека от неактивни услуги
- директория от активни услуги
- структурирано описание на услугите
- управление на потребителите
- статистика за цялостното състояние на системата
- потребителски и административен графичен интерфейс, поддържан от порталите

Всеки ресурс в системата се представя чрез услуга и всяко задание за изпълнение в системата представлява услуга с входни данни и ресурси, на които да се изпълни. Това прави моделът на услугите фундаментален за архитектурата на системата.

Услугите в GrOSD могат условно да се разделят на следните типове:

- Постоянни системни услуги (Persistent System Services, PSS) – това са изброените по-горе системни услуги, които реализират грид инфраструктурата.
- Постоянни приложни услуги (Persistent Application Services, PAS) – това са услуги, които могат да се използват директно от потребителите (за разлика от системните услуги) и които са постоянно активни.
- Временни приложни услуги (Transient Application Services, TAS) – този тип услуги представят потребителско задание, което се подава на портала за еднократно изпълнение.
- Временни библиотечни услуги (Transient Repository Services, TRS) – това са услуги, които потребителите добавят в грид-системата, и които могат да се ползват от други потребители и услуги. Тези услуги не са активни постоянно, а се активират когато някой потребител иска да ги използва.

Всяка услуга (без системните) притежава структурирано описание в XML формат. То се задава от доставчика на услугата и се използва от RMS, когато услугата трябва да бъде стартирана. Описанието съдържа следните данни:

- предназначение на услугата
- типове данни, които услугата поддържа за входните си параметри
- типове данни, които услугата поддържа за изхода си
- апаратни и програмни изисквания на услугата (например операционна система, вид процесор, количество памет и др.)
- каква е цената (натоварването върху системата) за изпълнение на услугата и от какво зависи тя
- роли, необходими за изпълнение на услугата
- роли от които се нуждае самата услуга, за да се изпълни

GrOSD позволява на потребителите да създават нови услуги като свързват приложни услуги(т.е. композиция от услуги) – изходният резултат от една услуга да се използва като вход на друга услуга. Комплементирането на входа и изхода се извършва само въз основа на типа на входните и изходните данни. Платформата предоставя подходящ интерфейс за изграждането и изпълнението на такива услуги.

Следва по-подробно описание на архитектурните компоненти на GrOSD.

Грид портал и клъстърни портали

Порталите представляват входна точка за грида или за конкретен клъстър съответно. През тях потребителят може да получи достъп до системата, ако е идентифициран от услугата за сигурност. Грид порталът дава достъп до грид-системата като цяло, а клъстърните портали – до съответния клъстър. От портала потребителите могат да разглеждат наличните библиотечни и активни услуги и да ги стартират, да добавят собствени услуги и да подават задания за изпълнение.

Услуга за управление на ресурсите (RMS)

Управлението на ресурсите е от съществена важност за функционирането на грид системата. В архитектурата на GrOSD тази услуга [23] заема централно място, като си взаимодейства с всички останали системни услуги. Тя извършва планиране на заданията за изпълнение, откриване, резервиране, заделяне на ресурси, както и мигриране на задания.

На клъстърно ниво RMS (CRMS – Cluster RMS) получава заявки за стартиране на услуги от съответния клъстърен портал. От информационната услуга на съответния клъстър CRMS получава списък с наличните ресурси за изпълнение на заданието. RMS приема като критерий при търсенето на ресурси и цената за изпълнение на заданието.

Ако има свободни ресурси, услугата се стартира на тях. В случай, че резултатът съдържа повече ресурси от необходимите, тогава се избират най-подходящите на базата на тяхната употреба и статистиката за натовареността. Ако няма налични ресурси услугата може да се насрочи за по-късно изпълнение. В този случай, както и ако тя първоначално е била зададена за по-късно изпълнение, се резервират нужните ресурси и тя се поставя в опашка на чакащите задания.

След като ресурсите за заданието са заделени, отбелязва се, че те вече не са свободни и заданието се предава на съответния компютърен възел, където се намират ресурсите за изпълнение. Състоянието на изпълняващото се задание се наблюдава от услугата за контрол и наблюдение (MS).

На грид ниво RMS (GRMS) получава задания за изпълнение от грид портала. При заделяне на нужните ресурси GRMS взаимодейства с клъстърните RMS, като накрая изпраща заданието за изпълнение на съответния CRMS. Възможно е паралелно изпълнение на множество подзадания на ресурси в различни клъстъри и тогава GRMS си взаимодейства с няколко CRMS.

Информационна услуга (IS)

Информационната услуга се използва за съхранение на всички данни, необходими за работата на системните услуги. На клъстърно ниво тя управлява данните за услуги и потребители, описанията на услугите, статистически данни за използването на системата, информация за възлите и ресурсите, както и текущото им състояние, но в рамките на клъстъра. На грид ниво грид информационната услуга управлява данни, общи за целия грид. В известен смисъл тази услуга може да се разглежда като интерфейс към база от данни. Обособяването на тази функционалност в отделна услуга улеснява работата на системните услуги с данните, като ги изолира от конкретните детайли на достъп до информацията. По този начин се улеснява и използването на различни източници на данни, тъй като услугата скрива конкретния метод на съхранение.

Услуга за наблюдение и контрол (MS)

Тази услуга периодично получава информация от NS за това, че е даден възел е активен. В случай, че след изтичане на определен интервал, MS не получи информация от NS, се счита че този възел е отпаднал. Тази промяна се отразява в съответната информационна услуга и се уведомява услуга за управление на ресурсите и портала. По този начин системата разполага с актуална информация за състоянието на възлите, което улеснява откриването на прекъснали или зациклили задания, повредени възли или ресурси и други проблеми. Другите системни услуги я използват за да събират счетоводна информация или информация за състоянието на системата, която им е нужна за тяхната работа. Работещите услуги и услугите, управляващи възли, могат да информират

информационна услуга при промяна в състоянието си, информационната услуга уведомява услугата за наблюдение и контрол за настъпилата промяна.

Комуникационна услуга (CS)

Комуникацията между задания и подзадания е с голямо значение в грид-системите, особено при реализация на паралелни изчисления, където е необходимо взаимодействие между паралелно изпълняващите се процеси. Тъй като архитектурата на GrOSD се базира на услуги, комуникацията между задания се осъществява от комуникационна услуга [26], която, както и останалите системни услуги, има клъстърен (CCS) и грид (GCS) варианти. Всяко задание в системата при стартирането си получава уникален идентификатор, който се състои от идентификатор на заданието (TaskId), уникален в клъстъра, и идентификатор на клъстъра (ClusterId), уникален в системата. Когато RMS разпредели някое задание за изпълнение на даден възел, тя създава съответствието между възела и идентификатора на заданието и това съответствие се пази от CCS. Останалите задания знаят само идентификатора и ако искат да комуникират с това задание, те пращат съобщението до CCS, която вече го препраща до съответния възел. Ако е необходима комуникация между задания в различни клъстъри, то CCS препраща съобщението до GCS, която знае в кой клъстър да го изпрати. В случай на миграция на задание от един възел на друг, или между клъстъри, RMS, която извършва миграцията, актуализира съответствието между идентификатора на заданието и правилния възел в съответната комуникационна услуга (CCS или GCS, в зависимост от това дали миграцията се извършва в рамките на един клъстър или между клъстъри).

Услуга за управление на възлите (NS)

Тази услуга се изпълнява на всеки възел в грид-системата. Тя прави един възел част от даден клъстър и съответно от GrOSD. NS следи ресурсите(хардуерни и софтуерни) на възела и обновява статистически данни и информация за състоянието му в информационната услуга(IS). NS се грижи за изпълнението на изпратените от RMS задания, както и за това да върне резултата от завършване на заданието обратно на RMS. Тази услуга предлага възможност на останалите системни услуги(дирекно или индиректно) да следят изпълнението на задания, да ги спират временно или да ги прекратяват и да откриват задания,

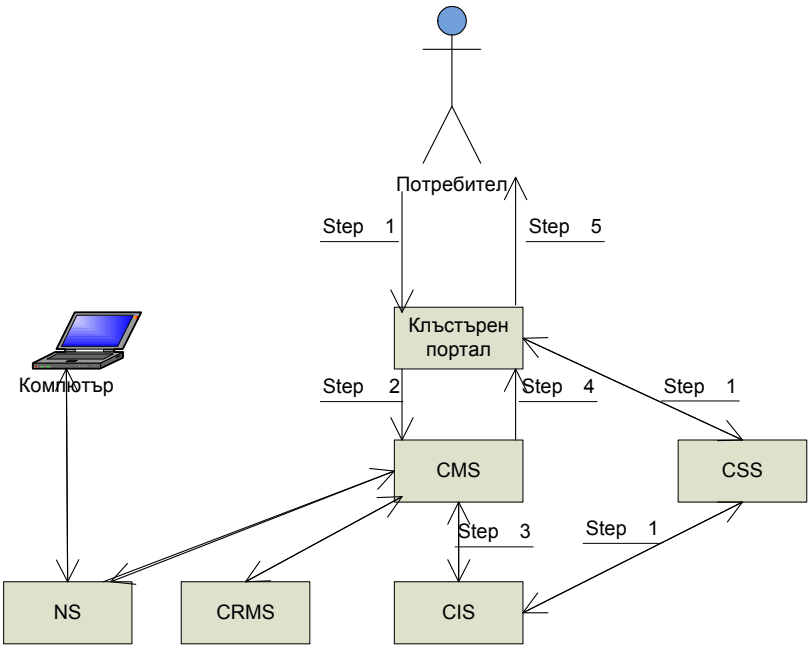
които е нужно да бъдат рестартирани или преместени на друг възел(поради прекалена натовареност на компютърният възел например).

NS няма варианти за клъстърно и грид ниво – тя работи единствено при компютърните възли.

2.2. Сценарии на използване на GrOSD

За илюстрация на тясното взаимодействие на различните системни услуги при изпълнение на задачи в описваната грид-система се разглеждат три основни сценария на използване на системата от потребител – изпълнение на потребителско задание (временна приложна услуга), използване на грид услуга (постоянна приложна услуга) и получаване на информация за ресурси. Както се вижда, работата, извършена от системата в първите два случая е доста сходна.

Примери за изпълнение на потребителско задание



Фигура 2. Изпълнение на потребителско задание

На Фигура 2 се илюстрира последователността от стъпки които системата изпълнява при подаване на потребителско задание.

Следват стъпките, които се извършват, когато потребител подава задание в GrOSD за изпълнение (разглежда се случая, когато потребител се идентифицира на клъстърен портал):

1. Преди всичко, потребителят трябва да се идентифицира в системата. След това той разполага с удостоверение, съдържащо неговата идентичност и роли и може да подаде задание.
2. След подаване на заданието, порталът се свързва с CRMS, която отговаря за планиране на изпълнението на заданието върху конкретни ресурси.
3. CRMS се обръща към CIS, за да получи списък със свободни ресурси. CIS търси ресурси, които отговарят на поставените критерии, като при това филтрира според потребителските роли, и връща на CRMS само тези, до които съответният потребител има достъп.
4. След като получи списъка, CRMS решава кои ресурси ще използва (ако са повече от необходимите) и разделя заданието на подзадания, в случай, че заданието допуска паралелно изпълнение.
5. CRMS се обръща към NS на възлите, където се намират избраните ресурси, за да провери дали наистина са свободни и дали няма допълнителни ограничения върху ролите на потребителя.
6. NS проверява при CSS дали потребителското удостоверение не е фалшиво и стартира заданието.

Ако потребител се обърне към грид портала, тогава действията, извършени от системата, са аналогични. В този случай грид порталът се свързва с GRMS, който отправя заявка за ресурси към GIS. GIS взаимодейства със CIS на различните клъстъри, за да намери ресурси. След като GRMS реши кои ресурси ще използва, тя препраща заданието на съответните CRMS, които го предават за изпълнение на NS.

Използване на услуга

Освен да подават задания за изпълнение потребителите могат да ползват готови услуги. След като потребителят се идентифицира успешно, той може да

разгледа списък с достъпните услуги. Потребителят избира услугата, която иска да използва, въвежда необходимите входни данни и клъстърният портал предава заявката към CRMS. Оттук нататък стъпките са същите като при подаване на задание. Ако потребителят се е свързал с грид портала, тогава заявката се обработва от GRMS, аналогично на предходния случай.

Информация за ресурси (услуги, задачи, възли):

Потребителите могат да получават информация за отделни услуги, задачи и възли. Потребителя се идентифицира успешно на портала и иска да получи информация за състоянието на определен ресурс. Порталът се обръща към CMS за информация, CMS извлича необходимата информация от CIS и я предава обратно на портала. Ако потребителят се е свързал с грид портала, тогава заявката се обработва от GMS.

Потребител се идентифицира на клъстърен портал:

1. Преди всичко, потребителят трябва да се идентифицира в системата. След това той разполага с удостоверение, съдържащо неговата идентичност и роли и може да подаде искане за информация.
2. След подаване на заданието, порталът се свързва с CMS, която отговаря за информацията за конкретни ресурси.
3. CMS се обръща към CIS, за да получи списък с исканите данни. CIS търси ресурси, които отговарят на поставените критерии,.
4. След като получи списъка, CMS предоставя информацията на портала
5. Порталът предоставя данните на потребителя.

2.3. Технологии използвани при реализацията на "Резидентен модул за управление на възлите в йерархичен грид"

Тъй като в частта описваща имплементацията на услугата за управление на възлите в йерархичен грид не бяха описани подробно използваните технологии, езици за програмиране и програмни средства, то тази част на настоящата глава го прави.

Java – основно използваният програмен език

Едно от най-важните изисквания към услугата за управление на компютърен възел (и към цялата грид-система GrOSD) е платформената независимост. В грид трябва да могат да се включат най-разнообразни хостове(компютри), както по отношение на хардуеърната архитектура, така и по отношение на софтуеърната платформа работеща на тях. Това изисква максимална преносимост на грид мидълуеъра – той трябва да работи на колкото е възможно повече хардуерни и софтуерни платформи. Поради тази причина за реализация на системата GrOSD, и съответно на системната услуга за управление на компютърен възел, беше избран програмният език Java. По този начин системата може да работи на всяка платформа, за която има създадена Java виртуална машина, без да се правят промени в кода(Това обаче е идеалният случай.Де факто се получава, че пишеш веднъж и тестваш навсякъде, все пак е по-рационално от пълно пренаписване, но миграцията съвсем не е толкова гладка). Тъй като Java е широко разпространена и широкоподдържана, тя е логичният избор за такъв тип приложения.

C++ - програмният език за който няма ограничения

Тъй като Java приложенията са платформено независими, Java като платформа не предоставя методи / начини за придобиването на така важната за модула функционалност, а именно взимане на информация за свободни ресурси(RAM , CPU натоварване, дисково пространство, информация от Window регистрите т.н.) и се наложи използването на езика C++.

Чрез езика C++ беше написан програмен модул за взимане на системни параметри на компютър с активна операционна система Windows, в резултат от който се получи dll(dynamic link library).Функционалността от получения dll беше енкапсулирана чрез Java технологията JNI(Java Native Interface), за да бъде използвана от основният модул написан на Java.

Платформата Java

Освен преносимостта, Java разполага и с редица други предимства. На първо място Java не е само език за програмиране, а е цялостна платформа, представляваща набор от технологии, които предлагат разнообразни услуги, като се започне от средства за обработка на изображения и звук и се стигне до

мрежови комуникации и сигурност. Голямата разпространеност на платформата е причина и за много налични библиотеки от трети страни, които не са включени в платформата, но могат да се използват. Не на последно място се нарежда и фактът, че Java платформата е безплатна. Тъй като GrOSD е академичен проект, едно от важните изисквания е той да бъде изграден с лицензно свободни езици и средства. При реализацията на услугата за управление на компютърен възел е използвана jdk1.5.0_06.

Платформата J2EE

J2EE (Java 2 Enterprise Edition) представлява платформа, задаваща стандарт за създаване на преносими, разпределени, многослойни сървърни приложения на езика Java. Тя включва множество технологии, които надграждат J2SE (Java 2 Standard Edition) и спомагат разработването на разпределени приложения. Някои от тях са Enterprise JavaBeans (EJB), Java Naming and Directory Interface (JNDI) – средство за достъп до директорийни услуги, Java Database Connectivity (JDBC) – API (Application Programming Interface) за достъп до бази от данни, Java Transaction API и Java Transaction Service – осигуряват надеждни транзакционни механизми, Java Mail – пакет за работа с електронни писма, и др. J2EE предлага компонентен модел за разработка на приложения. Важно място в този модел заемат контейнерите за компоненти, в които се изпълняват компонентите. Контейнерите предлагат интерфейси и услуги на работещите в тях компоненти. Тези интерфейси и услуги са стандартизирани в J2EE. Контейнерите за компоненти се предлагат от приложните сървъри (application servers).

За разработката на GrOSD беше избрана именно J2EE платформата, тъй като моделът на разработка отговаря на изискванията на разработваната система. Грид платформата GrOSD предлага основните си функции като системни услуги, които могат да се разработят като разпределени компоненти. Освен това, с множеството услуги, които J2EE предлага, тя улеснява разработката на приложенията – програмистът може да се абстрахира от технически детайли като синхронизация на нишки, поддържане на пул от обекти и др. – те се поемат от контейнерите за компоненти.

При реализацията на услугата за управление на компютърен възел е използвана версията J2EE 1.5

Enterprise JavaBeans (EJB)

Прегледа на EJB е по-скоро отразен поради факта, че се използва като компонент от останалите системни услуги в GrOSD. В услугата за управление на компютърен възел се налага извикването на експортнатите интерфейси, които също са обяснени в тази подсекция.

EJB представляват компонентна архитектура, която заема централно място в J2EE платформата. Чрез EJB компоненти обикновено се реализира логиката на разпределеното приложение. Те се зареждат и изпълняват от приложен сървър, който им предлага множество услуги и управлява изпълнението им.

Има три вида EJB компоненти:

- Session – реализират логика, свързана с процесите в приложението, например изчисления и т.н.
- Entity – реализират логика, свързана с достъп и работа с данни.
- Message – реализират комуникация, базирана на съобщения.

За разлика от останалите системни услуги в GrOSD, услугата за управление на компютърни възли не използва EJB компоненти.Тя изполва различните начини за повикване на тези компоненти(т.е. извикването на интерфейси на EJB-тата, които изграждата останалите системни услуги, тъй като се налага комуникация с тях).

Услугата за управление на компютърни възли използва главно обикновени Java класове. За комуникация с другите системни услуги се използва RMI(Remote Method Invocation).

Тук е момента да се обоснове защо се наложи да не се използва EJB модела като основа на услугата за управление на компютърен възел.Това се наложи поради факта, че използването на EJB компоненти би изискало наличие на EJB контейнер – например TomCat или Catalina. Това изискване е твърде ограничаващо, тъй като услугата за управление на компютърен възел работи на клиентски компютри, за разлика от останалите системни услуги, които работят на специално определени компютри.

Session EJB се делят на Stateful (пазят вътрешно състояние между извикванията на методите си) и Stateless (нямат вътрешно състояние).

Методите на EJB компонентите не се извикват директно от потребителите. Извикванията се прихващат от контейнера и се пренасочват към съответния компонент. По този начин контейнерът добавя прозрачно своите услуги като поддържане на пул, транзакции и др. Така че клиентите извикват методи на EJB обект, вместо на обект от класа, реализиращ компонента. Този обект се генерира автоматично от контейнера. В него се включват методите на компонента, които са достъпни отвън. Тези методи се включват в интерфейс, наречен remote interface. EJB обектите не могат да се инстанцират директно от кода, който ги използва. За получаване на референс към EJB обект се използва EJB home обект. Той се грижи за създаването и унищожаването на EJB обекти и също се генерира от контейнера. EJB home обектът получава методите, необходими за създаване на конкретния EJB обект от специален интерфейс, наречен home interface. EJB обектите и EJB home обектите се използват в мрежова среда, когато клиентският код и компонентите, които той използва работят в различни виртуални машини. Тези обекти се грижат за всички детайли, свързани с мрежовата комуникация. Освен тях са налични и техни локални аналози – EJB local обект (съответно реализиращ local interface) и EJB local home обект (съответно реализиращ local home interface). Те са предназначени за случаите, в които викащият код и използваните компоненти работят в границите на една и съща Java виртуална машина и предлагат по-висока производителност.

При реализацията на системните услуги в GrOSD, които използват EJB архитектурата е използвана версията EJB 2.1.

RMI(Remote Method Invocation)

Тази технология на Java беше изпозвана, за да могат останалите системни услуги да комуникират с услугата за управление на компютърен възел(използват се stub-skeleton модела).Наложително беше използването на RMI, тъй като тя не изисква допълнителни модули извън стандартната Java и не се нуждае от допълнителни ресурси.Алтернатива беше създаването на тази услуга като EJB(както бяха създадени останалите системни услуги), но това щеше автоматично да ни задължи на работещите компютърни станции да работи приложен сървър, което е в противовес на концепцията за лесно използване и минимално “замърсяване” на предоставената ни от крайният потребител среда.

JNI(Java Native Interface)

Това е библиотека към Java средата, която позволява на Java класове да извикват методи написани на други програмни езици като C++, C, Pascal например. За тази цел се прави dll(Dynamic link library) написана на някои от горепосочените езици(в която имената на методи и класове следват специални конвенции), с цел тези класове със специфичните си методи да се енкапсулират в Java класове и да може тяхната функционалност да бъде свободно използвана в Java програмата.

Инструменти за разработка на Java частта от модула

Основно изискване към инструментите, използвани за реализацията на GrOSD е те да са с лиценз за свободно ползване.

Една от лесните за използване и изискваща минимум хардуерен ресурс среда за разработка е JCreator Light. Това е безплатният вариант на JCreator Pro.

Тази програмна среда е чудесна за изграждането на по-обикновени(в смисъла на неизползващи по-сериозните технологии на Java) Java приложения.

Една от най-добрите свободни развойни среди за Java е платформата на IBM Eclipse [36]. Тя е разширяема и поддържа много добавки (plug-ins) на трети страни. Поради тези причини тя беше избрана за реализация на системните услуги в GrOSD(за услугата за управление на компютърни възли тази среда бе важна в етапа на интеграция, а не на имплементация, в който много по-често бе използвана JCreator Light).

Важна добавка към Eclipse, която се използва при интеграцията на услугата, е JBoss Eclipse IDE [28]. Тя е създадена специално за разработка на J2EE приложения на Eclipse, и също така е свободна. Тази развойна среда значително улеснява разработването на EJB компоненти, като генерира автоматично всички интерфейси на компонентите на базата на класа, реализиращ компонента и също така генерира нужните XML дескриптори. За това тя използва програмата XDoclet [30].

Конкретно при реализирането на услугата за управление на компютърни възли не се наложи използването на Eclipse или JBoss application server(приложен сървър), но използването им беше наложително при интеграцията на модула в проекта, за работата с другите системни услуги.

Инструменти за разработка на C++ частта

За разработването на компонентите имплементирани чрез програмният език C++ беше използвана програмата среда MS Visual Studio 6.0. Тази програмна среда е безплатна при използване за научни и изследователски цели(след като през 2006 беше спряна нейната поддръжка от страна на Microsoft е напълно възможно да е напълно безплатна за всякакви нужди вече).

Програмната среда VS 6.0 заема малко място, компилира програмите много бързо и прави изпълними файлове с минимален размер и максимална ефективност.

3. Проектиране и реализация на модул за управление на възлите в GrOSD

Целта на тази глава е да опише проектирането и програмната реализация на модула за управление на възлите на платформата GrOSD. Услугата за управление на компютърен възел трябва да реализира описания в глава 2 модел на услугата. Проектирането на останалите системни услуги е предмет на други дипломни работи, а не е предмет на това изложение.

Основната цел е изграждането на работещ прототип на системната услуга за управление на компютърен възел, който в последствие да бъде подобрен и разширен. Той трябва да включва базовата функционалност, необходима за реализация на разработения модел на управление на компютърен възел.

Тъй като системната услуга за управление на компютърен възел работи на клиентските компютри, то тази системна услуга няма клъстърна и / или грид версия.

Унифицираният процес за разработка на софтуер(Unified Process) е използван при проектирането на системната услуга, тъй като предоставящ обектно ориентиран анализ и дизайн.Проектиране като цяло се придържа към методологията.

Поради спецификата на заданието, някои традиционно съпътстващи документи са пропуснати(в смисъла на не напълно развити или просто опростени).Например, ако това приложение беше комерсиално, то щеше да включва

proposal document, specification document, understanding document, detailed design document. Но поради факта, че няма краен клиент и ние определяме спецификациите на модулите, голяма част от документите липсват или са нагодени към средата в която трябва да бъдат използвани.

Разработваната софтуерна услуга е част от системен middleware(бук. посредник, свързващо звено), което налага автоматично едно по-високо ниво на абстракция(проблемната област не се отнася директно към реалният свят), за разлика от разработването на приложен софтуер за директно използване от потребителя. От друга страна използването на модулен принцип позволява лесна промяна на реализацията, което да не довежда нуждата от промени в другите системни услуги.

3.1. Проектиране

Представяне на високо ниво

Тази точка има за цел да представи на високо ниво функционалността на разработваната услуга, като по този начин обобщава изложените в описанието на модела на управление на компютърни възли в GrOSD изисквания.

Увод

Настоящият проект предвижда изграждане на подсистема (системна услуга), която да осигурява адекватно управление на компютърните възли, като предоставя актуална информация както за хардуерни и софтуерни параметри на компютърният възел така и информация за работещите услуги и задания на него. През определен интервал от време услугата уведомява услугата за наблюдение(MS) за това,че компютърният възел работи. Взаимодействие има още и с услугата за управление на ресурси(RMS), с информационната услуга(IS) и с услугата за сигурност(SS).

Позиция

Съществуват множество готови решения за управление на компютърни възли в грид. Повечето от тях обаче не са ясно разграничени като функционалност(т.е. да правят единствено и само управление на компютърен възел), също така не се спазва единен дизайн, който би спомогнал за преизползването на услугата за управление на компютърните възли от една грид-система в друга. Платформата GrOSD се отличава със специфична йерархична и клъстъризирана архитектура,

базирана на услуги, която има ясно определени интерфейси за взаимодействие между услугите. Това изисква създаването на системна услуга за реализация на този модел, която е специално проектирана в съответствие с нуждите на платформата GrOSD.

Заинтересовани лица и техните цели на високо ниво

Системата за управление на компютърен възел е важна за:

- Потребители на грид платформата (потребители на услугите) – коректното извличане на информация за компютърният възел, пускането на потребителската задача навреме и връщането на коректен резултат от заданието е основна задача, поради която потребителите изобщо използват грид-системи.
- Администратори – тяхна цел е гарантират правилното функциониране на системата, това включва и равномерно натоварване на компютърните възли. Без актуална информация за хардуерните и софтуерни параметри на компютърните възли RMS (услугата за управление на ресурси) не би могла коректно да разпределя заданията за изпълнение.

Конкретни цели на потребителите

Специфично за разработваната система е, че нейните преки потребители не са потребителите на платформата GrOSD, а други системни услуги. Следват услугите, които ползват услугата за управление на компютърни възли, заедно с техните цели:

- клъстърна услуга за управление на ресурсите (CRMS) – заделя ресурси за изпълнение на определена задача и когато тя стартира уведомява CMS, а услугата за наблюдение информира портала.След като определи заданията върху кои компютри ще се изпълнят, тя уведомява NS(Node Service) на конкретния компютър кои са заданията, които трябва да се изпълнят при него, кога трябва да се изпълнят и какви са им входните и изходните параметри.След завършване на заданието резултата се връща от NS към CRMS.

- услуга за наблюдение (MS) – през определен интервал от време се проверяват работещите възли и в случай на отпадане на някой от тях, промяната се отразява в CIS, а CMS уведомява портала и CRMS
- клъстърна информационна услуга (CIS) – при заявка от портала за предоставяне на информация, порталът се обръща към CMS, а услугата за наблюдение извлича съответната информация от CIS. При стартиране на отложено задание или при отпадане на възел CMS променя статуса на задачата или възел в CIS. При доброволното напускане на възел, той уведомява за промяната CIS, а тя от своя страна ще информира услугата за наблюдение и контрол за настъпилата промяна.

Основна функционалност на "Резидентен модул за управление на възлите в йерархичен грид"

- да определя софтуерни и хардуерни параметри на компютърния възел на който е инсталиран
- да стартира, спира или забавя изпълнението на потребителска задача.
- да предоставя статус на задача както и резултата от изпълнението й.
- да уведомява системната услуга за наблюдение(Monitoring Service) през определен интервал от време, че компютърният възел продължава да функционира.
- да комуникира с останалите системни услуги, за да получава или предава информация(Security Service, Resource Management Service, Information Service, Monitoring Service)..

Допълнителни изисквания и ограничения

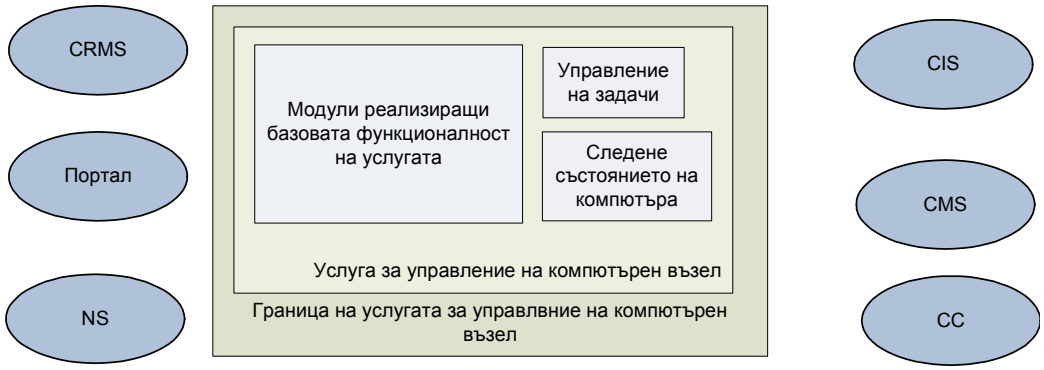
Включват лесна разширяемост, преносимост, надеждност и т.н. (по-подробно са изложени в точка Допълнителна Спецификация).

Изисквания

Тази секция описва функционалните изисквания към разработваната системна услуга. За изясняване на тези изисквания се използват потребителски сценарии (Use Cases).

Граници на системата

При разглеждане на системната услуга за управление на компютърен възел, границите на тази подсистема са ясно дефинирани. Тази услуга ще се използва пряко от услугата за управление на ресурсите (RMS), от услугата за наблюдение(Monitoring Service). От своя страна, услугата за управление на компютърен възел ще използва информационната услуга(Information Service) и услугата за сигурност (Security Service). Така изброените системни услуги определят границите на разглежданата подсистема. В нея остават модулът за управление на задачи, модула за следене на състоянието на компютърният възел както модулите реализиращи базовата функционалност на услугата.



Фигура 3. Граници на услугата за наблюдение и контрол

Чрез фигура 3 се илюстрира системната услуга за управление на компютърен възел, с нейните граници.

Потребители и цели

В следващата таблица са обобщени потребителите(тук под потребители разбираме други системни услуги) на услугата с техните цели при използването ѝ. Потребителите и целите им определят функционалните изисквания към разработваната услуга.

Потребител	***Цел***
RMS	Възлага изпълнението на потребителски задачи на NS.
MS	Проверява състоянието на компютърните възли като изисква информация от NS през предварително определен интервал от време или се уведомява от NS при настъпване на определено събитие.
IS	Получава информация за състоянието на възлите

	от NS, която информация в последствие изпраща на услугата за наблюдение и контрол(MS)
SS	NS прави запитване(проверява правата на потребителския, чието задание ще се изпълнява) към IS с цел да се удостовери, че потребителят има необходимите роли, за да бъде неговото задание изпълнено.

Таблица 2. Потребители с техните цели

Реализация на системната услуга за управление на компютърен възел

В тази точка се разглежда програмната реализация на услугата. Следва списък на софтуеърните класове с кратко описание на всеки от тях. Описват се отговорностите на всеки клас. В изложението не са включени пакетите на другите системни услуги, които не са част от тази дипломна работа макар и да са обединени от това, че са част от една и съща грид-система GrOSD.

Класове и пакети написани на Java

Пакет **bg.unisofia.fmi.grosd.core.nodeservice**

Този пакет съдържа класове и интерфейси, свързани с услугата за управление на компютърен възел.

Compiler.java – съдържа клас, който енкапсулира функционалност на Java компилатор

NodeTaskManager.java – съдържа клас, който отговаря за управлението(следене на изпълнението, стартиране на заданието в определено време и т.н) на заданията, които се изпълняват на компютърния възел.

NS.java – съдържа клас, който представлява един компютърен възел,

който включва компонент за хардуера и компонент отговарящ за изпълнението на заданията.

NSServiceIF.java – съдържа интерфейс към услугата за управление на възли

NSTaskThread.java - съдържа клас, който представлява thread (нишка), която се грижи за извършване на операции върху различните задания през определен интервал от време.

Zipper.java – съдържа клас, който предоставя функционалност за архивиране / разархивиране

Пакет **bg.unisofia.fmi.grosd.core.nodeservice.hardware**
Този пакет както показва и името му съдържа класове, които представляват различни хардуерни компоненти или предоставят средства за извличането на хардуерна информация.

CommunicationProperty – съдържа клас, представляващ комуникационни параметри на средата за пренос на данни

CPU – съдържа клас, представляващ един процесор

HardDisk - съдържа клас, представляващ един твърд диск

HardStorage - съдържа клас, представляващ трърдата памет в компютъра

LanCard - съдържа клас, представляващ мрежова карта

NodeHardware - съдържа клас, представляващ агрегатен клас, който съдържа информация за хардуера в компютъра

OperationSystem - съдържа клас, представляващ операционната система на компютъра

RAM - съдържа клас, представляващ RAM паметта в компютъра

SysDynInfoCAPI - съдържа клас, представляващ обвивка на C++ функционалността за извличане на актуална хардуерна информация за компютъра.

SystemDynamicInfo - съдържа клас, който дава информация за различни параметри на средата като пътища на инсталация, име на потребителска директория, информация за инсталираната Java и JVM и др.

UnknownHardware - съдържа клас, представляващ неизвестен към момента на писането на модула хардуер, ще се използва за бъдеща употреба.

VGAAdapter - съдържа клас, представляващ графичната карта на компютъра.

Пакет **bg.unisofia.fmi.grosd.core.nodeservice.rmi**

Този пакет е натоварен със задачата да осигури комуникацията от страна на другите системни услуги към услугата за управление на компютърен възел.Както беше обяснено тук не бе удачно използванеот на EJB архитектура(поради изискването на минимална промяна в средата на клиента), затова беше избрана RMI технологията, която отговаря на изискването за минимално натоварване на клиентската машина и добавянето на никакъв допълнителен софтуер(извън стандартната Java).

RMINSInterface - remote(отдалечен) интерфейс за компютърния възел

RMINS – remote(отдалечен) обект представляващ компютърния възел

RMINSClient – обект представляващ клиентската част при RMI комуникацията.Използва се там, където работят останалите системни услуги.

RMINSServer– обект представляващ сървърната част при RMI комуникацията.Работи при системната услуга за управление на компютърен възел

Пакет **bg.unisofia.fmi.grosd.core.nodeservice.service**
Този пакет съдържа класове, които описват услуги или задания.

Service – съдържа клас представляващ услуга

Task - съдържа клас представляващ задача

TaskStatus – съдържа клас, представящ статус на задача

Пакет **bg.unisofia.fmi.grosd.core.nodeservice.ui**
Този пакет съдържа няколко потребителски интерфейсни класове, целта е визуализация на състоянието на компютърният възел

CCPUPage – страница показваща процесора(-ите)

CHardStoragePage – страница показваща твърдият диск

COperatingSystemPage – страница показваща всички особености на операционната система

CPage – базов клас, който се наследява от другите потребителски страници

CRamPage – страница показваща информация RAM паметта

CTasksMonitorPage – страница за следене на задачите

JobDeployerUI –страница за тестване на ръчно стартиране на задание

NodeUI – основният контейнер, в който се съдържат останалите страници, показващи информация за различни параметри компютърният възел.

Пакет **bg.unisofia.fmi.grosd.core.nodeservice.useful**
Това е пакет, съдържащ допълнителни класове, чиято функционалност е полезна.

CErrorLogger - клас записваю системни грешки, с цел добиване на представа защо системата неработи както се очаква

CFileFuncs – клас за файлови операции.

CStringFuncs- клас за операции с низове

InterfaceForConstants- интерфейсен клас за константи

MyWinRegWrapper – клас обвивка на Windows Registry

Класове и пакети написани на C++

Проект с име "JNI", чията цел е създаването на dll, която да бъде заредена чрез JNI в java клас(SysDynInfo.java), където да бъде използвана функционалността и.

Създаването на проект написан на C++ и използването на JNI се наложи поради факта, че заради преносимостта на Java тя не предоставя начин за извличането на софтуерни и хардуерни параметри на системата.

Точно за това се наложи създаването на "JNI", за да могат чрез извиквания на функции на Windows API да бъдат определени софтуерни и хардуерни параметри на системата, които да бъдат използани от софтуерния модул за управление на компютърния възел.

Проекта "JNI" включва:

Стандартни файлове за MS VC 6.0 dll проект
StdAfx.h

StdAfx.cpp

Readme.txt

Специфични файлове за проекта

DynamicInfo.h - header file (заглавен файл) за CDynamicInfo класа.

DynamicInfo.cpp – имплементационен файл за CDynamicInfo класа.

JNI.cpp - определя входната точка за dll приложението

Класа CDynamicInfo има следната декларация:

```
class CDynamicInfo
{
public:
	//retrieves information about RAM
	static void GetRamInfo(  FILE* a_pFileOutput  );

	//retrieves information about the user that uses the computer
	static void QueryUserName(  FILE* a_pFileOutput  );

	//retrieves information about computer name
	static void QueryComputerName(  FILE* a_pFileOutput  );

	//retrieves information about Hard disk drives
	static void QueryHardDisks( FILE* a_pFileOutput );

	//retrieves information about drive space
	static void QueryDriveSpace( FILE* a_pFileOutput );

	//Constructor / Destructor
	CDynamicInfo();
	virtual ~CDynamicInfo();
};
```

В JNI.cpp освен стандартните методи съдържа и метода, който ще бъде експортиран през JNI за използване в Java:

```
JNIEXPORT                          void                          JNICALL
Java_bg_unisofia_fmi_grosd_core_nodeservice_hardware_SysDynInfoCAPI_U
pdateInformationFile(JNIEnv *env, jobject obj , jstring name)
```

Това е метода, заради който е изграден целият проект.

Името на метода е сложно поради факта, че използването на JNI изисква определено именоване на методите и променливите(иначе съответния метод няма да бъде зареден от JNI).

Името на метода означава:

а) че този метод ще се използва от Java JNI

b) че пакета е bg.unisofia.fmi.grosd.core.nodeservice.hardware

c) че класа се казва SysDynInfoCAPI

d) че името на метода е UpdateInformationFile

e) първите 2 аргумента са служебни за JNI, първият е указател към Java виртуалната машина, а третият аргумент показва, че реално Java метода ще приема точно един аргумент(от тип String).

4. Инсталиране и интеграция в системата

Тази глава разглежда дейностите, свързани с внедряването на "Резидентен модул за управление на възлите в йерархичен грид", извършени след приключване на разработката.

4.1 Инсталиране

Инсталирането на самият програмен модул е достатъчно просто.

Класовете, интерфейсите се включват в NS.jar(Node Service.jar). Този файл се копира в съответната директория на приложният сървър(в конкретният случай JBOSS), с което самата инсталация приключва.След стартиране на сървъра услугата е достъпна за използване от останалите системни услуги на GrOSD.

4.1.1 Стартиране на услугата при компютърният възел

За стартирането на услугата при компютърният възел е предоставен олекотен интерфейс, позволяващ настройката на някои параметри на услугата.Стартира се сървъра на RMI(имплементиран в IRMNSServer)

4.1.2 Стартиране на услугата при компютри, на които са инсталирани и работят системни услуги

Както беше посочено по-горе в документа, за да не се налага допълнителното условие за инсталиране на приложен сървър при компютърните възли, услугата за управление на компютърните възли не използва EJB.Поради тази причина, за да може да се осъществи комуникацията между останалите системни услуги и

услугата се управление на компютърни възли се използва IRM технологията, посредством която става комуникацията.Стартира се клиента на RMI(IRMNSClient).

4.2. Интеграция

Услугата за управление на компютърен възел взаимодейства с:

- услугата за наблюдение(на която тя предава информация за състоянието на компърният възел).
- услугата за управление на ресурсите(на която връща резултата от работата на изпълнената задача и която услуга и възлага изпълнението на дадена задача).
- услугата за сигурност(чрез която преди изпълнението на задание се проверява дали потребителят наистина има права да го изпълни).
- информационната услуга, в която се запазва различна информация относно компютърният възел.

Използването на компонентен модел при изграждането на системните услуги в GrOSD с ясно дефинирани интерфейси за взаймодействие за взаимодействие спомага за лесната замяна / подмяна на функционалност в различните и компоненти.

Това спомага всяка(в това число и услугата за управление на компютърен възел) системна услуга в GrOSD да има няколко различни реализации, които лесно да бъдат подменяни, тъй като са ясно дефинирани интерфейсите за комуникация, които те трябва да имплементират.

Резидентен модул за управление на възлите в йерархичен грид е минимално обвързан с останалите системни услуги на базата на няколко класа, служещи като интерфейс за комуникация с останалите системни услуги, затова и интеграцията им е проста и не предизвиква никакви проблеми.

Заключение

Основната цел на дипломната работа е написването на резидентен софтуерен модул, който да управлява възлите в йерархичен грид (по-конкретно в грид-системата GrOSD).

Тази цел беше постигната чрез участие в цялостния дизайн на грид-системата GrOSD както и конкретно в проектирането и реализацията на услугата за управление на компютърен възел.

В глава 1 беше направен обзор на леките гридове, дадена бе дефиниция за леки гридове. Включен бе и преглед на механизмите и технологиите използвани за реалиазацията на услугата за управление на компютърни възли, макар и да има грид-системи в които тя да не е толкова ясно обособена като отделна услуга.

В глава 2 беше представен концептуалният модел за осигуряването на управлението на компютърните възли в системата, взимайки под внимание спецификата на системата GrOSD(йерархичен, многослоен грид, който е съставен от клъстъри и базиран на услуги).

В глава 3 беше представен проекта и конкретната реализация на системната услуга.За реализацията е използвана платформата J2EE , технологията RMI(Remote Method Invocation), JNI(Java Native Interface). Компонентите, които изграждат системната услуга са описани в тази глава и реализират цялата функционалност на системната услуга, а именно комуникация с останалите системни услуги, следене на хардуерните и софтуерни параметри на компютърния възел, както и управление на работещите задачи на съответният компютърен възел.

Основно изискване за разработката на програмни модули за грид-системата GrOSD беше използването на некомерсиални продукти.Това условие беше изпълнено с използването на некомерсиални програмни продукти и платформи като: платформата J2EE, средите JCreator Light, Eclipse, MS Visual Studio 6.0(безплатен за научни и учебни цели).

Използването на Java гарантира тяхната преносимост, но както беше обяснено по-горе има и ограничения, в конкретният случай на услугата за

управление на компютърни възли невъзможност за определяне на системни хардуерни и софтуерни параметри.

Именно това наложи използването на езика C++, с чиято помощ бяха определени тези параметри, а чрез Java технологията JNI(Java Native Interface) те бяха използвани от основният Java модул.

Използваната компонентна архитектура улеснява надстройката и разширяването на системата с нова функционалност.

Възможни бъдещи разширения

Резидентен модул за управление на възлите в йерархичен грид е резултат от първия етап на проекта СУГрид.Основна цел е създаването на първоначален работещ вариант на системните услуги, които в последствие да бъдат усложнени и подобрени, ако се налага. Както беше посочено по-горе в документа, използването на модулност и ясните интерфейси за комуникация между услугите правят лесно замяната на един вариант на системните услуги с други.

Резидентен модул за управление на възлите в йерархичен грид би могла да бъде разширена по следния начин:

- както беше отбелязано по-горе използването на платформено независимата среда и език Java носи и своите проблеми – невъзможност за определяне на системните софтуерни и хардуерни параметри на системата. В конкретната реализация, беше наблегнато на операционната система Windows. За вбъдеще може да бъде добавена функционалност за определяне на софтуерните параметри на Linux, Unix, Mac, Solaris и др.

- може да бъде разширена функционалността за изпълнението на потребителски задачи.В момента може да бъде изпълнен изпълним файл или java клас или jar файл, в последствие може динамично да се зареждат и изпълняват определени java класове или методи, както и да се предостави възможност за паралелност при изпълнението на програмата.

Приложение А

В това приложение е включен речник на използваните в дипломната работа термини.

Actor – действащо лице

Adaptability - адаптируемост

Communication Service – комуникационна услуга

Detailed design document – документ за детайлен дизайн

Dll(Dynamic Link Library) – библиотека, която се зарежда динамично

Fault tolerance - отказоустойчивост

Header file – заглавен файл

Information Service – информационна услуга

Jar(Java archive) – Java архив

JNI(Java native interface)

JVM(Java virtual machine) – java виртуална машина

Loose coupling – минимално обвързване

Monitoring Service – услуга за наблюдение

Node Service - услуга за управление на компютърен възел

Proposal document – документ предложение за възлагане на проект

Resource Management Service – услуга за управление на ресурси

RMI(Remote Method Invocation) – отдалечени викане на методи

Security Service – услуга за сигурност

Service Oriented Architecture - ориентирана към услугите

Specification document – документ специфициращ проекта

Thread – нишка(един процес се състои от много ниюки)

UML – Unified Modeling Language

Understanding document – документ представящ разбирането на разработчиците относно спецификациите, които изисква клиента

Unified Process - Унифицираният процес за разработка на софтуер

Use Cases - потребителски сценарии

Windows Registry – регистри на Windows операционната система.

Библиография

1. I. Foster and C. Kesselman. The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufman Publishers 1999, ISBN 1558604758
2. I. Foster, C. Kesselman and S.Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. IJSA, 15(3), Sage Publications, USA , 2001
3. Globus Toolkit, www.globus.org/toolkit
4. Core Grid Technologies
5. The Open Grid Services Architecture, Version1.0, http://www.ggf.org/documents/GFD.30.pdf
6. R. Badia, O. Beckmann, M. Buback, D. Caromel, V. Getov, S. Isaiadis, V. Lazarov, M. Malawski, S. Panagiotidi, J. Thiyagalingam. Lightweight Grid Platform: Design Methodology. In Proceedings of GRIDS@Work, Sophia Antipolis, France, October 10 – 14, 2005
7. Y.M. Teo and X. B. Wang. AliCE: A Scalable Runtime Infrastructure for High Performance Grid Computing. In Poceedings of IFIP International Conference on Network and Parallel Computing, Springer-Verlag Lecture Notes in Computer Science, Wouhan, China, October 2004
8. AliCE Grid Computing Project, http://www.comp.nus.edu.sg/~teoym/atsuma.htm
9. D. Kurzyniec, T. Wrzosek, D. Drzewiecki and Vaidy Sunderam. Towards Self-organizing Distributed Computing Frameworks: the H2O Approach. Parallel Processing Letters, 13(2):pp 273-290, 2003
10. H2O Project, www.maths.emory.edu/dcl/h2o/
11. L. B. Costa, L. Feitosa, E. Araujo, G. Mendes, R. Coelho, W. Cirne and D. Fireman. MyGrid: a Complete Solution for Running Bag-of-tasks Applications. In Proceedings of the SBRC 2004
12. OurGrid Project, www.ourgrid.org
13. Jini, www.jini.org
14. JavaSpaces Specification, www.jini.org/nonav/standards/davis/doc/specs/html/js-title.html

15. GigaSpaces, www.gigaspaces.com

16. D. Kurzyniec, T. Wrzosek, Vaidy Sunderam and A. Slominski. RMIX: A Multiprotocol RMI Framework for Java. In Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'03), pp.140-146, Nice, France, April 2003

17. Autopilot, June 2003, http://vibes.cs.uiuc.edu/Software/Autopilot/autopilot.htm

18. GridICE: The eyes of the Grid, February 2004, http://server11.infn.it/gridice

19. GridRM, February 2004, http://gridrm.org

20. Grid Portals Infromation Repository, February 2004, http://www.tacc.utexas.edu/projects/gpir

21. Java Agents For Monitoring and Management (JAMM), July 2000, http://www-didc.lbl.gov/JAMM

22. M. Blyantov, L. Kirchev, V. Georgiev and K. Boyanov. A Hierarchical Architecture Supporting Services in Grid. In Proceedings of the Automatics and Informatics'05, Sofia, October 3-5, 2005, pp. 186 - 192

23. M. Blyantov, V. Georgiev and K. Boyanov. A Model of Simplified Resource Management for Lightweight Multilevel Grid. Information Technologies and Control, 2005 (?)

24. L. Kirchev, M. Blyantov, V. Georgiev, K. Boyanov, I. Taylor, A. Harrison, S. Isaiadis, V. Getov and N. C. Linde. Mapping “Heavy” Scientific Applications on a Lightweight Grid Infrastructure. CoreGRID Integration Workshop CGIW'05, Piza, Italy, 28-30 November, 2005

25. L. Kirchev, M. Blyantov, V. Georgiev and K. Boyanov. Communication Model Supporting Process Migration in Grid. EXPGRID, Paris, 21 June 2006

26. L. Kirchev, M. Blyantov, V. Georgiev, K. Boyanov, M. Malawski, M. Bubak, S. Isaiadis and V. Getov. User Profiling for Lightweight Grids. In Proceedings of GRIDS@Work, Sophia Antipolis, France, October 10 – 14, 2005

27. C. Larman, Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process, 2nd edition, Prentice Hall

28. Eclipse, http://www.eclipse.org

29. Jboss, http://www.jboss.org

30. Xdoclet, http://xdoclet.sourceforge.net

31. Jcreator, www.jcreator.com/download.htm