МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«САМАРСКИЙ ГОСУДАРСТВЕННЫЙ АЭРОКОСМИЧЕСКИЙ
УНИВЕРСИТЕТ имени академика С.П. КОРОЛЕВА
(национальный исследовательский университет)»

СОЗДАНИЕ ПРОГРАММНЫХ КОМПЛЕКСОВ РАСПРЕДЕЛЕННЫХ ВЫЧИСЛЕНИЙ НА ОСНОВЕ ТЕХНОЛОГИИ XDP В РАМКАХ ВЫЧИСЛИТЕЛЬНОЙ GRID-СИСТЕМЫ СГАУ

Методическое пособие по выполнению курсового проекта «Создание программных комплексов распределенных вычислений на основе технологии XDP», дисциплина «Технологии сетевого программирования»

САМАРА
2010

УДК 004.75

А.В. ГАВРИЛОВ

АННОТАЦИЯ

Методические указания к курсовому проектированию направлены на ознакомление студентов с основами архитектуры распределённых вычислительных систем. В качестве платформы для реализации распределённых программ рассматривается технология XDP, позволяющая прозрачным для программиста образом использовать разнородные вычислительные ресурсы.

В указаниях рассматриваются архитектурные принципы XDP, выделяются важные элементы сервис-ориентированных распределённых систем. Подробно излагаются порядок установки и настройки необходимого для работы программного обеспечения, порядок разработки модулей для работы в рамках XDP, приводятся примеры и объясняются базовые приёмы работы. Также рассматриваются основные этапы выполнения курсового проекта с применением технологии XDP.

Разработка предназначена для студентов специальностей и направлений «Прикладная математика и информатика», «Прикладные математика и физика». Рекомендуется использование настоящих методических указаний при выполнении курсовых проектов по дисциплине «Технологии сетевого программирования».

УДК 004.75

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Развитие вычислительной техники и телекоммуникационных технологий, с одной стороны, открывает новые возможности для решения задач высокой сложности, которые раньше не могли быть решены за разумное время. С другой стороны, при этом возникают новые задачи по созданию программных комплексов, позволяющих использовать такие вычислительные мощности.

Особое место в этом процессе занимают вычислительные GRID-технологии, позволяющие объединить в одну вычислительную среду как обычные компьютеры, создавая кластерные системы, так и суперкомпьютеры. При этом вычислительная среда представляется потребителю «прозрачной»: ему не нужно знать, где именно и как выполняется его задача, какие именно вычислители участвуют в её решении и с помощью каких средств и технологий вычислители взаимодействуют друг с другом.

В настоящих методических указаниях рассматривается технология XDP, разрабатывающаяся в Самарском государственном аэрокосмическом университете. В её основе лежат язык программирования Java и фреймворк OSGi, делающие технологию XDP независимой от вычислительной платформы, гибкой и легко настраиваемой. Универсальность платформы делает её выгодным инструментом для решения самых разнообразных задач, а простота её использования позволяет легко развернуть систему как в локальной сети, так и на вычислительных кластерах и суперкомпьютерах университета.

В пособии рассматриваются основы архитектуры распределённых вычислительных систем и особенности архитектуры XDP, подробно излагаются порядок установки и настройки необходимого для работы программного обеспечения, приводятся примеры создания элементов распределённой программы и рекомендации о порядке выполнения курсового проекта с применением технологии XDP.

1 АРХИТЕКТУРА ТЕХНОЛОГИИ XDP

Распределенная вычислительная система (РВС) [1] представляет собой систему, обеспечивающую инфраструктуру для запуска и выполнения задач на множестве входящих в нее узлов, работающих, например, на персональных компьютерах, объединенных сетью. В настоящее время такие системы составляют серьезную конкуренцию кластерам в силу активного распространения и роста компьютерных сетей. Для РВС адаптируют алгоритмы решения различных задач: расчета напряжений в сложных механизмах [2], расчета микро- и нано-оптических элементов, обработки большеформатных изображений, различных задач математической физики. В большинстве своем это задачи, требующие сверхбольших вычислительных и временных затрат, например трехмерный рендеринг сложных деталей.

На практике обычно используется реализация РВС, специализированная для решения конкретного класса задач. Перед разработчиками такой РВС стоит задача выбора программных средств (языков программирования, образцов проектирования, фреймворков (например, Globus Toolkit)) для реализации конкретной РВС. Другим подходом является использование реализованных универсальных РВС (Condor, X-Com) и специализация этих РВС под конкретный вид задач.

Таким образом, выбор разработчика определяет архитектуру РВС, а значит и возможности РВС, гибкость и удобство ее использования, настройки и сопровождения.

Ниже будет описано видение сервис-ориентированной архитектуры РВС, которое было положено в основу разработанной системы. Результаты реализации показывают не только возможность существования такого подхода, но и некоторые его преимущества.

1.1 Существующие подходы к реализации РВС

Существующие технологии позволяют разработчику самостоятельно реализовать РВС, например, с использованием функциональных языков Erlang или MPI. При этом разработчик должен спроектировать архитектуру и реализовать всю требуемую функциональности РВС. Как правило, такого рода системы, спроектированные под конкретную задачу, остаются крайне специализированными и трудны для применения в других условиях эксплуатации, например с новым протоколом коммуникации, или специализированной стратегией планирования.

Другой подход заключается в использовании реализованной РВС, например, Condor или X-COM. Обычно такая РВС предоставляет набор базовых интерфейсов программирования приложений (*Application Programming Interface, API*), позволяющий разработчику использовать инструменты РВС, соответственно в данном случае возможности разработчика ограничены предоставляемым API.

Бóльшие возможности для реализации РВС предоставляет набор инструментов Globus Toolkit. Он, по сути, является стандартом, который предоставляет набор инструментальных средств, а не замкнутый комплект модулей. Однако стоит отметить большую сложность при организации и администрировании РВС, реализованной на основе Globus.

Таким образом, при использовании существующих подходов возникают сложности адаптации РВС под конкретные условия эксплуатации.

1.2 Жизненный цикл работы РВС

Для эффективной работы РВС обычно используется много вычислительных узлов, на каждом из которых должна быть установлена соответствующая программа, выполняющая роль узла РВС.

Рисунок 1.1. Порядок запуска задачи на РВС

Как видно из рис. 1.1, на одном вычислительном устройстве может быть запущенно несколько узлов. Обычно выделяют один центральный узел РВС (Master), который будет распределять задачи между другими узлами (Slave) [3]. Этот же узел может использоваться для получения задачи от пользователя, сборки результата вычислений и возвращения ответа пользователю. Так как количество задач, поступающих на этот узел, заранее неизвестно, то возникает задача динамического планирования вычислительных ресурсов, решение которой возлагается на специальную службу – *планировщик*. В соответствии со стратегией планирования планировщик принимает решение о том, когда и на каких узлах запустить вычисление распределенной программы.

В процессе работы РВС могут происходить исключительные ситуации, чаще всего связанные с отказом узла РВС, влияющие на выполнение вычислительных задач и функционирование РВС в целом. Это существенно усложняет задачу коммуникации между узлами и модулями распределенных программ, запущенных на них, так как при отказе узла возможны потери данных, необходимых для дальнейших вычислений, потеря связи с модулем распределенного программы и т.д. Таким образом, РВС должна решать задачи создания

контрольных точек при работе распределенной программы, поддерживать репликацию данных, миграцию модулей распределенных программ. Эти и другие функции предопределяют роль РВС в качестве инфраструктуры для распределенных программ.

1.3 Жизненный цикл распределённой программы

Как было показано на рис.1.1, модули распределенной программы работают изолированно с точки зрения разных физических узлов и разных процессов операционной системы. На одном узле могут быть одновременно запущены несколько разных модулей вычислительных программ, которые будут работать в некоторой среде, предоставляемой узлом РВС. Такую среду назовем *Контейнером*.

Контейнер обеспечивает жизненный цикл модулей распределенных программ, предоставляет им API распределенной вычислительной системы для получения начальных данных, коммуникаций между модулями распределенной программы, сохранения результата и т.д. В связи с возможность отказа узлов, единая архитектура контейнера на разных узлах позволяет легко реализовать миграцию задачи в другой контейнер с возможностью старта задачи с последней контрольной точки, что позволяет минимизировать потери времени при отказе узла.

1.4 Особенности параллельных алгоритмов с точки зрения архитектуры системы

В литературе подробно описаны параллельные алгоритмы, предназначенные для выполнения на системах параллельных вычислений [4,5]. Параллельные алгоритмы достаточно часто являются основой при разработке распределенных программ.

Одним из недостатков разработки распределенных программ на основе параллельных алгоритмов является то, что кроме непосредственного решения задачи, параллельные алгоритмы содержат алгоритмы передачи данных по сети. Как видно из рис. 1.2, для работы

распределенной программы необходимо решить две задачи: *вычислительную* и *коммуникационную*.

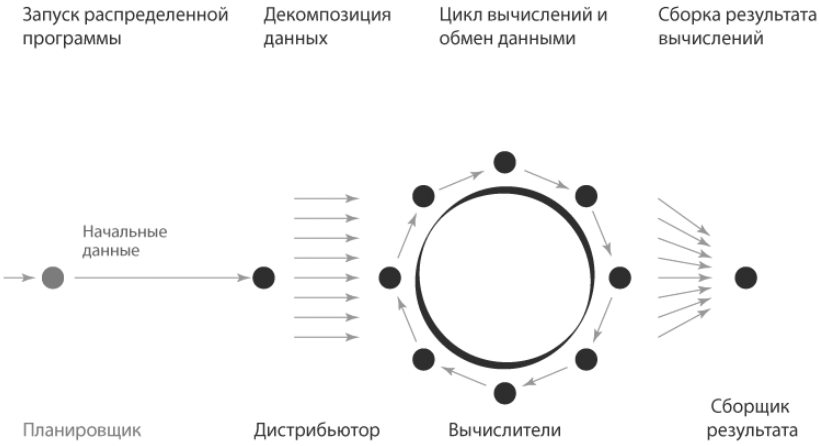

Рисунок 1.2. Этапы работы распределённой программы

Вычислительная задача представляет собой вычислительный код алгоритма, на вход которого подаются начальные данные, а на выход поступает результат вычислений.

Коммуникационная задача, включает в себя запуск модулей распределенной программы на разных узлах, организацию коммуникации между ними, передачу начальных данных, сборку результата, организацию конкретной топологической структуры [4] вычислительных модулей, запущенных на различных узлах.

Такая архитектура предопределена при работе параллельного алгоритма на системах параллельных вычислений, так как параллельная программа, формируя поток инструкций для процессора, определяет и порядок вычислений, и порядок передачи данных между процессорами.

Однако в рамках работы распределенной программы на РВС задачи коммуникации представляют собой очевидную проблему в си-

лу того, что разные узлы РВС могут выходить из строя, модули могут мигрировать, данные реплицироваться. Это означает, что модули распределенной программы не знают, на каком физическом узле находятся необходимые им данные, где работают другие модули распределенной программы и как с ними обмениваться данными. Вообще говоря, модуль даже не может гарантировать, что он закончит вычисления на том же физическом узле, где он их начал. Информацией о местонахождении данных и модулей распределенных программ владеет только РВС.

Решение обозначенных выше задач имеет смысл переложить с разработчиков распределенных программ на средства РВС. Для этого предлагается подход, предусматривающий наличие в РВС единой информационного среды (ЕИС) – системы хранения и коммуникации данных, которая позволяет отделить решение вычислительной задачи от решения коммуникационной. Все операции, связанные с хранением и передачей данных, а также коммуникацией между узлами (например, на основе распределенных событий) берет на себя ЕИС.

1.5 Роли модулей в жизненном цикле распределенной программы

Как было сказано выше, основная задача распределенной программы – производить вычисления в соответствии с вычислительным алгоритмом. Однако на практике кроме реализации вычислений приходится выполнить ряд вспомогательных задач: декомпозиция начальных данных и их подготовка для вычислений, сборка результата и т.д.

Эти операции можно выполнять асинхронно по отношению к операциям вычисления. Так, например, сборщик результата часто может начинать работу, не дожидаясь результатов отстающего вычислительного модуля, а с получением данных первого.

Как видно из рис. 1.2, структурно в распределенной программе можно выделить несколько взаимодействующих модулей, каждый из которых решает собственную задачу.

Основной модуль – модуль вычислений (Solver), который будет запущен на предоставленных РВС узлах.

Вспомогательные модули, решающие задачи:

- декомпозиции начальных данных (Distributor),
- сборки конечного результата (Combiner),
- определяющий жизненный цикл распределенного алгоритма (Director).

В состав распределенного алгоритма могут входить несколько различных вычислительных модулей, которые запускаются на РВС поэтапно. Например, преобразование изображения из цветового пространства RGB в HSL, применение к изображению в пространстве HSL КИХ-фильтра, преобразование результирующего изображения в цветовое пространство RGB. В этом случае Director описывает весь жизненный цикл работы такого распределенного алгоритма.

Структурно модули, выполняющие разную роль, абсолютно идентичны как для контейнера, так и ЕИС. Такой подход упрощает разработку распределенных программ. Во-первых, за счет декомпозиции сложного параллельного алгоритма на несколько более простых частей, каждая из которых выполняет свою роль. А во-вторых, данный подход позволяет разработчику сосредоточиться на решении вычислительной задачи, перекладывая решение коммуникативной задачи на РВС.

1.6 Архитектура РВС на основе сервис-ориентированного подхода

Как было сказано выше, РВС является инфраструктурой для распределенных программ, предоставляя им богатые возможности и

отвечая за стабильность работы РВС. Исходя из этого и приведенных ранее соображений, сформулируем требования к РВС.

Распределенная вычислительная система должна:

- предоставлять систему для хранения и передачи данных;
- поддерживать необходимые для полноценной работы (в рамках поставленных перед РВС задач) протоколы коммуникации (например, пиринговые);
- поддерживать репликацию данных;
- предоставлять среду для запуска и выполнения вычислительных задач;
- управлять жизненным циклом РВС и запущенных на них вычислительных задач;
- предоставлять механизм сохранения контрольных точек задачи;
- в случае отказа, самостоятельно мигрировать задачу на стабильный узел и запускать ее с последней сохраненной контрольной точки;
- предоставлять инструмент простого распределенного самообновления на всех узлах, задействованных в РВС;
- предоставлять адаптивные алгоритмы (например, планирования) учитывающие изменяющиеся условия эксплуатации РВС;
- измерять необходимые метрики для работы алгоритмов.

Как видно, вышеизложенные требования достаточно независимы и могут решаться независимыми модулями — сервисами, согласованно взаимодействующими на каждом узле и во всей РВС [6]. Использование сервис-ориентированного подхода к реализации РВС позволяет расширять возможности конкретного узла РВС за счет использования согласованно работающих независимых сервисов, каждый из которых выполняет свою задачу.

Архитектурный слой, который занимается обеспечением жизненного цикла сервисов, принято называть *платформой*. Важную роль играет выбор фреймворка для реализации платформы, т.к. он

во многом предопределяет возможности РВС. Отдельно отметим, что одним из ключевых моментов является возможность замены и запуска сервисов «на лету» (без перегрузки платформы), что позволяет устанавливать и обновлять необходимые сервисы на всех узлах РВС, обеспечивая однородную по предоставляемой функциональности среду на разных узлах РВС.

С учетом изложенного выше предлагается концепция сервис-ориентированной архитектуры РВС, которая позволяет произвести декомпозицию различных частей РВС на самостоятельные архитектурные слои (наборы сервисов), что облегчает реализацию конкретной РВС и специализацию ее под конкретные задачи (путем специализации только конкретных сервисов).

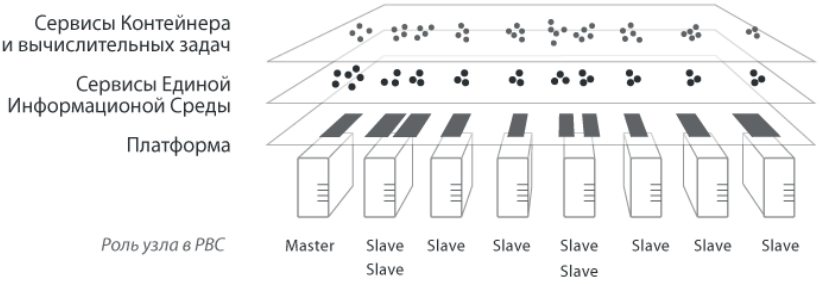

Рисунок 1.3. Архитектура сервис-ориентированной РВС

На рис. 1.3 представлены основные архитектурные слои, каждый из которых использует предоставляемые возможности предыдущего слоя. Рассмотрим каждый из них подробнее.

Контейнер обеспечивает:

- запуск модулей вычислительных программ,
- жизненный цикл модулей вычислительных программ,
- предоставление доступа к API РВС.

Единая информационная среда реализует:

- хранение вычислительных данных, метрик, событий, информации о модулях вычислительных программ,
- создание реплик данных, обеспечивающее сохранность данных в случае отказа узлов РВС,
- предоставление сервисам единого информационного пространства имен сервисов, данных, событий и метрик,
- предоставление сервисам доступа к чтению, записи и удалению данных,
- коммуникацию между модулями (например, на основе модели событий).

Платформа, в свою очередь, реализует:

- обеспечение жизненного цикла служебных сервисов,
- обеспечение жизненного цикла вычислительных сервисов,
- запуск, остановку, и перезапуск сервисов,
- обновление, существующих сервисов,
- установку новых сервисов.

Существующие подходы к созданию РВС в том или ином (обычно неявном) виде содержат данные архитектурные слои, однако предлагаемая архитектура в явном виде перераспределяет задачи, решаемые РВС, на слабо связанные и относительно независимые сервисы. Такой подход делает прозрачным архитектуру РВС и упрощает ее разработку, адаптацию, развитие.

1.7 Реализация сервис-ориентированной архитектуры XDP

Далее приведено описание реализации РВС по рассмотренному выше принципу построения архитектуры.

Фреймворк XDP (eXtensible Distributed Platform) был разработан для решения вычислительных задач в рамках одноранговой локальной сети. Данные ограничения определены выбором протоколов коммуникации (сервис Connector) и не являются ограничениями архитектуры РВС. Языком программирования является язык Java вер-

сии 1.5, определяющий возможность кроссплатформенной работы РВС.

Структурно следует выделить 3 архитектурных слоя:

- набор сервисов Slavery, реализующих контейнер и сервисы распределенных вычислительных алгоритмов;
- набор сервисов XDP, реализующих ЕИС;
- сервис-ориентированная платформа на основе спецификации OSGi.

Каждый уровень работает независимо от вышестоящих и лишь определяет возможности распределенной системы. Таким образом, фреймворк XDP не специализирован только для выполнения распределенных вычислений и может быть расширен и дополнен другими сервисам и архитектурными слоями.

1.8 Сервис-ориентированная платформа OSGi

Платформа XDP была разработана с применением технологии OSGi на базе реализации этой технологии – фреймворка Equinox от Eclipse Foundation – с использованием сервисов, предоставляемой другой реализацией технологии – Knoplerfish. Фреймворк Equinox представляет собой динамическую систему, обеспечивающую модульность для языка программирования Java.

Технология OSGi позволяет создавать программы из небольших модулей (*bundles*), пригодных для повторного использования. OSGi использует сервис-ориентированную модель взаимодействия компонентов, а также позволяет подключать, отключать и изменять компоненты *без перезагрузки* системы [7]. В настоящее время существует множество компонентов, разработанных для платформы OSGi, выполняющих самые различные функции: обеспечение безопасности, поддержку ведения журналов, конфигурирование приложений.

На рис. 1.4 приведена общая архитектура OSGi и взаимодействие с платформой Java.

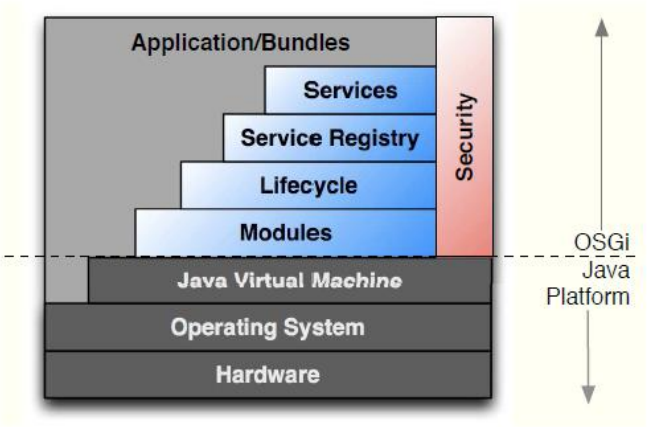

Рисунок 1.4. Структура платформы OSGi

Фреймворк OSGi состоит из следующих программных слоев.

1. *Среда исполнения* (Execution Environment) – средой исполнения программных модулей OSGi является один из стандартных вариантов среды исполнения Java (J2SE, MIDP, CDC, CLDC и т.п.).

2. *Система поддержки модульности* (Modules) – эта система определяет политику загрузки Java-классов. OSGi предоставляет гибкую и мощную систему загрузки классов, базирующуюся на аналогичной системе Java, но существенно расширяющей ее возможности. Эта система вводит понятие программного модуля, для которого можно определить набор принадлежащих ему классов, и правила взаимодействия с классами, принадлежащими другим программным модулям. Система поддержки модульности тесно интегрирована с системой безопасности.

3. *Система управления жизненным циклом модулей* (Life Cycle) – эта система вводит понятие бандлов (bundles), которые могут быть динамически установлены, запущены, обновлены, остановлены, ли-

бо удалены. Механизм бандлов использует для загрузки классов механизм модулей, но добавляет к нему функциональность управления модулями без необходимости остановки системы. Операции поддержки жизненного цикла полностью защищены системой безопасности, что делает эту подсистему практически неуязвимой для атак.

4. *Система реестра сервисов* (Service Registry) – реестр сервисов предоставляет средства коммуникации между бандлами, принимая во внимание динамику их жизненного цикла. Существует механизм событий, позволяющий отслеживать появление или исчезновение сервисов. Каждый сервис является обычным Java-объектом и может выполнять любые функции. Сервис характеризуется своим Java-интерфейсом. Бандлы могут реализовать этот интерфейс и зарегистрировать реализацию в реестре сервисов. Клиенты этого сервиса могут получить его из реестра, либо выполнять различные действия при его появлении и исчезновении. Система безопасности предоставляет полный набор функциональности для защиты механизма сервисов.

5. *Система безопасности* (Security) – тесно интегрирована со всеми слоями фреймворка. Она основана на обычных моделях безопасности Java и Java2. Эти системы безопасности сами по себе защищают систему от многих видов атак. Система безопасности OSGi расширяет стандартную функциональность механизмом приватных для модуля классов и полным динамическим управлением правами доступа.

Таким образом, фрэймворк Equinox предоставляет законченную динамичную компонентную модель, которая позволяет создать РВС. Каждый модуль такой системы может быть удаленно установлен, запущен, остановлен и удален без перезапуска платформы.

Как видно, реализация РВС в виде набора сервисов дает нам следующие преимущества:

- возможность динамического добавления новых сервисов;

- интеграция с существующими сервисами платформы;
- возможность простого обновления сервисов на вычислительных узлах.

1.9 Единая информационная среда

Как было описано ранее, единая информационная среда — архитектурный слой, задачей которого является предоставление сервисам информации о других сервисах, событиях произошедших в системе, метриках и данных сохраненных в системе.

В XDP ЕИС реализован набором взаимодействующих сервисов *Хранилища данных* (Storage) и *Коммуникаций* (Connector), а также вспомогательных сервисов Метрик (Metrics), Событий (Events), репликаций (Replication), работы с базой данных (SQL-совместимой) и некоторых других.

Сервис *Хранилища* предоставляет интерфейс другим сервисам для взаимодействия с данными, и осуществляет работу с данными с помощью интерфейса предоставляемого сервисом *Коммуникаций*.

Минимально необходимый интерфейс, предоставляемый ЕИС для взаимодействия с данными, определяется следующими функциями:

- чтение (load) — загрузка данных из *Хранилища;*
- запись (save) — запись данных в *Хранилище;*
- удаление (delete) — удаление данных из *Хранилища*.

Операции записи данных и удаления должны выполниться для всех реплик данных, расположенных на разных узлах РВС.

Сервис *Коммуникаций*, в свою очередь, определяет протоколы взаимодействия. Реализуются обработчики следующего набора протоколов: UDP, TCP, HTTP(S), LRMP, TRAM, RPC. Все обработчики делятся на два вида: обработчики протоколов потоковой передачи данных и обработчики протоколов пакетной передачи данных. Обработчики потоковых протоколов предоставляют интерфейс для по-

лучения потоков ввода\вывода, с помощью которых и пересылаются данные, в то время как обработчики пакетных протоколов предоставляют интерфейс для отправки пакетов данных. Гибкость инфраструктуры подключений позволяет реализовать обработчики для протоколов, не имеющих отношения к наиболее часто используемым протоколам TCP/IP и UDP.

В рамках платформы XDP минимально возможный интерфейс, предоставляемый ЕИС, расширен с помощью следующей функцией:

- ассинхронная загрузка (asynh_load) — метод позволяет предварительно запросить данные из *Хранилища*, не блокируя работу вычислительного алгоритма. Вычислительный алгоритм, запрашивает данные и продолжает работу, до того момента, как эти данные действительно потребуются, к этому моменту данные либо уже доставлены, либо алгоритм, засыпает (wait) до момента прихода данных.

Так же в рамках XDP разработаны другие сервисы, расширяющие возможности ЕИС и предоставляющие разработчику вычислительных алгоритмов удобные инструменты разработки распределенных вычислительных программ. Ниже приведены основные программные модули, содержащие сервисы XDP, имеющие непосредственное отношение к реализации жизненного цикла вычислительных сервисов.

- Сервис сбора метрик (`org.xdp.metrics`) – сервис, отвечающий за сбор и хранение метрик, генерируемых системой и распределенным программами.
- Сервис расчета метрик (`org.slavery.metrics`) – сервис, получающий события от сервиса сбора метрик и рассчитывающий метрики на основе хранящихся в сервисе сбора метрик.
- Сервис событий (`org.xdp.eventmanager`) – сервис позволяет создавать и распространять распределенные и локальные события по РВС.

- Сервис миграции (`org.slavery.sentinel`) – сервис, отслеживающий работу платформы и запущенных на ней задач, при отказе узла, мигрирует вычислительные задачи на другие узлы и запускает их с текущей контрольной точки.

Система XDP предоставляет множество сервисов, реализующих функциональность вычислительного узла РВС. При написании вычислительных сервисов программист может пользоваться как стандартными сервисами OSGi, так и сервисами XDP, однако, следует отдавать предпочтение последним, так как их использование приводит к увеличению надежности и качества написания вычислительных сервисов. Многие из сервисов XDP носят чисто служебные функции, не предоставляя разработчику вычислительных сервисов возможности их использовать напрямую, тем не менее, предоставляемый ими функционал используется другими сервисами XDP.

- Сервис запуска вычислительных задач (`org.xdp.starter`) – представляет базовые классы для создания вычислительной задачи в понятном системе виде и механизмы запуска задач как изнутри системы так из внешних клиентов. В том числе поддерживает эффективный механизм сохранения контрольных точек (состояния вычислительного алгоритма) для миграции вычислительного алгоритма между узлами.

Также в платформе реализованы сервисы: распространения исполняемого кода (автоматического обновления сервисов), поддержки БД SQL Lite и ряд других.

Отдельно стоит остановиться на сервисах коммуникации и хранения и передачи данных.

Сервис хранения и передачи данных представляет собой механизм поддержки распределенного хранилища данных системы. Данное хранилище может быть использовано для унификации интерфейса обмена данными. Обмен данными с использованием хранилища происходит прозрачно и не требует знания местоположения

адресата, которому предназначены данные. Отправитель сохраняет данные с некоторым строковым идентификатором. Получатель, зная этот идентификатор, может синхронно либо асинхронно загрузить данные на свой узел. После использования данные могут быть удалены. На каждом узле РВС размещается два сервиса – сервис хранилища данных и сервис реестра данных. Сервис хранилища данных активен на каждом узле РВС, в то время как сервис реестра данных активен одновременно только на одном узле РВС (но его копии могут храниться на других узлах). Оба этих сервиса состоят из двух частей – локальной реализации сервиса и его сетевой части. Локальная часть каждого сервиса, по сути, представляет собой интерфейс к одной из двух (для каждого сервиса своей) таблиц служебной БД xdp_storage. В таблице storage для каждого идентификатора данных хранятся данные, сжатые методом GZIP и закодированные в Base64, в таблице storage_registry для каждого идентификатора данных хранится набор URL хранилищ тех узлов, которые содержат копии данных, сохраненных с этим идентификатором.

По умолчанию XDP предоставляет две системные базы данных:

- • xdp_storage – БД, используемая распределенной системой хранения данных. Содержит таблицы, обеспечивающие работу хранилища и реестра данных системы.
- • xdp_other – БД, используемая различными сервисами системы, например, сервисом сбора метрик.

Хранение системных данных внутри БД обеспечивает множество преимуществ: повышенная скорость работы, улучшенная надежность хранения, возможность обеспечения безопасности, гибкие возможности запроса системной информации. В частности, БД незаменима для хранения информации о метриках системы – в связи с большим объемом этой информации. БД предоставляет широкие возможности выборки и анализа информации о системе.

1.10 Контейнер. Уровень Slavery

Как уже отмечалось, основной задачей контейнера является обеспечение жизненного цикла вычислительных распределенных программ. Несмотря на то, что вычислительная распределенная программа является сервисом, для полноценной работы в рамках РВС необходим механизм запуска и обеспечения жизненного цикла сервиса распределенной программы, расширяющий стандартные механизмы OSGi.

Следует отметить, что многие сервисы OSGi не являются активными сервисами, выполняющими функции в рамках жизненного цикла, но необходимы для имплементации их кода вычислительными распределенными программами, например, математические алгоритмы, которые удобно выделить в отдельный сервис, позволяющий независимо развивать математические возможности РВС.

С точки зрения архитектуры уровень состоит из:

- контейнера;
- планировщика вычислительных задач, стартующих модули на узлах РВС;
- математических сервисов, предоставляющих доступ к реализованному математическому аппарату на основе apache.common;
- сервиса базовых классов модулей распределенных программ, описывающих роли и базовые возможности распределенных программ;
- вычислительных распределенных программ, реализованных в виде сервисов, наследующих базовые классы модулей распределенных программ.

Основными сервисами *Контейнера* являются:

- *Исполнитель* вычислительных распределенных программ (Executor) – представляет базовые классы для разработки распределенной программы, механизмы запуска задач как изнутри системы, так и из внешних клиентов, поддерживает эффективный ме-

ханизм сохранения контрольных точек для миграции вычислительного алгоритма между узлами;

- *Часовой* (Sentinel) – отслеживает жизненный цикл распределенных программ и обеспечивающих миграцию сервисов.

Таким образом, набор сервисов Контейнера реализует необходимые для работы распределенных программ инструменты и управляет жизненным циклом распределенных программ.

Исходя из общих предположений, не уменьшающих универсальность распределенной вычислительной системы, предложена архитектура РВС. Выделение нескольких архитектурных слоев разделяет сервисы по сферам их ответственности и облегчает их замену. Наличие единой информационной среды облегчает работу с данными и обеспечение их сохранности.

С использованием разработанной архитектуры была реализована экспериментальная РВС. Выбор фрэймворка OSGi эффективно обеспечивает модульность системы.

Опыт создания распределенных программ для данной системы говорит о том, что она соответствует поставленным требованиям: универсальности, гибкости и простоты.

Универсальность платформы XDP дает возможность использовать ее не только в исследовательских, но и в производственных целях.

2 УСТАНОВКА И НАСТРОЙКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Поскольку технология XDP основана на фреймворке OSGi, наиболее подходящей средой разработки (*IDE*) для программ в рамках этой технологии будет Eclipse (см. рис. 2.1). Также возможна разработка и в других средах (NetBeans, IDEA и т.д.), однако рассмотрение данного вопроса находится за рамками настоящих методических указаний.

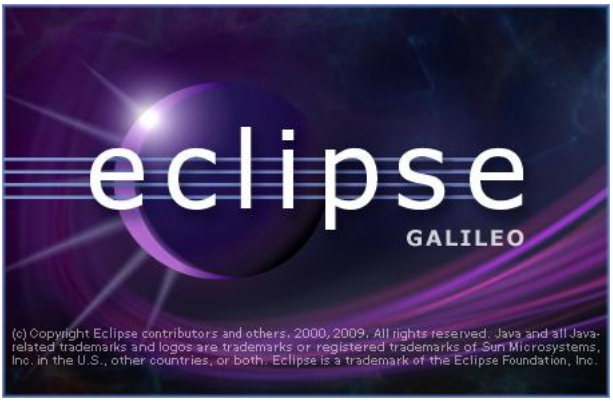

Рисунок 2.1. Окно запуска среды разработки Eclipse

Перед началом работы потребуется:

- установить среду разработки Eclipse;
- установить и настроить модуль платформы XDP
- настроить целевую платформу для проектов.

2.1 Установка среды Eclipse

Дистрибутив среды может быть загружен по адресу `http://www.eclipse.org/downloads/`. Лучше использовать

вариант среды Eclipse for RCP and RAP developers. Также вы можете получить текущий рекомендуемый дистрибутив у преподавателя.

После установки и запуска среды вам будет предложено выбрать т.н. «рабочее пространство» (*workspace*), см. рис. 2.2. Рабочее пространство определяется папкой, в которой Eclipse будет сохранять ваши проекты, а также хранить ряд настроек. Причём эти настройки будут общими для всех проектов в рамках рабочего пространства. Таким образом, рабочее пространство является метапроектом, определяющим параметры разработки и выполнения ваших проектов.

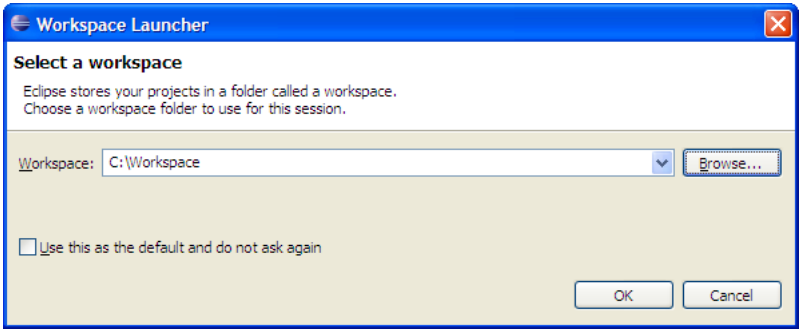

Рисунок 2.2. Окно выбора рабочего пространства

После выбора папки рабочего пространства следует нажать кнопку *OK*. Также вы можете упростить последующие запуски среды и выбрать данную папку в качестве рабочей по умолчанию, поставив галочку *Use this as the default and do not ask again*.

После запуска среды разработки вы увидите приветственное окно, выбрав в котором переход к рабочему пространству (*Go to the workbench*), вы перейдёте к основному рабочему окну (см. рис. 2.3). В верхней части располагается общее меню и панель быстрого доступа. В левой части – инспектор проектов и пакетов. В нижней – закладки со вспомогательными окнами (там, в том числе, будет и окно с информацией, выводимой приложениями в консоль). Центральную

часть будет занимать окно с закладками, соответствующими редактируемым файлам.

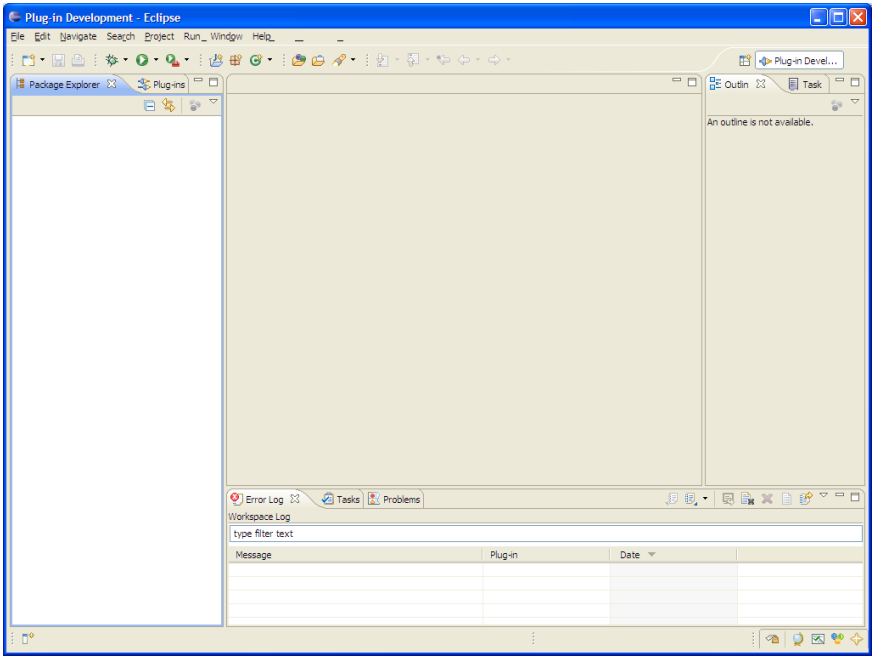

Рисунок 2.3. Общий вид среды разработки Eclipse

2.2 Установка и настройка модуля платформы XDP

Установка модуля платформы XDP производится простым разворачиванием архива в указанную папку. Текущую версию платформы вы можете получить у преподавателя или загрузить из указанного им файлового хранилища.

Платформа состоит из:

- папки `Plugins`, в которой находятся jar-архивы, обеспечивающие работу платформы OSGi, и jar-архивы модулей платформы XDP;
- файлов конфигурации платформы;

- файла запуска платформы в виде самостоятельного приложения.

Перед использованием платформы следует сконфигурировать её для вашей сети, отредактировав файл `hosts.conf`. В файле указываются компьютеры, на которых установлена платформа XDP. Если при формировании файла будут допущены ошибки (указаны несуществующие компьютеры, неверно указаны порты, на этих компьютерах не развёрнута платформа XDP и т.д.), запуск платформы произойдёт с ошибками и корректная её работа не будет возможна.

Каждая строчка файла имеет вид:

```
<host name>:<port number>
```

где `<host name>` – имя или IP-адрес компьютера в сети, а `<port number>` – номер порта на этом компьютере, на который будет производиться соединение. Если номер порта не будет указан, то будет использоваться значение порта по умолчанию (оно является настройкой платформы в целом). Если в файле будет несколько эквивалентных записей, то они будут трактоваться как одна запись. Примерный вид файла приведён в листинге 2.1.

Листинг 2.1. Возможный вид файла hosts.conf

```
Ivanov:15000
Petrov:15000
Fileserver:4000
192.168.0.10:2001
192.168.0.11:20002
```

Для работы с одним из кластеров или суперкомпьютеров GRID-сети СГАУ вам следует получить файл с готовыми настройками сети у преподавателя. Если вам необходимо промоделировать работу распределённого приложения на одном компьютере, вы можете сконфигурировать платформу так, как показано в листинге 2.2.

Листинг 2.2. Возможный вид файла hosts.conf для работы на одном компьютере

```
localhost:15000
```

```
localhost:15001
localhost:15002
```

2.3 Настройка целевой платформы для проектов

В среде разработки Eclipse одним из ключевых параметров в рамках рабочего пространства, определяемых для проектов встраиваемых модулей, является т.н. целевая платформа (*Target Platform*), т.к. она определяет настройки, параметры сборки и запуска проекта. Для её настройки следует в основном меню выбрать пункт *Window/Preferences*, после чего в дереве в левой части окна выбрать пункт *Plug-in Development/Target Platform* (см. рис. 2.4.)

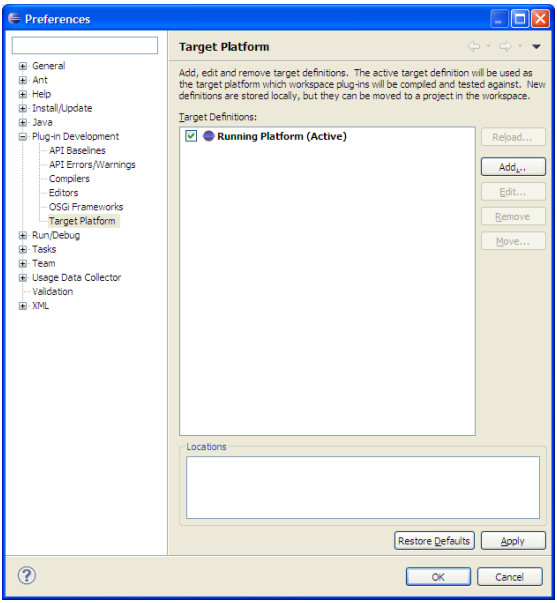

Рисунок 2.4. Окно выбора целевой платформы

Для добавления платформы XDP следует нажать на кнопку *Add*, после чего появится окно выбора способа создания новой конфигурации (см. рис. 2.5). В данном окне следует оставить выбранным

параметр *Nothing: Start with an empty target definition*, после чего нажать кнопку *Next* для перехода к выбору параметров платформы.

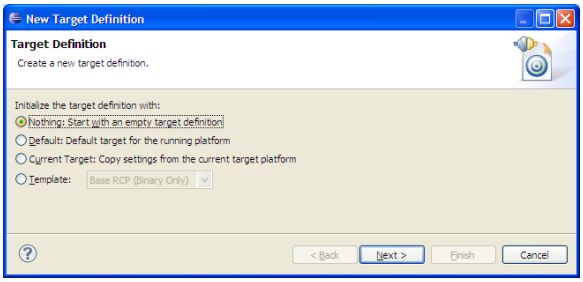

Рисунок 2.5. Окно выбора способа создания конфигурации целевой платформы

В первую очередь следует указать имя для конфигурируемой целевой платформы в поле редактирования *Name* (см. рисунок 2.6).

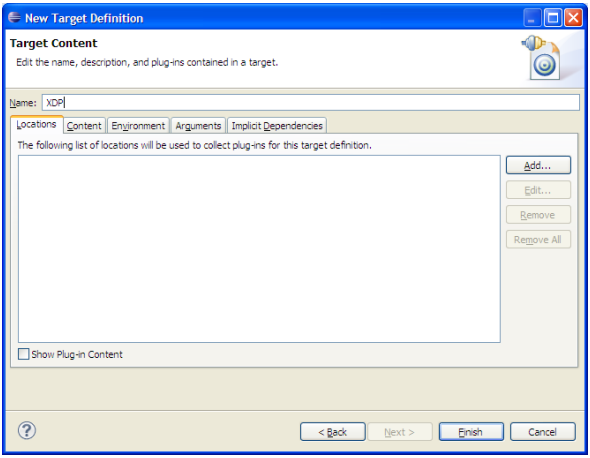

Рисунок 2.6. Окно параметров целевой платформы

Далее следует указать физическое расположение модуля используемой платформы. Для этого нужно при открытой закладке *Locations* нажать на кнопку *Add*. После этого появится окно выбора

способа доступа к платформе (см. рис. 2.7), в котором следует выбрать вариант *Directory*, после чего нажать кнопку *Next*.

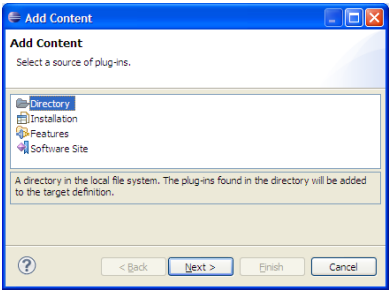

Рисунок 2.7. Выбор способа доступа к платформе

В следующем окне вам будет предложено выбрать путь к папке, где расположена реализация платформы, например, `C:\XDP` (см. рис. 2.8). После указания пути можно либо сразу нажать кнопку *Finish*, либо нажать кнопку *Next* и предварительно просмотреть список модулей платформы.

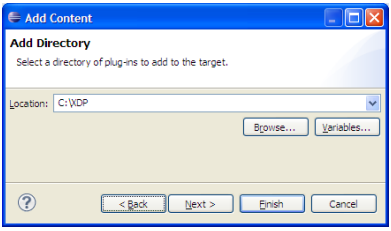

Рисунок 2.8. Выбор пути к реализации платформы

Также следует указать параметры окружения, в котором будет выполняться программа. Для этого в редакторе параметров целевой платформы следует перейди на закладку *Environment* и выбрать параметры из предлагаемых. На рис. 2.9 приведены типовые параметры для запуска в ОС Windows (32-х разрядной). В случае запуска в других ОС вам следует выбирать другие параметры. Так же вы мо-

жете указать и выполняющую среду Java (*JRE*), однако данный параметр имеет смысл, только если у вас установлено несколько JRE.

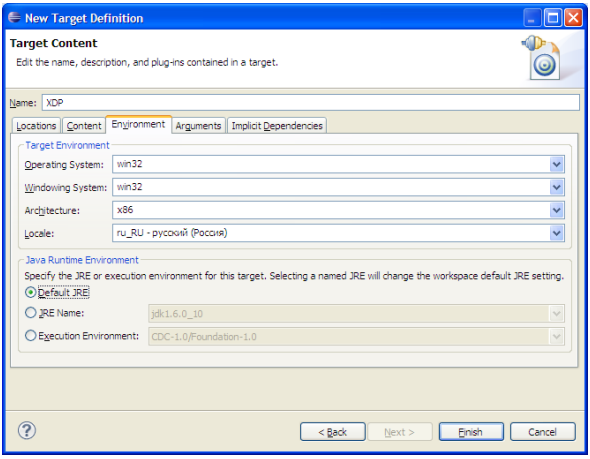

Рисунок 2.9. Настройка окружения целевой платформы

После завершения настроек следует нажать кнопку *Finish*, после чего вы перейдёте обратно к списку доступных в рабочем пространстве целевых платформ. Для того, чтобы вы могли использовать настроенную платформу, её следует сделать активной. Для этого нужно нажать рядом с её названием галочку (см. рис. 2.10).

Также будет полезным сменить кодировку редактора в рамках вашего рабочего пространства. Для этого в окне свойств (доступном при выборе в основном меню пункта *Window/Preferences*) следует выбрать в дереве в левой части пункт *General/Workspace*, где в разделе *Text file encoding* нужно выбрать вариант *Other:*, и среди предложенного выбрать кодировку UTF-8 (см. рис. 2.11).

В результате описанных выше действий вы получите установленную среду разработки Eclipse, в рамках рабочего пространства которой настроена целевая платформа XDP. После этого вы сможете

создавать и запускать свои проекты распределённых программ на основе технологии XDP.

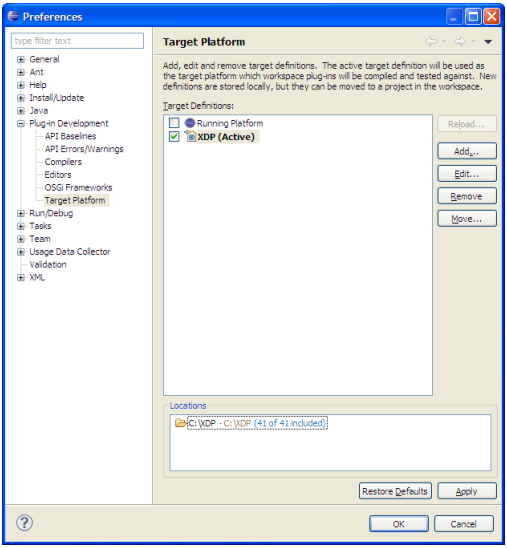

Рисунок 2.10. Выбор активной целевой платформы

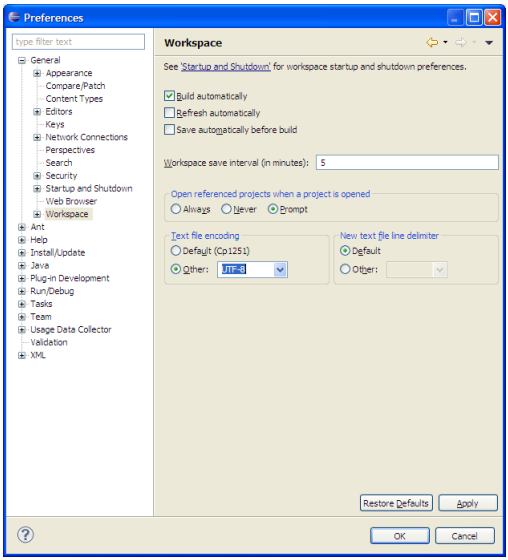

Рисунок 2.11. Выбор кодировки рабочего пространства

3 СОЗДАНИЕ ПРОЕКТА МОДУЛЯ ДЛЯ XDP

Поскольку модули XDP являются так же и модулями (*bundles*) фреймворка OSGi, их структура и порядок разработки продиктованы этой технологией. В результате сборки проекта должен получиться jar-архив, состоящий из:

- файла манифеста `manifest.mf`, в котором описываются параметры модуля;
- пакетов с классами, реализующими логику вашего модуля;
- файла `Activator.java`, в котором описывается класс `Activator`, используемый платформой OSGi как точка входа модуля.

В ходе разработки модуль будет представляться в Eclipse как проект для Plug-in, для которого потребуется указать некоторые дополнительные настройки.

3.1 Создание и настройка проекта модуля

Для создания нового проекта следует в основном меню Eclipse выбрать команду *File/New/Project*, после чего появится окно выбора вида проекта (см. рис. 3.1). В дереве вариантов проектов следует выбрать пункт *Plug-in development/Plug-in Project*, после чего нажать кнопку *Next*.

В появляющемся после этого окне настроек проекта (см. рис. 3.2) следует указать имя проекта (поле ввода *Project name:*). Также следует в разделе целевой платформы (*Target Platform*) выбрать вид модуля *an OSGi framework*, а в качестве реализации OSGi выбрать платформу *Equinox*. После этого следует нажать на кнопку *Next*, чтобы перейти к дальнейшим настройкам проекта (см. рис. 3.3). В этом окне вы можете отредактировать идентификатор проекта, его версию, а также указать параметры среды исполнения.

В данном случае вы можете оставить все эти параметры без изменений и нажать кнопку *Next*.

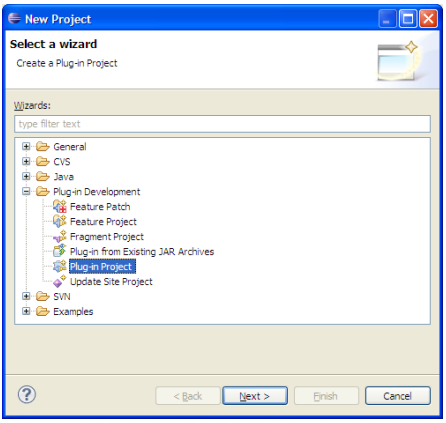

Рисунок 3.1. Выбор вида проекта

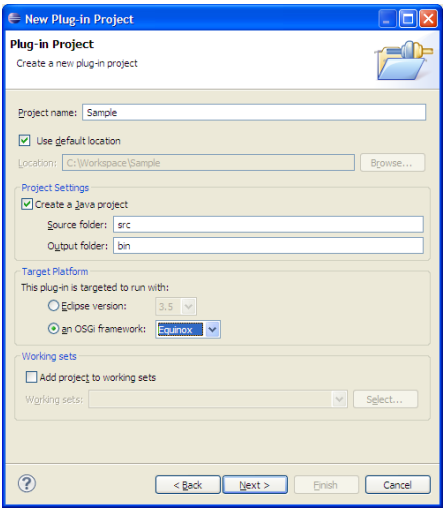

Рисунок 3.2. Настройки проекта модуля

Далее следует выбрать один из шаблонов создания проекта. Для простоты можно выбрать шаблон *Hello OSGi Bundle* (см. рис. 3.4).

После нажатия кнопки *Next* вам будет предложено выбрать сообщения о запуске и завершении работы модуля. После нажатия кнопки *Finish* создание проекта будет завершено.

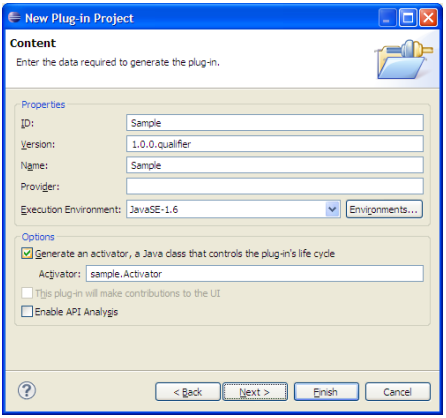

Рисунок 3.3. Настройки при создании проекта

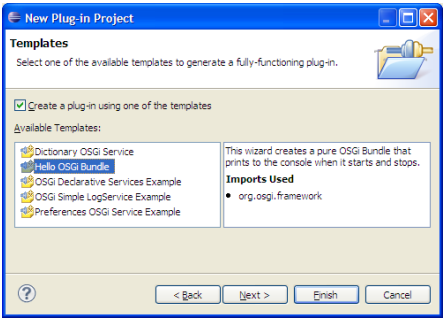

Рисунок 3.4. Выбор шаблона проекта при создании

Далее вам будет доступно редактирование файлов проекта в обычном для среды Eclipse стиле (см. рис. 3.5). В левой части в закладке Package Explorer будет выводиться структура проекта, включая файлы свойств, и списки библиотек, используемых в проекте. Вся эта информация представляется в виде дерева, в котором щел-

чок по узлам дерева будет разворачивать/сворачивать содержимое узла, а двойной щелчок по элементу будет открывать его для редактирования в основной части. Сразу после создания проекта для редактирования будет открыт файл манифеста `manifest.mf`.

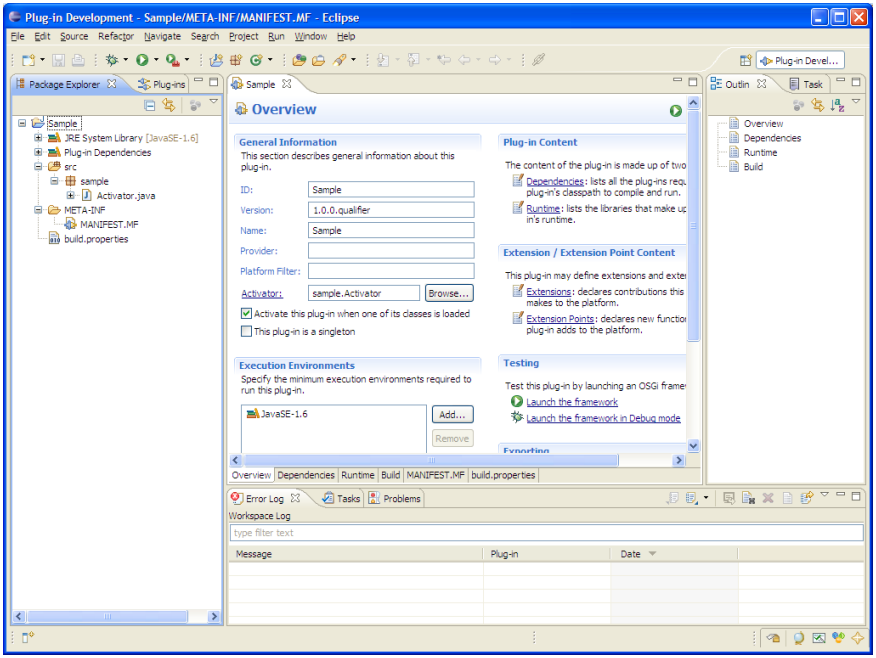

Рисунок 3.5. Окно проекта с открытым файлом manifest.mf

Для корректной работы вашего модуля платформы XDP следует установить зависимости от других модулей. Для этого нужно в разделе *Plug-in Content* выбрать ссылку *Dependencies* или выбрать закладку *Dependencies* в нижней части основного окна редактора. После этого вы увидите списки требующихся модулей (раздел *Required Plug-ins*, по умолчанию он пуст) и импортированных пакетов (раздел *Imported Packages*, по умолчанию будет содержать один пакет `org.osgi.framework`), см. рис. 3.6. Также вам будут доступны

инструменты анализа зависимостей (поскольку импортируемые модули так же могут зависеть друг от друга) *Dependency Analysis* и средство автоматического управления зависимостями *Automated Management of Dependencies*, однако в случае простых модулей эти средства не будут вам необходимы.

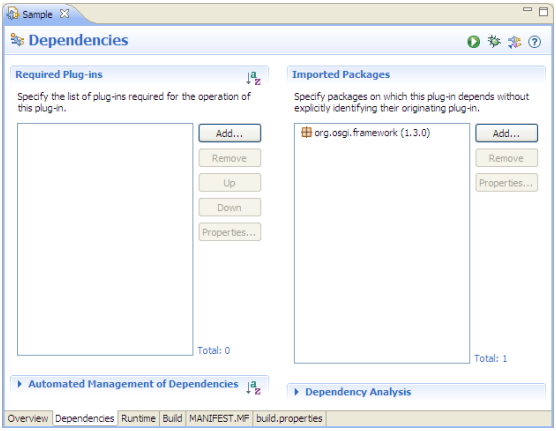

Рисунок 3.6. Окно редактирования зависимостей от других модулей

Для корректной работы вашего модуля, кроме пакета `org.osgi.framework`, также следует импортировать следующие пакеты:

- `org.eclipse.ecf.remoteservice.eventadmin`
- `org.osgi.service.event`
- `org.slavery.basic`
- `org.xdp.commons`
- `org.xdp.commons.interfaces`
- `org.slf4j`

Чтобы импортировать пакет, нужно в разделе *Imported Packages* нажать на кнопку *Add*, что приведёт к появлению окна выбора пакета (см. рис. 3.7). Имя пакета следует вводить в поле редактирования

Exported Packages: в верхней части окна. В нижней части окна при этом будет выводиться список пакетов, начало названия которых совпадает с введённым фрагментом, что значительно упрощает ввод полного имени пакета (по полному имени, например, можно сделать двойной щелчок мышью).

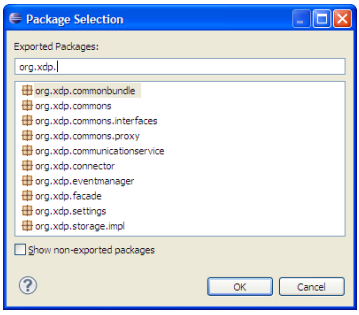

Рисунок 3.7. Выбор пакета для импортирования

После включения всех названных выше пакетов следует сохранить файл `manifest.mf`, после чего можно переходить к созданию классов вашего пакета.

3.2 Реализация класса-наследника Solver

Согласно модели технологии XDP, классы, непосредственно выполняющие вычисления в распределённой системе, являются наследниками класса `Solver`. Поэтому, чтобы ваша распределённая программа выполняла какие-то реальные действия, вы должны реализовать как минимум один свой такой класс. Рассмотрим его создание на примере.

Для создания нового класса можно сделать щелчок правой кнопкой мыши на имени пакета (в инспекторе в левой части рабочего окна) и в появившемся контекстном меню выбрать пункт *New/Class*. После этого появится окно выбора параметров нового класса (см. рис. 3.8). В нём следует обязательно выбрать имя нового класса (в

примере – `SampleSolver`), а также родительский класс. Последнее делается с помощью кнопки *Browse*, находящейся напротив поля ввода *Superclass:*. В появляющемся при этом окне можно вписать в поле ввода фрагмент названия класса, после чего в нижней части будут предложены доступные возможные варианты. Следует выбрать класс `org.slavery.basic.Solver`.

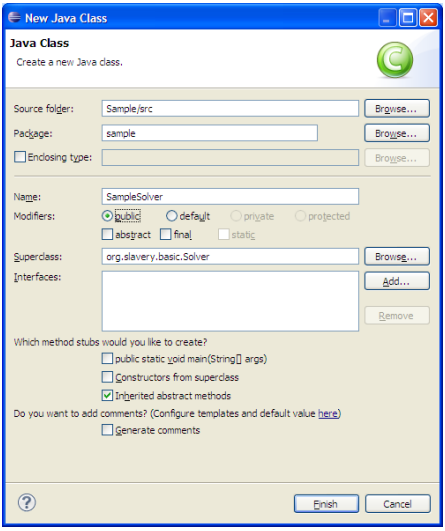

Рисунок 3.8. Создание нового класса-наследника класса Solver

После нажатия кнопки *Finish* в проекте появится новый файл, имя которого будет совпадать с именем класса. Также он будет открыт вам для редактирования.

Изначально класс не может быть откомпилирован, т.к. в нём присутствует только конструктор по умолчанию, а в родительском классе `Solver` конструктора без параметров нет. Можно реализовать конструктор самостоятельно, а можно воспользоваться одним из мастеров среды Eclipse. Для этого следует навести курсор мыши на подчеркнутое красной линии название класса `SampleSolver`,

после чего в появившемся контекстном меню выбрать пункт *Add constructor*. В коде при этом появится корректная реализация конструктора с параметрами.

Основные действия, выполняемые объектами класса в распределённой системе, должны быть реализованы в методе `perform()` вашего класса. В простом приложении типовая последовательность действий следующая:

- считывание исходных данных;
- выполнение вычислений;
- запись результата.

В листинге 3.1 приведён пример класса, вычисляющего значение синуса заданного числа.

Листинг 3.1. Класс-наследник Solver, вычисляющий значение синуса заданного числа

```
package sample;

import org.slavery.basic.Solver;
import org.xdp.commons.SessionID;
import org.xdp.commons.URI;

public class SampleSolver extends Solver {

    public SampleSolver(SessionID sid, String name) {
        super(sid, name);
    }

    @Override
    public void perform() {
        double d = (Double) storage.load(
            new URI(getName() + "/initdata"),
            getSessionID());
        double r = Math.sin(d);
        storage.save(new URI(getName() + "/resdata"),
            r, getSessionID());
    }
```

```
}
```

Для считывания исходных данных используется метод `load()` объекта `storage`, доступного наследникам класса `Solver`. Аналогично, для записи данных используется его метод `save()`. То, что при этом происходит передача сериализованных данных по сети, от программиста оказывается скрыто. Также в примере показано, как следует формировать путь для хранения данных с помощью класса `org.xdp.commons.URI`, и как при этом должны использоваться вспомогательные методы класса `Solver` `getName()`, возвращающий имя текущего экземпляра, и `getSessionID()`, возвращающий идентификатор текущей сессии. Такой подход позволяет различать данные, использующиеся в различных приложениях различными модулями.

Нетрудно заметить, что написание основного модуля распределённого приложения действительно сводится к наследованию класса, использованию готовых методов обмена информацией и, собственно, реализации бизнес-логики. Причём за счёт предоставляемых средств платформы первые две операции оказываются тривиальными.

3.3 Реализация класса-наследника Director

Аналогичным описанному выше образом следует также создать наследника класса `Director`. В модели XDP объекты этого типа отвечают за запуск модулей в распределённой системе, а также могут использоваться для загрузки и формирования начальных данных, а также для вывода результатов.

В листинге 3.2 приведён пример наследника класса Director, соответствующего наследнику класса Solver из листинга 3.1.

Листинг 3.2. Класс-наследник Director, формирующий начальные данные и собирающий результаты

```
package sample;

import java.util.List;
import java.util.ListIterator;
import org.slavery.basic.Director;
import org.xdp.commons.PlatformURL;
import org.xdp.commons.SessionID;
import org.xdp.commons.URI;

public class SampleDirector extends Director {

    public SampleDirector(SessionID sid, String name) {
        super(sid, name);
    }

    @Override
    public void perform() {
        List<PlatformURL> executorList =
            (List<PlatformURL>) storage.load(
                new URI(getName() + "/solverList"),
                getSessionID());

        int count = executorList.size();
        double max = Math.PI;
        double current = 0;
        ListIterator<PlatformURL> iter =
            executorList.listIterator();
        for (int i = 0; i< count; i++) {
            current = max / count * i;
            storage.save(
                new URI("solver" + i + "/initdata"),
                current, getSessionID());
            starter.startExecutor(iter.next(),
                new SampleSolver(getSessionID(),
               "solver" + i));
        }
        for (int i = 0; i< count; i++) {
```

```
                current = (Double)storage.load(
                    new URI("solver" + i + "/resdata"),
                    getSessionID());
                System.out.println(current);
            }
        }

    }
```

Как и ранее, основным методом класса-наследника `Director` является метод `perform()`. В примере в этом методе выполняются следующие действия.

Сначала путём считывания данных из общего хранилища находятся доступные реализации платформы, на которых может выполнятся наследник класса `Solver`, для этого используются объект `storage` и метод `getSessionID()`, что позволяет избежать конфликта с другими работающими на платформе приложениями. Также определяется общее количество доступных узлов.

Далее для каждого узла формируются начальные данные в виде числа от 0 до PI, с равным шагом в зависимости от числа узлов. Данные помещаются в хранилище с помощью метода `save` объекта `storage`.

После формирования данных запускаются исполнители. Для этого вызывается метод `startExecutor()` объекта `starter`, которому передаётся номер исполнителя, его объект (создаваемый экземпляр класса `SampleSolver`) идентификатор исполнителя.

Полученные в результате работы исполнителей данные считываются с помощью метода `load()` объекта `storage`, после чего выводятся в консоль.

Опять же, нетрудно заметить, что операции по занесению начальных данных в систему и по считыванию результатов являются типовыми, поэтому разработчик может сосредоточить свои усилия

на формировании представления начальных данных для исполнителей и на интерпретировании результатов их работы.

3.4 Реализация класса Activator

Данный класс является точкой входа модулей платформы OSGi и наследует от класса `org.osgi.framework.BundleActivator`. Его основными методами являются методы `start()` и `stop()`, запускаемые платформой, соответственно, при начале и при завершении работы модуля.

Поскольку мы воспользовались шаблоном OSGi при создании нашего модуля, в нём уже был создан класс с тривиальной реализацией, выводящей сообщения при начале и завершении работы. Для того, чтобы создаваемый модуль выполнял поставленные перед ним задачи, остаётся добавить в методы класса вызовы наших классов в рамках платформы XDP. Для этого обычно достаточно добавить в метод `start()` строчку следующего вида:

```
<Класс-директор>.registerTask("<TaskName>",
    <Класс-директор>.class)
```

вызывающую унаследованный из класса платформы XDP `org.slavery.basic.Director` метод `registerTask()` и передающую ему имя запускаемой задачи и рефлексивную ссылку на класс директора. В листинге 3.3 приведён пример класса-активатора для класса-директора из листинга 3.2.

Листинг 3.3. Класс-активатор

```
package sample;

import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;

public class Activator implements BundleActivator {

    public void start(BundleContext context)
```

```
                    throws Exception {
        System.out.println("Hello World!!");
        SampleDirector.registerTask("Sample",
            SampleDirector.class);
    }


    public void stop(BundleContext context)
                    throws Exception {
        System.out.println("Goodbye World!!");
    }

}
```

После написания класса-активатора код модуля нашего приложения для платформы XDP можно считать завершённым.

3.5 Запуск приложения на платформе XDP

Для запуска разработанного приложения из Eclipse потребуется скорректировать параметры запуска, создав новую конфигурацию запуска. Выберите в основном меню команду *Run/Run Configurations…*, после чего в левой части окна выберите вид запуска *OSGi Framework* и нажмите кнопку создания новой конфигурации (верхняя часть окна, иконка со знаком +). В итоге вам станет доступна новая конфигурация и редактирование её параметров (см. рис. 3.9).

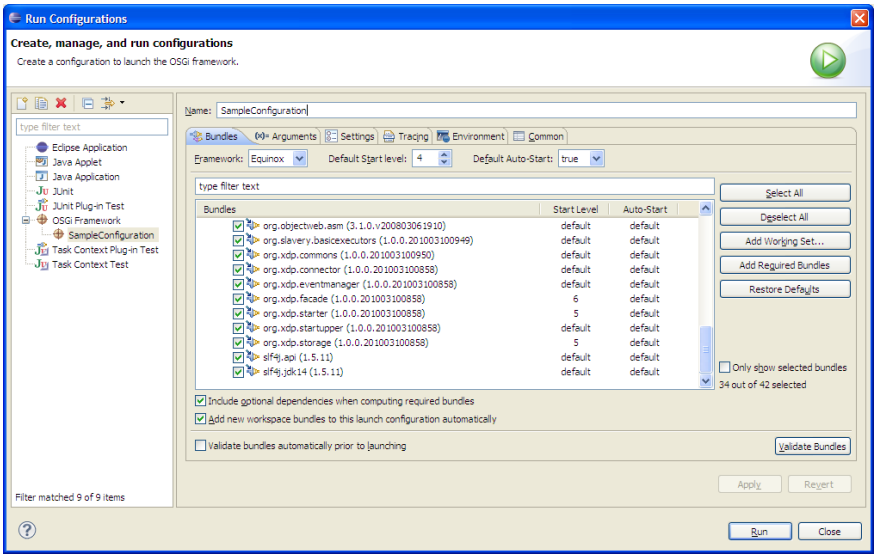

Рисунок 3.9. Конфигурация запуска

Вы можете задать имя конфигурации (это будет полезным, если вы используете несколько конфигураций), отредактировав поле ввода *Name:*.

Также вы должны изменить значения некоторых параметров используемых модулей, а именно изменить порядок запуска (значения *Start Level* в таблице со списком модулей) следующим образом:

- для модуля `org.xdp.storage` установите значение 5;
- для модуля `org.xdp.starter` установите значение 5;
- для модуля `org.xdp.facade` установите значение 6.

Также на вкладке *(x)= Arguments* вы должны добавить в раздел *VM Arguments:* строчку следующего вида:

```
-Dorg.xdp.connector.communication.port=NNNN
```

где `NNNN` – номер порта, указанного для вашего компьютера в файле конфигурации `hosts.conf` вашей платформы. Кроме того, в качестве рабочего каталога при запуске приложения следует указать

путь к папке, в которой была развёрнута платформа XDP. Пример заполнения этих параметров приведён на рис. 3.10.

После завершения редактирования конфигурации следует нажать кнопку *Apply* для её сохранения. Нажатие кнопки *OK* приведёт к запуску программы.

Далее вы можете запускать программу простой командой Run (команда основного меню *Run/Run* или кнопка с зелёным кругом с белым треугольником в панели быстрого запуска). Вывод из программы будет производиться в консоль, отображаемую при этом в нижней части окна Eclipse (см. рис. 3.11).

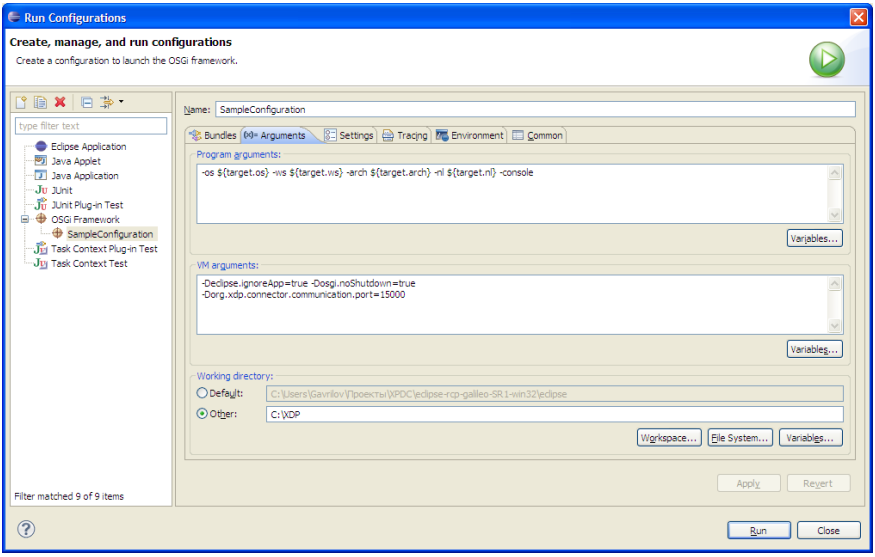

Рисунок 3.10. Конфигурация запуска, параметры запуска виртуальной машины Java

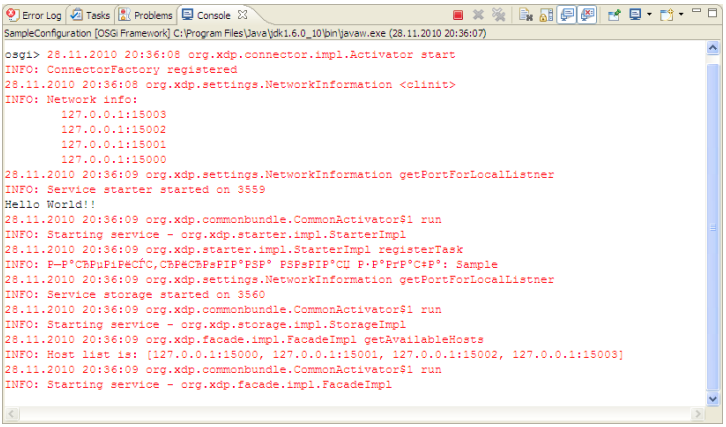

Рисунок 3.11. Пример вывода приложения в консоль при запуске

4 РЕКОМЕНДАЦИИ К ВЫПОЛНЕНИЮ КУРСОВОГО ПРОЕКТА

Безусловно, каждая конкретная задача и каждый конкретный проект, направленный на её решение, имеют свои особенности. Однако представляется целесообразным выделить следующие этапы выполнения курсового проекта с применением технологии XDP.

1. Получение задания. На этом этапе следует внимательно изучить задание, определиться с тем, какие данные и какими методами будут обрабатываться в ходе решения задачи.

2. Изучение платформы XDP. Перед реализацией программного комплекса как такового, следует изучить возможности платформы, написать и опробовать несколько простых распределённых приложений. При этом вы освоите базовые механизмы технологии, научитесь разрабатывать и запускать приложения для неё.

3. Реализация базовых алгоритмов на языке Java. На этом этапе вы должны будете выбрать модель представления данных, реализовать её средствами языка Java. Также следует написать программу, реализующую основной алгоритм, в виде обычного, не распределённого приложения, и отладить её на контрольных примерах. Связан данный этап с тем, что отладка распределённого приложения может оказаться достаточно затруднительной, поэтому желательно избавиться от потенциальных ошибок ещё до создания распределённой версии приложения.

4. Создание модели распределённого приложения. Перед созданием приложения также важно понять топологию и порядок работы элементов вашей распределённой системы. Здесь следует определиться с количеством и видами исполнителей (наследников класса `Solver`), порядком работы классов-наследников класса `Director`, видами передаваемых между модулями данных и формой их пред-

ставления, порядком передачи данных и т.д. Без чёткого понимания порядка и правил взаимодействия элементов распределённого приложения его написание может быть весьма затруднительным.

5. Реализация приложения по технологии XDP. На этом этапе следует реализовать классы-наследники, участвующие в работе модулей, в соответствии с разработанной на этапе 4 моделью. Функциональным наполнением классов исполнителей при этом станет программный код, разработанный на этапе 3.

6. Отладка распределённого приложения на локальном компьютере или в локальной сети. Здесь рекомендуется использовать некоторые тестовые примеры, не требующие больших вычислительных мощностей. Этап позволит вам убедиться в том, что разработанная на этапе 4 модель, реализованная на этапе 5, действительно позволяет эффективно решать поставленную задачу с помощью распределённого приложения. Также вы сможете устранить ошибки, возникшие при переходе от обычного приложения к распределённому.

7. Запуск на реальной вычислительной системе. На этом этапе вы должны запустить программу на платформе, сконфигурированной для работы с одним из вычислительных GRID-ресурсов СГАУ. Сложность решаемых задач при этом может быть значительной. Результаты работы программы следует сохранить для отчёта.

8. Написание отчёта о выполнении курсовой работы. В отчёте должны быть отражены следующие элементы:

- постановка задачи;
- алгоритмы и методы её решения;
- модель взаимодействия элементов распределённой системы (с применением диаграмм UML);
- краткое описание использованных элементов технологии XDP;
- результаты работы приложения на реальной вычислительной системе;
- исходный код программ (в приложениях).

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. **Хорошевский, В.Г.** Архитектура вычислительных систем [Текст] / В.Г. Хорошевский. – М.: МГТУ им. Н.Э. Баумана, 2005. – 410 с.

2. **Soblewski, M.** Sorcer: Computing and Metacomputing Intergrid [Текст] / Soblewski, Michael // Texas Tech University. – 2007. – P. 74–85.

3. **Седельников, М.С.** Алгоритм распределения набора задач с переменными параметрами по машинам вычислительной системы [Текст] / М.С. Седельников // Автометрия. – 2006. – №1. – С. 68-75.

4. **Голуб, Дж.** Матричные вычисления [Текст] / Дж. Голуб, Ч. Ван Лоун. – М.: Мир, 1999. – 512 с.

5. **Ортега, Дж.** Введение в параллельные и векторные методы решения линейных систем [Текст] / Дж. Ортега. – М.: Мир, 1991. – 342 с.

6. **Богданов, А.Б.** Сервис-ориентированная архитектура: новые возможности в свете развития GRID технологий / А.Б. Богданов, Е.Н. Станкова, В.В. Мареев. – Институт высокопроизводительных вычислений и интегрированных систем, 2005. – 32 с.

7. **Спецификация OSGi** [Электронный ресурс]. – Режим доступа: http://www.osgi.org/Specifications/

8. **Гаврилов, А.В.** Архитектура сервис-ориентированной распределенной вычислительной системы и ее реализация на основе фреймворка OSGI [Текст] / А.В. Гаврилов, И.И. Доровских, И.Д. Красинский // Научно-технический вестник СПбГУ ИТМО. – 2008. – Т. 54. – с. 113-119