



МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«МИРЭА - Российский технологический университет»

РТУ МИРЭА

Институт Информационных технологий

Кафедра инструментального и прикладного программного обеспечения

(ИиППО)

ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ №1

по дисциплине

«Архитектура программных продуктов и систем»

Выполнил студент группы ИКБО-13-17
Принял

Комов Д.И.
Куликов А.А.

Лабораторные работы выполнены

«__» 2021 г.

(подпись студента)

«Зачтено»

«__» 2021 г.

(подпись преподавателя)

Москва 2021

Цель практической работы

Ознакомиться с теоретическим аппаратом MDA подходом при разработке и проектировании архитектур программного обеспечения.

Задание

1. Необходимо проанализировать концепцию «архитектура, управляемая моделью», дав краткое теоретическое обоснование и исторические предпосылки возникновения, выделив преимущества и недостатки данного подхода и определив те условия, при которых, по вашему мнению, использование данного подхода оправдано.
2. Провести анализ литературных источников и подобрать библиотеки, платформо-генераторы, которые поддерживает Model Driven Architecture (MDA) и дать им обобщенную характеристику.

Выполнение

1. Концепция MDA (Model Driven Architecture) призвана обеспечить общую основу для описания и использования большинства существующих стандартов, не ограничивая разработчиков в выборе конкретных технологий. Интеграция стандартов достигается за счет: введения концепции платформно-независимой модели приложения; использования унифицированного инструмента (UML) для описания таких моделей; наличия разработанных OMG стандартных отображений моделей в среду большинства технологических платформ и программных инструментов промежуточного слоя.

Предпосылкой к ее созданию послужило потребность в интеграции и обеспечении взаимодействия систем, основанных на разных технологиях, а

также в модернизации существующих программ и их переработке в соответствии с новой технологической основой

Достоинства MDA

Тестирование и модификация

Пользуясь MDA, можно организовать не только переход от абстрактной модели к детальной (от PIM к PSM, от PSM к коду системы), но и обратное преобразование — повышение уровня абстракции. Возможно создание инструментов, позволяющих осуществлять не только прямое, но и обратное преобразование моделей на основе стандартных отображений. Благодаря этому открывается возможность вести разработку, тестирование и модификацию одновременно платформно-независимой и платформно-зависимой моделей; если возникает необходимость изменить логику работы программы на абстрактном уровне (т.е. изменить PIM), эти изменения могут быть отражены в изменения PSM.

Платформно-независимая модель имеет достаточно высокий уровень детализации и является исполняемой, что позволяет тестировать систему еще до начала ее практической реализации — на уровне требований к системе и технического задания, с самых первых этапов дизайна. Это очень важное достоинство, так как обычно ошибки, возникшие на ранних стадиях проектирования, очень трудно исправить на более поздних стадиях, после реализации прототипа системы.

Построение нескольких PSM по одной PIM

MDA существенно облегчает создание программной системы сразу на нескольких платформах промежуточного слоя. При создании PIM разработчик получает модель системы, которая не зависит от технологий и деталей реализации. Если применить к PIM несколько стандартных отображений на различные платформы, можно получить несколько платформно-зависимых моделей

системы. После необходимой доработки по этим моделям можно получить реализацию системы на нескольких технологических платформах. При этом, поскольку абстрактная логическая модель у этих реализаций общая, существенно уменьшается риск ошибки и расхождения в функционировании различных реализаций.

Разумеется, такой способ разработки гораздо более эффективен, чем традиционный подход, так как можно многократно использовать одну и ту же платформно-независимую модель, вместо того чтобы разрабатывать реализацию системы на каждой платформе «с нуля».

Простота модернизации и смены платформы

Не обязательно разрабатывать систему на нескольких платформах сразу: MDA облегчает перенос разработанной системы на новую технологическую основу. Если разработчик сохранил платформно-независимую модель системы, то при необходимости в дальнейшем можно отобразить эту модель на нужную платформу. При этом, как и при одновременной разработке на нескольких платформах, сохраняется логическая целостность и совместимость со всеми остальными реализациями, сделанными на основе данной платформно-независимой модели (разумеется, подобие имеет место только на уровне архитектуры и бизнес-логики: детали реализации могут отличаться).

Используя PIM и соответствующее отображение, всегда можно будет с относительной легкостью заново реализовать и модернизировать систему на основе новейших технологий и платформ. Создатели MDA называют это свойство *future proofing* — защищенностью от устаревания ее технологической платформы.

Недостаток MDA:

Ограниченнность этой архитектуры языком UML. В междисциплинарном

проекте мы не можем ожидать, что все специалисты-пользователи говорят на UML, даже если мы расширим эти языки специфичными для предметных областей стереотипами

2.На данном этапе работы мы рассмотрим два инструмента, позволяющих внедрять MDA в программные продукты: BOLD, Spring Roo

BOLD – инструмент, позволяющий внедрять MDA в проекты на языке программирования Delphi.

Его основные возможности:

1. Присутствует встроенный текстовый редактор UML-моделирования для создания моделей приложения;
2. Импорт и экспорт UML-моделей из Rational Rose - CASE
3. Автоматическая генерация экранных форм для просмотра и редактирования данных;
4. Поддержка создания многозвенных приложений и тонких клиентов на базе DCOM.
5. Поддержка подмножества языка UML - OCL (Object Constraint Language);
6. Возможность хранения базы данных в XML-документе без использования СУБД;
7. Автоматическая генерация баз данных практически для всех реляционных СУБД, существующих в настоящее время (доступных через интерфейсы BDE, ADO, dbExpress);
8. Поддержка модификации базы данных с сохранением информации (DataBase Evolution);
9. Автоматическая генерация программного кода на языке Object Pascal;

Spring Roo — фреймворк с открытым исходным кодом для быстрого со-
зания бизнес-приложений на Java. Полученные в результате приложения ис-
пользуют общие технологии Java, такие как Spring Framework, Java Persistence
API, JSP, Apache Maven и AspectJ.

Данный инструмент позволяет внедрить MDA в проекты на языке прош-
граммирования Java.

Spring Roo отличается от других аналогичных приложений по следую-
щим основным причинам:

1. Юзабилити: Оболочка Roo разработана с целью обеспечить простую
для использования среду разработки, которая сводит к минимуму требования
предварительного обучения. Аннотации Roo всегда начинаются с @Roo (в ко-
мандной строке в IDE). Кроме того, пользователи могут редактировать про-
граммные файлы Roo, когда IDE не работает

2. Нет лишних библиотек: Roo не использует API среды исполнения и не
требует наличия различных системных библиотек. Это гарантирует, что нет
связанного с Roo потребления ресурсов процессора, диска и оперативной па-
мяти. Код оптимизирован для компактного развертывания облачных вычисле-
ний и множества вариантов использования масштабируемости.

3. Производительность платформы Java: Roo обеспечивает для Java-раз-
работчиков производительность их решений. Пользователь должен использо-
вать только Java. Roo использует основные стандарты и технологии бизнес-
приложений, чтобы максимизировать удобство разработки поверх уже гото-
вого кода.

На рисунке 1 представлен пример архитектуры приложения, построен-
ного с помощью AndroMDA.

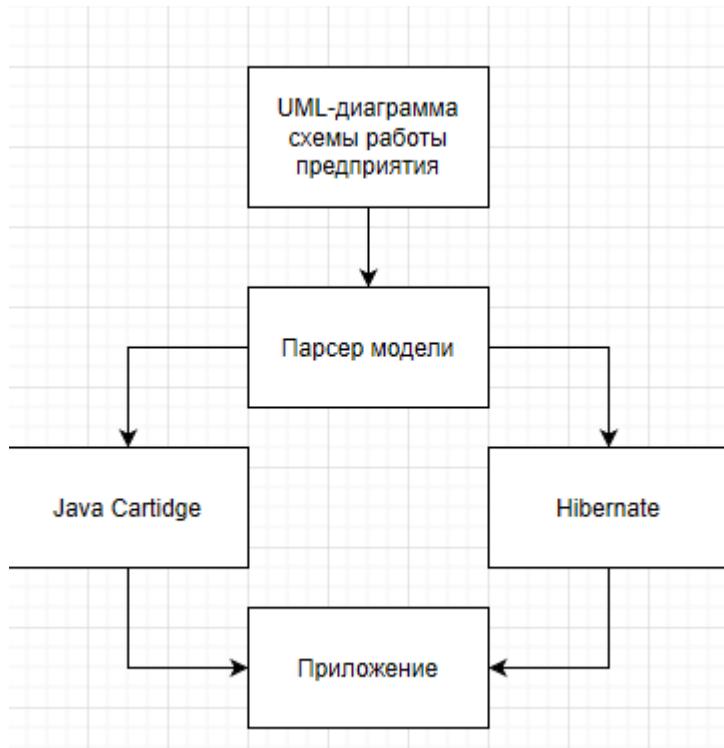


Рисунок 1 – Архитектура приложения

Вывод

В результате выполнения было проведено ознакомление с теоретическим аппаратом MDA подходом при разработке и проектировании архитектур программного обеспечения.