**Практическая работа № 4**

**Архитектура, управляемая моделью (Model-driven architecture, MDA)**

**Цель работы:**

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

**Теоретические сведения**

Последние годы наиболее часто используемым подходом к проектированию программного обеспечения (ПО) является объектно-ориентированный подход (ООП) . Разработанный в 80-х годах XX века он активно развивается. В основе подхода лежат понятия класса (типа), объекта (экземпляра класса), метода (некоторой реализуемой функциональности) и атрибута (характеристики). Система, построенная согласно данному подходу, представляет собой совокупность взаимодействующих между собой объектов. Задачей разработчика является разработка эффективной иерархии классов для реализации приложения, основываясь на трех основных принципах ООП: полиморфизме, инкапсуляции и наследовании.

Первые среды, поддерживающие ООП, реализовывали функции упрощенного создания синтаксических конструкций (классов, методов, атрибутов). Кроме того, поставщиками сред стали разрабатываться библиотеки классов, реализующих сервисные функции в объектно-ориентированном стиле (например, скрывающие вызовы операционной системы (ОС), предоставляющие возможность упрощенной работы с пользовательским интерфейсом и др.). Со временем сложность поставляемых библиотек росла в интересах поддержки новых технологий разработки, создания распределенных приложений, работы с Internet. Кроме того стала очевидна низкая степень совместимости созданных на их базе сред и приложений. В результате по мере увеличения степени сложности систем все более явными становились сложности создания объектно-ориентированных приложений: 

* высокий риск ошибок, заложенных при дизайне системы из-за недостаточного опыта исполнителя или высокой степени сложности ПО; 
* сложности при разработке распределенных приложений и интеграции созданных ранее систем из-за отсутствия унифицированных способов сетевого взаимодействия;
* низкая степень совместимости сред разработки и приложений, построенных на базе платформ от разных поставщиков; 
* сложности при переделке систем как результат исторического развития архитектуры и низкого качества или отсутствия документации.

Перечисленные проблемы привели к удорожанию разработки и осознанию необходимости поиска решения озвученных проблем. Одним из способов решения первой проблемы является использование шаблонов проектирования и описанных архитектурных подходов. Шаблоны проектирования являются важным этапом эволюции подходов к разработке, расширяя концепцию повторного использования результатов разработки на архитектурный уровень. Они активно используются на практике, а знание шаблонов проектирования стало одним из критериев для определения профессионального уровня разработчика. В тоже время, использование шаблонов привело к появлению новой проблемы. Недостаточно обдуманное применение шаблонов разработчиками низкого уровня приводит к перегруженности архитектуры, даже если она построена как совокупность известных шаблонов проектирования. Другим способом снижения риска ошибок на стадии разработки архитектуры является ее моделирование. В этом случае можно говорить о визуализации разрабатываемой архитектуры посредством некоторого программного средства и методологии моделирования. Такие средства и модели имели независимое развитие и получили широкое распространение не только в сфере разработки ПО, но и за ее рамками. В частности широкое распространения получило моделирование бизнес процессов (Business Process Modeling, BPM). Моделирование способствует решению третьей и четвертой проблемы из списка выше. В области разработки ПО средства моделирования постепенно эволюционировали от средств визуального представления к инструментам с поддержкой автоматической генерации кода и далее к комплексным интегрированным средам. Концепция ООП к архитектуре приложений нашла свое развитие в компонентно-ориентированном подходе. Главным отличием последнего является использование понятия компонента, а не класса или объекта в качестве базовой конструкции. Компонент – это составная единица программной системы, четко заданная на уровне интерфейса и связей с другими компонентами. Компонентно-ориентированный подход можно рассматривать как основу для сервис-ориентированного подхода, являющегося в последние несколько лет одним из наиболее популярных. Как результат развития перечисленных направлений выделился ряд архитектурных подходов, отражающих концепцию построения сложного ПО и соответствующие практические рекомендации. Часто эти подходы не являются взаимоисключающими и дополняют друг друга. Model Driven Architecture (MDA, архитектура управляемая моделью) – это архитектурный подход к построению многокомпонентного ПО, основанный на разработке независимого от платформы и языка программирования представления системы (модели) с последующим переходим к исходному коду системы через автоматизированную генерацию кода. Разработка MDA началась в 2000 году. Предпосылкой к разработке являлся тот факт, что уже к концу 90-х годов прошлого столетия было создано большое количество технологий и протоколов для создания распределенных приложений. В качестве примеров можно привести COM/DCOM, CORBA, Java/RMI, XML/SOAP/RPC и др. Большинство таких технологий созданы зависимыми от платформы, не все из них были описаны в виде стандарта. Это создавало большие проблемы при разработке ПО и интеграции компонент в единую инфраструктуру. Основной целью разработчиков MDA являлось создание архитектурного подхода, позволяющего снизить риск, вызванный различиями между и технологиями разработки программных систем и платформами. Для решения этой задачи в MDA вводятся понятия модели, зависимой от платформы (Platform Specific Model (PSM)), и модели, независимой от платформы (Platform Independent Model (PIM)). MDA предполагает создание PIM и последующий переход к PSM с использованием специализированных средств. Кроме того возможен переход от одной PSM модели к другой. Консорциум OMG как разработчик MDA не предоставляет никаких программных средств для обеспечения перехода от модели к модели. Предполагается, что они будут реализоваными силами сторонних разработчиков. Фактически, MDA – это концепция разработки, поддержанная группой стандартов (UML, MOF, CWM и XMI), разработанных OMG. Эти стандарты накладывают требования, которым должны удовлетворять модели, и предоставляют рекомендации для разработчиков сред создания ПО по MDA.

**Цель MDA**

В современных программных системах активно используются много различных стандартов и технологий промежуточного слоя — CORBA, DCOM, .Net, Web-службы, технологии, основанные на Java и т.д. Все чаще возникает 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. Новая архитектурная концепция, предложенная консорциумом OMG, основанная на модельно-ориентированном подходе к разработке программного обеспечения, позволяет существенно упростить и частично автоматизировать разработку, интеграцию и модернизацию систем. В связи с большим количеством используемых и разрабатываемых стандартов и технологий стало очевидно, что попытка создать единый универсальный стандарт построения и взаимодействия программных систем обречена на неудачу. Примером тому стандарт CORBA, который, хотя и приобрел определенную известность, так и не занял предполагаемого для него места «универсального стандарта». Поэтому консорциум OMG принял решение перейти от «стандарта интеграции» к интеграции стандартов.

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.

Использование MDA для разработки и интеграции программного обеспечения позволяет сохранить инвестиции, сделанные в разработку бизнес-логики даже при смене технологических платформ. Модельно-ориентированный подход облегчает интеграцию как разнородных, основанных на различных технологиях распределенных систем, так и организацию взаимодействия между системами, основанными на одной технологии, но использующими разные интерфейсы, сервисы и стандарты. Кроме того, MDA позволяет разрабатывать стандартные сервисы (репозитарии, сервисы событий и сообщений и т.д.), которые поддерживают сразу несколько технологий промежуточного слоя. Например, можно создать репозитарий объектов, который предоставляет соответствующий сервис системам как на базе CORBA, так и на базе Web-сервисов. Это не только упрощает разработку программного обеспечения, но и позволяет организовать взаимодействие различных систем.

**Ядро MDA**

Ядром MDA являются несколько стандартов — UML, MOF, CWM и XMI. Язык UML (Universal Modeling Language) используется для описания всех моделей. Совокупность метамоделей CWM (Common Warehouse Model) представляет наиболее часто используемые в базах данных и инструментах бизнес-анализа метаданные. CWM содержит большое количество различных метамоделей, описывающих функционирование бизнеса. MOF (Meta Object Facility) — общий абстрактный язык для описания метамоделей; на его основе построены формальные описания метамодели для CWM и UML. Последний стандарт, XMI (XML Metadata Interchange), играет служебную роль, описывая отображение моделей MOF и UML на стандарт XML. При этом метамодели преобразуются в DTD-структуру документа, а модели — в тело XML-документа. Это позволяет объединить модель и ее метамодель в одном документе и получить так называемый «самоописываемый» (selfdescribing) документ, содержащий не только данные, но и информацию, необходимую для их интерпретации.

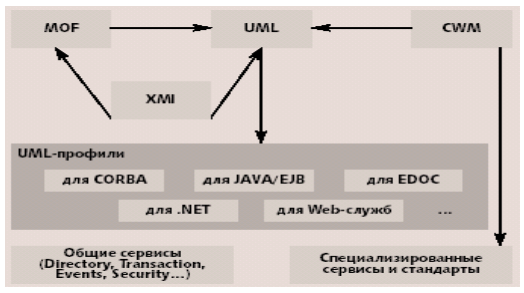


Рисунок 1-

В основе MDA лежит понятие платформно-независимой модели (platform independent model, PIM). Речь идет о детальной исполняемой модели на языке действий UML (action semantics) с пред- и постусловиями, сформулированными на OCL. Вполне логично, что для описания PIM выбран язык UML. Этот язык принципиально позиционировался как независящий от платформ и технологий. Однако UML в своих ранних версиях не являлся «точным» языком: он только предоставлял дизайнеру возможность описать структуру и поведение системы, практически не определяя ее функционирование и используемые алгоритмы. В новый стандарт UML 2.0, который был одобрен OMG в июне 2003 года, включено большое количество средств, позволяющих описывать внутреннюю организацию и функционирование системы. Одна из основных задач, которые были решены при создании этого стандарта, — превратить UML в алгоритмически полный исполнимый язык, в то же время, по возможности, не повышая уровня детализации UMLмоделей. Несмотря на то, что PIM — это детальная исполняемая модель, ее вряд ли можно использовать на практике как финальный программный продукт. UML-инструментарий нового поколения предоставляет унифицированную среду, в которой можно интерпретировать эту модель и получать исполняемый код прототипного качества. Такой код может быть крайне неэффективным, не удовлетворять некоторым функциональным требованиям, не полностью реализовывать функциональность системы и даже требовать участия человека в процессе исполнения. Инструментарий, используемый для интерпретации UML-модели, может быть полуавтоматическим и допускать вмешательство оператора. Только после привязывания к конкретной платформе можно получить код промышленного качества. Но, хотя исполнение PIM-модели и нельзя применять для решения практических задач, оно необычайно важно для целей тестирования и отладки. Фактически, у разработчиков появляется возможность получить первый прототип системы еще до начала стадии кодирования, когда сравнительно легко вносить даже существенные изменения в систему (в том числе, изменения требований и технического задания). Многие крупные производители объявили о поддержке MDA и начале разработки соответствующих инструментов, и в ближайшее время можно ожидать выхода первых инструментариев разработки с использованием MDA. Но, вероятно, потребуется значительный период времени, чтобы инструменты развились и смогли максимально использовать возможности технологии. Пока это не сделано, значительную часть рутинной работы придется выполнять вручную.

**Архитектура CWM**

**CWM**

Спецификация Common Warehouse Metamodel (Общая метамодель Хранилища данных, далее CWM) определяет метамодель (модель модели данных), представляющую как бизнес, так и технические метаданные, которые в большинстве случаев присутствуют в области технологии Хранилищ данных и бизнес аналитики. Она используется как основа для обмена экземплярами метаданных между гетерогенным программным обеспечением, поставляемым различными производителями. Системы, которые "понимают" метамодель CWM, обмениваются данными в форматах, которые согласуются с этой моделью. CWM выражен на языке UML (Unified Modeling Language, Унифицированный язык моделирования). Но, хотя UML является нотационным основанием для определения CWM, CWM расширяет базовую метамодель UML с помощью концепций технологий Хранилищ данных и бизнес-анализа. Можно сказать, что CWM расширяет язык UML в том смысле, что каждый метакласс (metaclass) CWM наследуется напрямую, либо ненапрямую из метаклассов UML. Таким образом, CWM можно характеризовать как язык определенной области применения, предназначенный для определения моделей Хранилищ данных. CWM определяющая метамодель для обмена экземплярами метаданных между различными средствами моделирования и информационными системами.

**MOF**

Другой стандарт OMG - Meta Object Facility (Средство метаобъекта, MOF) - определяет общие интерфейсы и семантику для взаимодействующих метамоделей. MOF - это пример мета-метамодели, или модели метамодели (подмножество UML). Он также определяет набор IDL-преобразований (Interface Definition Language, язык описания интерфейса, который устанавливает спецификацию интерфейса для обнаружения и управления моделей с помощью программных APIs). Помимо определения общей семантики для метамоделей MOF также служит в качестве модели для UML (то есть в конечном итоге MOF определяет язык, на котором выражается метамодель UML). Поскольку CWM наследуется из UML, MOF также является моделью и для CWM. Все модели CWM выражаются на UML и реализуют семантику MOF. MOF состоит из четырех уровней, верхний из которых соответствует мета-метамодели. Третий уровень соответствует метамодели (например, UML). Второй уровень соответствует экземпляру UML модели, описывающее какую-либо систему. Самый нижний уровень отражает экземпляры моделируемых объектов.

**XMI**

Наконец, третий стандарт, который непосредственно задействован в обмене метамоделями - это XMI. XMI (XML Metadata Interchange, Обмен метаданными XML) - это стандарт OMG, который устанавливает правила преобразования метамоделей MOF в XML. XMI определяет, как использовать XML-теги для представления сериализованных моделей, совместимых с MOF. Метамодели MOF транслируются в XML DTD, а модели - в XML-документы, которые согласуются со своими DTD. XMI описывает отображение метамоделей, построенных согласно MOF в XML, который в свою очередь может являться основой для обмена метаданными.

**Представление метамодели**

Каждая метамодель CWM представляется как XML DTD (в соответствие с правилами XMI), так и определение IDL. В первом случае 16 модели CWM преобразуются в поток (serialize), после чего ими обмениваются, как документами XMI. При экспорте метаданные посредством XMI-документа, необходимо выполнить XMI-преобразование (MXI-rendering) в форме, которая легальна по отношению к DTD. При импорте данных с помощью XMI-документа, следует проверять модель на допустимость по этим DTD. Во втором случае моделей объектов CWM создаются в памяти или хранятся в репозитории - в этой ситуации IDL предпочтительней, поскольку он определяет необходимые интерфейсы, подписи методов и структуру совокупности , которые эта модель должна поддерживать.

Итак, CWM фактически состоит из ряда составных метамоделей (или суб-метамоделей), которые организованы в виде следующих 4 слоев: базовый слой (Foundation), источники данных (Resources), анализ (Analysis) и управление Хранилищем (Management) . Базовый слой состоит из метамоделей, которые поддерживают моделирование таких различных элементов и сервисов, как типы данных, 17 системное преобразование типов, абстрактные ключи и индексы, выражения, бизнес-информация и включения программного обеспечения, основанного на использовании компонентных объектов. Слой источников данных предоставляет возможность моделировать существующие и новые источники данных, в том числе реляционные базы данных, ориентированные на запись базы данных (record oriented databases), а также XML- и основанные на объектах (object-based) источники данных. Слой анализа предоставляет средства для моделирования сервисов информационного анализа, которые обычно используются в Хранилище данных. Он определяет метамодель для преобразования данных, OLAP, визуализации информации/репортинга (business nomenclature) и data mining. Слой управления состоит из метамоделей, представляющих стандартные процессы и операции Хранилища данных, журнализации (activity tracking) и планирования работ [scheduling] (например, ежедневной загрузки и выгрузки). Этот набор метамоделей, предоставляемых CWM, достаточен для моделирования всего Хранилища данных. Используя инструмент, поддерживающий CWM, можно было бы сгенерировать экземпляр Хранилища данных прямо из модели Хранилища данных. Каждый из этих различных инструментов использует те части модели, которыми можно воспользоваться. Например, сервер реляционной базы данных задействует реляционный блок этой модели и будет использовать его для построения его каталога. Аналогично OLAP-сервер будет отыскивать в модели метаданные OLAP и использовать их для определения многомерной схемы. А инструмент извлечения, преобразования и загрузки данных (ETL) скорее всего обработал бы срез модели Хранилища данных, которая охватывает несколько метамоделей CWM, в том числе метамодели OLAP, преобразования, типа данных, преобразования типов, выражений и реляционную метамодель.

**Технология Model Driven Architecture (MDA), суть, перспективы использования**

Model Driven Architecture — модельно-ориентированный подход к разработке программного обеспечения. Суть этой технологии состоит в построении абстрактной метамодели управления и обмена метаданными (моделями) и задании способов ее трансформации в поддерживаемые технологии программирования (Java, CORBA, XML и др.). Архитектура MDA предлагает новый интегральный подход к созданию многоплатформенных приложений и базируется на трех основных элементах (рисунок 2): 

UML(Unified Modelling Language) унифицированный язык моделирования;  MOF (MetaObject Facility) абстрактный язык для описания информации о моделях (язык описания метамоделей); 

CWM (Common Warehouse Metamodel) общий стандарт описания информационных взаимодействий между хранилищами данных.

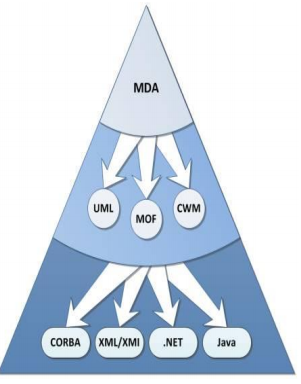


Рисунок 2-

На центральном уровне структуры находится собственно MDA, которая «разворачивается» наружу посредством второго уровня, содержащего вышеперечисленные базовые составляющие UML, MOF и CWM, и на третьем, внешнем уровне представлены некоторые из широко известных в настоящее время программных платформ разработки: CORBA, XML, .NET, JAVA. На этом внешнем уровне, по замыслу OMG, могут и должны быть размещены и все возможные будущие технологии разработки. Этим подчеркивается тот факт, что OMG считает архитектуру MDA не просто новой технологией, а скорее «метатехнологией» создания приложений. Последняя отныне будет и единственно актуальной — вне зависимости от развития и появления новых средств разработки, которые MDA уже «заранее интегрировала» в себя. Само понятие «разработчик программного обеспечения» может при внедрении MDA довольно сильно видоизмениться. Со смещением акцента на создание модели разработкой приложений будут заниматься не столько программисты, сколько специалисты, владеющие описываемой предметной областью. Возможно, что также в какой-то степени «пострадает» традиционное деление специалистов на разработчиков баз данных и разработчиков приложений баз данных. Уже сейчас возможно при разработке MDA-приложений практически полностью абстрагироваться от знания конкретной СУБД; более того, во многих случаях нет необходимости и использовать язык SQL, поскольку многие инструменты MDA предоставляют возможность работать на более «высоком» уровне (бизнесуровне), где становится абсолютно не важным знание конкретной структурной схемы базы данных или состава полей ее таблиц. Однако программисты-разработчики вряд ли останутся без работы, так как, с одной стороны, создание MDA-инструментария само по себе является чрезвычайно интересной, сложной и объемной задачей для них. А с другой стороны, внедрение MDA уже сейчас избавляет и самих программистов от рутинной работы, передавая большую ее часть искусственному программному интеллекту — инструментам реализации MDA.

**Жизненный цикл разработки с помощью MDA**

Жизненный цикл разработки с помощью MDA не сильно отличается от традиционного. Он разделен на такие же этапы. Одно из существенных различий состоит в природе артефактов, которые создаются во время процесса разработки. Здесь артефакты являются формальными моделями, то есть (говоря примитивным языком), моделями, понятными компьютерам. Далее рассмотрим три вида моделей, из которых состоит ядро технологии MDA.

**Типы моделей**

Архитектура MDA описывает и структурирует поэтапный процесс разработки любых программных систем на основе создания и использования моделей. При этом используется несколько типов моделей, создаваемых и преобразуемых на различных этапах разработки. Процесс разработки по MDA это последовательное (поэтапное) продвижение от одной модели системы к другой. При этом каждая последующая модель преобразуется из предыдущей и дополняется новыми деталями. Модели, общая схема разработки и процесс преобразования моделей - ключевые составные части архитектуры.

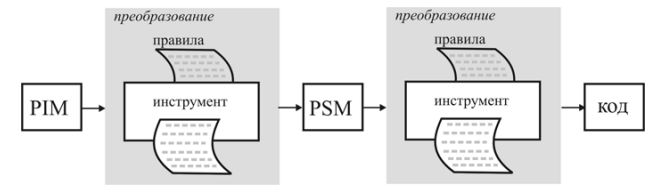


Рисунок 3 -

Основная идея MDA в том, что преобразование из PIM в PSM, а также же генерация кода из PSM, может производиться автоматически. Преобразования проводятся при помощи инструментов преобразования (transformation tools), которые в свою очередь используют правила преобразования. Эти правила будут написаны на языке, который будет описан стандартом QVT (Queries, Views, Transformations). Преобразования могут быть параметризованы, что позволит их подстраивать под нужды конкретных проектов. Рассмотрим типы моделей, используемых в архитектуре MDA.

**CIM**

Common Information Model (типовая информационная модель, CIM) — открытый стандарт, определяющий представление управляемых элементов IT среды в виде совокупности объектов и их отношений, предназначенный обеспечить унифицированный способ управления такими объектами, вне зависимости от их поставщика или производителя. В упрощенном виде CIM можно представить как способ, позволяющий нескольким участникам обмениваться информацией, необходимой для управления их элементами. Упрощение заключается в том, что CIM не только определяет представление управляемых элементов и управляющей информации, но и предоставляет возможность управлять ими и контролировать их работу. Управляющее программное обеспечение, созданное с использованием CIM, может работать с множеством реализаций этого стандарта без потери данных или сложных перекодировок. CIM разработан и опубликован Distributed Management Task Force. Связанный с ним стандарт Web-Based Enterprise Management (также разработанный DMTF), определяет реализацию CIM, включая протокол обнаружения и доступа. Стандарт CIM включает спецификацию инфраструктуры и схему. Cпецификация инфраструктуры определяет архитектуру и понятия CIM, включая язык определения CIM Schema (и любых её расширений), и способ отображения CIM на другие информационные модели, например SNMP. Архитектура CIM объектно-ориентированная, поскольку основывается на UML: управляемые элементы представляются классами CIM, любые отношения между ними представляются ассоциациями CIM, а наследование позволяет создавать специализированные элементы из более простых базовых. Cхема SIM - концептуальная схема, определяющая набор объектов и отношений между ними, представляющих общую основу управляемых.

элементов в IT среде. Схема охватывает большую часть современных элементов IT среды, например компьютеры, операционные системы, сети, подпрограммное обеспечение, сервисы и хранилища. Cхема определяет общий базис представления таких элементов. Поскольку большинство управляемых элементов для каждого типа элемента и его производителя имеют своё поведение, схема является расширяемой и даёт возможность производителям представлять специфический функционал сходным образом с базовым функционалом, определенном в схеме. На CIM основаны либо используют большинство остальных стандартов DMTF (так как WBEM или SMASH). Также он является основой стандарта SMI-S, предназначенного для управления хранилищами. Множество производителей предоставляют различные реализации CIM: 

* большинство операционных систем предоставляют реализацию CIM. Например, CIM реализован в семействе Microsoft Windows (WMI) и некоторых дистрибутивах GNU/Linux 
* CIM и WBEM активно применяется в области сетей хранения данных в виде основанного на CIM стандарта SMI-S, определенного ассоциацией SNIA  большинство производителей серверов сотрудничают с DMFT в рамках стандарта SMASH, основанного на CIM 
* DMTF разрабатывает стандарт DASH управления настольными компьютерами
* CIM описывает общие требования к системе, словарь используемых понятий и условия функционирования (окружение).

Модель не должна содержать никаких сведений технического характера, описаний структуры и функционала системы. CIM максимально общая и независимая от реализации системы модель. Спецификация MDA подчеркивает, что CIM должна быть построена так, чтобы ее можно было преобразовать в платформенно- независимую модель. Поэтому CIM рекомендуется выполнять с использованием унифицированного языка моделирования UML.

**PIM**

На этапе анализа на основании требований вырабатывается платформно независимая модель системы (PIM). Она привязана к постановке задачи и предметной области и не зависит от таких деталей реализации, как, например, язык программирования или тип базы данных (реляционная, объектная, иерархическая и т.д.). Платформенно-независимая модель (Platform Independent Model, PIM) описывает состав, структуру, функционал системы. Модель может содержать сколь угодно подробные сведения, но они не должны касаться вопросов реализации системы на конкретных платформах. Модель PIM создается на основе CIM. Для создания модели используется унифицированный язык моделирования UML. PSM Далее, на этапе дизайна будет осуществлен выбор деталей реализации: платформ, языков, распределенной или централизованной архитектуры. На основании эти решений PIM будет преобразована в соответствующие платформно зависимые модели (PSM). Для этого преобразования скорее всего будут использоваться готовые инструменты преобразования и библиотеки правил преобразования. Из одной PIM может быть сгенерировано несколько PSM. Например, одна из PSM может основываться на CWM метамодели для реляционых баз данных и описывать модель данных. В тоже время другая модель может, используя UML/EJB Mapping, представить PSI в терминах Enterprise Java Beans. Для стыковки разных PSM моделей в процессе PIM⇒PSM трансформации могут также быть сгенерированы так называемые «bridges» — связки между разными PSM моделями, сгенерированными из общей PIM модели [8]. 25 Платформенно-зависимая модель (Platform Specific Model, PSM) описывает состав, структуру, функционал системы применительно к вопросам ее реализации на конкретной платформе. В зависимости от назначения модель может быть более или менее детализированной. Модель создается на основе двух моделей. Модель PIM служит основой модели PSM. Модель платформы используется для доработки PSM в соответствии с требованиями платформы.

**Задание на практическую работу**

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.