РАЗРАБОТКА ТЕХНОЛОГИИ И ИНСТРУМЕНТАРИЯ ДЛЯ СОЗДАНИЯ РАСПРЕДЕЛЁННЫХ СИСТЕМ НА ОСНОВЕ УНИФИЦИРОВАННОЙ СТРУКТУРЫ КАРКАСОВ СИСТЕМ И ПРИЛОЖЕНИЙ

Карымов И.Л., Князьков К.В., Крючкова Е.Н.
Алтайский государственный технический университет им. И.И. Ползунова
(г. Барнаул)

Распределённые системы являются принципиально отличным классом программных комплексов, способных решать задачи, с которыми системам других архитектур справиться не под силу.

Рост производительности, снижение стоимости сетевого оборудования и аппаратной части обычных компьютеров привели к тому, что всё более широкий круг программистов (небольшие компании) получает возможность создавать распределённые системы для эффективного решения повседневных задач. Зачастую существующая инфраструктура организации среднего размера (например, ВУЗа) уже может стать базой для создания мультикомпьютерной гетерогенной среды. Однако, для того, чтобы процесс создания распределенных систем смог окончательно выйти за стены лабораторий и больших корпораций, необходимо появление нового поколения инструментов и технологий, которые изменили бы облик процесса их разработки.

Такая технология должна удовлетворять определенным требованиям: позволять программисту сосредоточиться на написании функциональной части приложения, не отвлекаясь при этом на решение сложных моментов реализации распределенных систем, таких как масштабируемость, репликация, параллельная обработка, балансировка нагрузки, учет реальных физических условий (отказ оборудования и программного окружения, адресации узлов, топология сети).

Авторами предлагается технология и ее реализация, которая позволяет обеспечить выполнение перечисленных требований. Идея предлагаемого подхода состоит в совместном использовании теоретической модели при проектировании распределенной системы, каркаса при разработке и системы исполнения при эксплуатации.

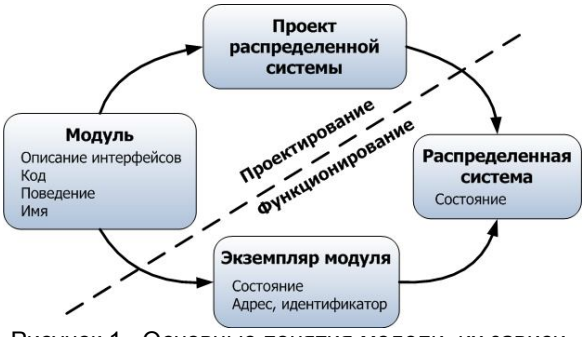

Рисунок 1 - Основные понятия модели, их зависимости и характеристики

Основополагающими понятиями теоретической модели являются: проект распределенной системы, модуль, экземпляр модуля и распределенная система (рисунок 1).

Проект распределенной программной системы представляет собой композицию из набора взаимодействующих модулей. **Модуль** подобен классу в объектно-ориентированных языках программирования: он включает в себя внутренние данные, операции работы с данными и набор сигнатур, описывающих множество запросов, на которые он может отвечать (интерфейсы). Но, в отличие от класса, модуль имеет собственный код, является самостоятельным приложением и не может являться родительским классом для других модулей. Известно понятие активного объекта [1], которое можно сравнить с понятием модуля. Активный объект - это объект, являющийся также процессом: у него есть собственная исполняемая программа. С этой точки зрения, модуль похож на активный объект, он одновременно является «хранилищем услуг» и процессом, который имеет собственный план решения задачи. Итак, далее под модулем будем понимать абстракцию, характеризуемую (рисунок 2):

- описанием своих интерфейсов и внешних вызовов, которые специфицируют возможности взаимодействия с другими модулями;

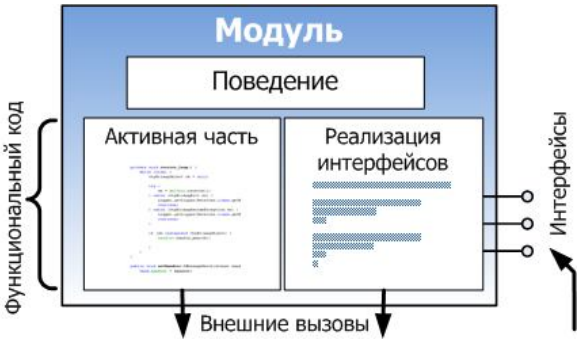

Рисунок 2 - Абстракция модуля

- шаблоном поведения, представляющим собой ряд правил, по которым система исполнения будет работать с модулем;
- функциональным кодом, состоящим из активной части и реализации интерфейсов (пассивной части).

Модуль может иметь внешние интерфейсные методы, которые предоставляются другим модулям для вызова. **Интерфейсный метод** - это функция модуля, которая имеет множество входных параметров и один результат. Для описания интерфейса модуля необходимо задать имя метода, типы его параметров, тип результата и область его видимости. Каждый модуль в терминах ООП является классом для своих **экземпляров**. Любое взаимодействие модулей возможно только на уровне классов, т.е. модуль может вызвать метод или послать сообщение только классу (другому модулю), а не конкретным его экземплярам.

Помимо базовых типов взаимодействия (вызов интерфейсного метода и отправка сообщения) существуют несколько модификаторов множественности вызова, которые позволяют распараллеливать операции на несколько экземпляров модулей:

- вызов метода на произвольном экземпляре модуля;
- параллельный вызов метода на нескольких экземплярах модуля с разными параметрами;
- параллельный вызов метода на всех экземплярах;
- параллельный вызов нескольких разных методов на разных модулях.

Поведение модуля представляет собой декларируемое свойство, позволяющее придать модулю некоторые шаблонные особенности элемента распределенной системы, например:

- масштабируемость — возможность запуска дополнительных экземпляров модуля;
- реплицирование — необходимость создания пассивных копий модуля (с поддержкой непротиворечивости их состояний [3]) для обеспечения замены экземпляра модуля в случае его отказа;
- сериализуемость — возможность сохранения текущего состояния модуля;
- переносимость — возможность переноса модуля в распределенной среде с сохранением его внутреннего состояния.

Под **функциональным кодом модуля** будем понимать программный код, который реализует программист для воплощения логики работы системы. Функциональный код имеет две составляющих: реализацию активной части и реализацию интерфейсных методов. Реализация активной части - это программный код, который начинает исполняться при запуске модуля. Реализация интерфейсного метода - это программный код, который выполняется при вызове интерфейсного метода модуля и преобразует входные параметры в результат с возможной модификацией внутреннего состояния экземпляра модуля. Работа кода обработки интерфейсов и активного кода производится параллельно. При написании кода все межмодульные вызовы интерфейсных методов выглядят прозрачно, как локальные.

Основополагающим понятием в процессе функционирования распределенной системы является состояние системы. **Состояние распределенной системы** в определенный момент времени характеризуется расположением запущенных экземпляров модулей на узлах и информацией о нагрузке на каждый модуль, машину и канал связи. К возможным вариантам изменения состояния относятся такие действия, как запуск и удаление экземпляров модулей на некоторой машине, перенос экземпляра с одной машины на другую с сохранением состояния, создание реплики экземпляра модуля, сохранение состояния модуля. Переход из состояния в состояние возможен только согласно поведениям модулей. Рассмотрим пример влияния поведения модуля на изменение

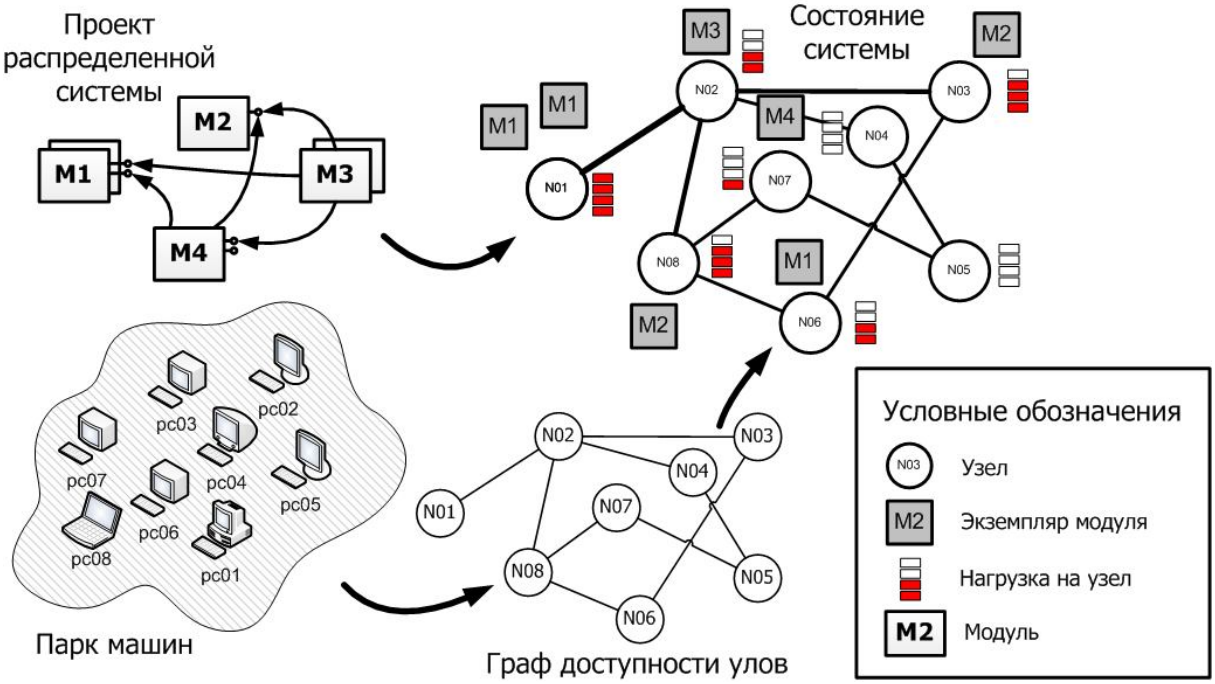

Рисунок 3 - Состояние системы

состояния системы. Если модуль масштабируемый, то во время функционирования системы может быть создано множество экземпляров этого модуля. Количество экземпляров зависит от нагрузки на данный модуль, от имеющейся вычислительной мощности и от некоторых эвристических решений по поводу резервирования вычислительной способности системы.

Стратегия изменения системы относится к вопросу автоматического управления функционированием системы и предполагает некоторые общие правила изменения состояний, приводящие систему к более эффективному состоянию.

Использование параллельно работающих модулей дает возможность удобной декомпозиции задачи в тех случаях, когда проблемная область «естественно» разделяется на набор параллельно работающих и взаимодействующих процессов. Модули снижают сложность реализации параллелизма в рамках распределенной системы. Прозрачность и простота представленной модели позволяют легко описывать с её помощью сложные системы, но при этом она полностью отстраняется от таких обязательных моментов реализации, как механизм обнаружения, именования, распределения (планирования) ресурсов и коммуникации.

В качестве средства, позволяющего создавать распределенные программные системы в терминах предложенной теоретической модели, нами был разработан **программный комплекс**, состоящий из системы исполнения и каркаса программной системы.

Система исполнения обеспечивает использование ресурсов вычислительной среды, адресацию, службу именования, а также поддержку абстракций теоретической модели (модуль, поведение модуля, вызов). Она состоит из двух основных частей: программ агентов и подсистемы управления. Программы-агенты образуют промежуточный слой (middleware) между вычислительной средой и модулями системы. Этот слой обеспечивает распределенной системе возможность использования ресурсов вычислительной среды и при этом скрывает их природу и физическое место положения. Вычислительной средой для программ, написанных на основе предложенной технологии, в общем случае, является гетерогенный комплекс, базовые аппаратные компоненты которого представляют собой компьютерные ресурсы организации, объединенные средствами локальной сети. Основными функциями промежуточного слоя являются:

- объединение распределенных узлов в единую вычислительную среду;

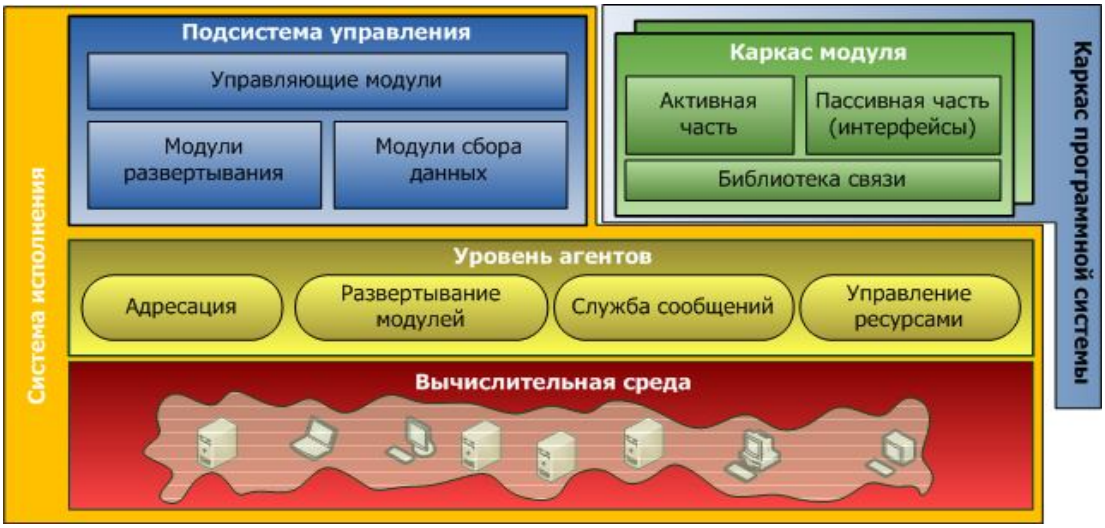

Рисунок 4 - Архитектура программного комплекса

- обеспечение служб именования и адресации;
- обеспечение службы межмодульных сообщений;
- управление развертыванием экземпляров модулей;
- мониторинг и сбор информации об используемых ресурсах.

Подсистема управления обеспечивает такие функции, как управление агентами, сбор информации о функционировании всей системы и ее частей, управление приложениями, а также предоставляет пользовательский интерфейс для администрирования системы исполнения и наблюдения за ее работой (рисунок 5).

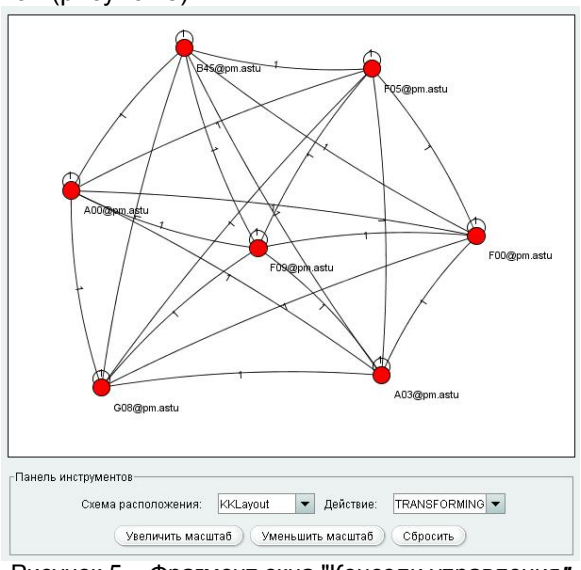

Рисунок 5 - Фрагмент окна "Консоли управления"

Рассмотрим процесс разработки распределенной системы с точки зрения программиста. Главной идеей, положенной в основу предлагаемого подхода, является предоставление программисту возможности создавать эффективную распределенную систему, работая лишь в терминах абстракции модели и явно кодируя только её функциональный код. Для этого был создан инструмент, позволяющий автоматизировать и ускорить некоторые этапы цикла разработки - каркас. Каркас – это набор взаимодействующих классов, описывающих повторно применимый дизайн некоторой категории программ. Каркас задает архитектуру приложения, разбивая его на отдельные классы с четко определенными функциями и взаимодействиями. Разработчик настраивает каркас под конкретное приложение путем порождения подклассов и составления композиций из объектов, принадлежащих классам каркаса [2]. Каркас программной системы необходим для интеграции разрабатываемых приложений в систему исполнения. Он состоит из трех основных частей:

- служебной библиотеки, отвечающей за связь с системой исполнения;
- библиотеки, генерируемой на основе спецификаций и содержащей иерархию классов для сокрытия деталей реализации прозрачности интерфейсов;
- шаблонов классов, содержащих функциональный код.

Для генерации каркаса в автоматическом режиме был разработан программный продукт - генератор кода. Генератор представля-

ет собой консольную программу, работающую в нескольких режимах: создание

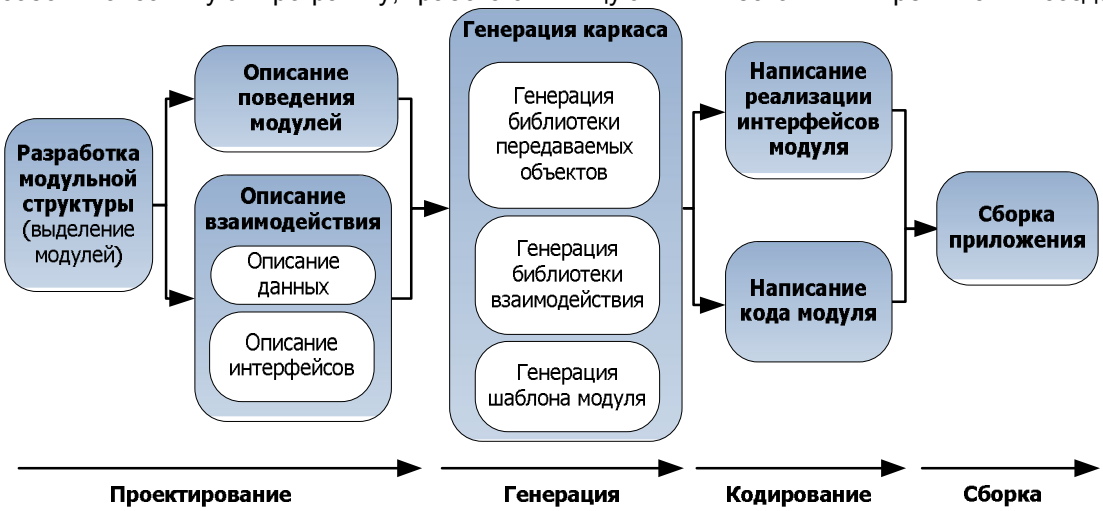

Рисунок 6 - Цикл разработки распределенной системы

приложения, создание модулей, генерация проекта, сборка. В основу работы генератора был положен принцип специфицирования свойств распределенной системы и ее частей. Каждый режим опирается на определенный набор спецификаций. После стадии генерации кода создается полностью работоспособный проект, который может быть без каких либо изменений импортирован в среды программирования Java, например, в Eclipse.

Для отладки всего приложения с использованием интегрированных средств разработки и для отладки вручную на локальной машине был предусмотрен и реализован специальный механизм. После этапа сборки каждого модуля создается готовое к отладке приложение и пакет программной системы, готовый к загрузке в систему исполнения.

На рисунке 6 представлен цикл разработки распределенной системы с применением инструментария.

Можно выделить несколько направлений развития проекта:

1. Разработка типичного реального приложения и его обширное тестирование на производительность, нагрузку, и сравнение с аналогичными системами.

2. Совершенствование системы исполнения, так как от качества её реализации напрямую зависят такие ключевые параметры создаваемых приложений, как быстродействие, надежность, масштабируемость. Поэтому целесообразно в ходе дальнейших исследований обратить внимание на совершенствование ее компонентов: поддержка стратегий при автоматическом управлении распределенными системами; алгоритмы принятия решений, используемых управляющей подсистемой (внедрение алгоритмов, базирующихся на идее самообучающихся систем); библиотеки инструментальных средств; децентрализацию подсистемы управления (применение механизма голосования и выборов для организации устойчивости ее архитектуры).

3. Модульная структура компонентов системы стимулирует программиста к повторному использованию кода, а также дает возможность применять уже готовые модули, реализующие типовые компоненты (например: распределённая файловая система; высокопроизводительные алгоритмы сортировки, поиска и шифрования). На этой почве можно сформулировать еще один вектор развития – разработку библиотек активных компонентов, которые выполняют типичные задачи и часто могут быть использованы в распределенных приложениях.

СПИСОК ЛИТЕРАТУРЫ

1. Мейер, Бертран. Объектно-ориентированное конструирование программных систем / Бертран Мейер. — М.: Издательско-торговый дом Русская редакция, 2005. — 1232 с.

2. Приемы объектно-ориентированного проектирования. Паттерны проектирования / Э. Гамма, Р. Хелм, Р. Джонсон, Д. Влиссидес. — СПб.: Питер, 2009. — 366 с.

3. Распределенные системы. Принципы и парадигмы / Э. Таненбаум, М. ван Стеен. — Спб.: Питер, 2003. — 877 с.