СОВРЕМЕННЫЕ АРХИТЕКТУРНЫЕ ПОДХОДЫ К ПОСТРОЕНИЮ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ[1]

Борисов А.В., Куриленко И.Е.

Московский Энергетический Институт (Технический Университет), Россия

Последние годы наиболее часто используемым подходом к проектированию программного обеспечения (ПО) является объектно-ориентированный подход (ООП) [1]. 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;

[1] Работа выполнена при финансовой поддержке РФФИ

- 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.

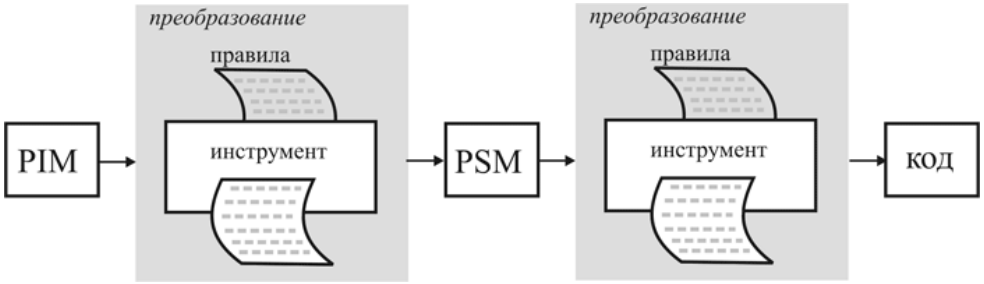
Концепция ООП к архитектуре приложений нашла свое развитие в компонентно-ориентированном подходе. Главным отличием последнего является использование понятия компонента, а не класса или объекта в качестве базовой конструкции. Компонент – это составная единица программной системы, четко заданная на уровне 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 реализованными 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.

Meta Object Facility (MOF) – это мета-метамодель (модель для описания метамоделей). 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.

Common Warehouse Model (CWM) – спецификация, определяющая метамодель для обмена экземплярами метаданных между различными средствами моделирования и информационными системами.

XML Metadata Interchange (XMI) – спецификация описывает отображение метамоделей, построенных согласно MOF в XML, который в свою очередь может являться основой для обмена метаданными.

Совокупность стандартов MDA MOF достаточна для определения моделей и метамоделей без введения дополнительных уровней абстракции.

Event Driven Architecture (EDA) – это архитектурный подход, основанный на понятии события и его обработке [2]. Система, построенная по EDA, представляет собой совокупность слабосвязанных компонент, обменивающихся сообщениями. Действие в системе выполняется как реакция на такое сообщение. Отправитель сообщения может не знать о том, какой компонент системы будет его обрабатывать.

Приложения, построенные по EDA, содержат следующие компоненты:

- генератор событий – осуществляет первичную обработку сообщений и выделяет из потока события, на которые требуется реакция.
- канал событий – обеспечивает транспортный уровень, т.е. доставку сообщений от компонента к компоненту.
- процессор событий – реализует логику обработки события, анализирует правила обработки, направляет события подписчикам.
- управление последующими событиями – обеспечивает реакцию на событие, фактически, формирует последствия наступления события.

В приложениях, построенных по EDA, в той или иной форме реализованы перечисленные компоненты, однако реализуемые ими функции могут быть совмещены. Выделяют три стиля обработки сообщений:

- *простая обработка сообщений* предполагает, что все поступающие сообщения значимые и требуют обработки.
- *обработка потока сообщений* предполагает, что не все сообщения, попадающие на вход, требуют реакции. Значимые сообщения выбираются и обрабатываются. Остальные сообщения могут использоваться в информационных целях или для анализа исключительных ситуаций.
- *сложный стиль обработки сообщений* предполагает, что на вход системы могут поступать сообщения разных типов и только определенная последовательность событий требует реакции.

Наиболее востребованными на практике являются приложения со сложным стилем обработки сообщений. В частности, он реализуется на базе таких продуктов как IBM WebSphere Business Events, TIBCO Business Events и др.

Service Oriented Architecture (SOA) – это архитектурный подход, основанный на понятии сервиса как базовой единицы построения системы. Обсуждение SOA затруднено тем, что до последнего времени нет единой трактовки этого термина. При наличии общего понимания о SOA как архитектуре, основанной на сервисах, в различных источниках возникают несогласованности как в определении как самого понятия сервис, так и концепции в целом [3]. Можно выделить как минимум две трактовки SOA:

- SOA – это программная архитектура, основанная на построении системы как совокупности Web сервисов, реализованных на базе одной или нескольких платформ разработки и взаимодействующих посредством стандартизованного протокола взаимодействия и четко описанного на некотором мета-языке интерфейса;
- SOA – это идеология информатизации бизнеса, основанная на процессном подходе и локализации этапов бизнес-процессов посредством выделения сервисов, ассоциированных и обладающих описанным интерфейсом для взаимодействия.

Согласно первой трактовке – SOA это архитектурный подход, который напрямую проецируется на одну из платформ для разработки приложений (например, .Net или J2EE). При этом понятие сервиса ассоциируется с приложением типа Web-сервис, имеющим вполне определенный смысл с точки зрения программной реализации. Коммуникации при этом осуществляются через некоторый стандартный протокол, завернутый в HTTP (например, SOAP), или сам HTTP.

Согласно второй трактовке SOA - это способ проектирования и построения информационной архитектуры организации и подход к переносу бизнес-функциональности на информационную среду. Он предполагает выделение задокументированных и зафиксированных бизнес потребностей, обеспечиваемых информационной средой предприятия и их реализацию в виде совокупности слабосвязанных приложений (сервисов), функциональность которых определяется бизнес потребностями и задана интерфейсом (описанием на мета-языке). Сервисы при этом взаимодействуют по четко определенному стандартному протоколу.

Как видно, вторая трактовка более широкая, но менее привязанная к технологии реализации. В тоже время она не противоречит первой трактовке, обобщая ее.

SOA предполагает, что приложение может являться провайдером услуги или ее потребителем. Провайдер реализует некоторый интерфейс, а потребитель через этот интерфейс использует возможности сервиса. Помимо информации структурного характера SOA отражает общие подходы к расширению систем. Согласно SOA вновь реализуемые провайдеры будут либо реализовывать принципиально новую услугу, определяя тем самым новый интерфейс, либо будут поддерживать существующий интерфейс, реализуя услугу, аналогичную существовавшей ранее. При разработке приложения-потребителя осуществляется оркестрирование, т.е. координация вызовов сервисов провайдеров услуг, и обработка возвращаемого результата. Таким образом, в теории разработка нового приложения по SOA может осуществляться на языке моделирования бизнес процессов, фактически задающим алгоритм оркестрирования. Так как сервисы SOA реализуют некоторую бизнес функциональность, такого рода моделирование может быть осуществлено с минимальным участием технического персонала.

Рассмотрим моменты, которые объединяют различные трактовки SOA от разных авторов. Поход SOA характеризуется следующими основными признаками:

- системы, построенные по SOA реализованы как совокупность слабосвязанных компонент (сервисов);
- сервис реализует некоторый набор услуг, определенных в терминах, понятных потребителю;
- поведение сервисов строго определено и представлено интерфейсом;
- сервисы и поддерживаемые интерфейсы описываются на метаязыке;

- сервисы и интерфейсы могут регистрируются в глобальном или локальном репозитории сервисов, доступном для поиска стандартным способом;
- сервисы взаимодействуют между собой с использованием некоторого сетевого канала и стандартного языка взаимодействия;

Подход к SOA как к совокупности взаимодействующих Web сервисов следующим образом реализует описанные принципы:

- сервис, реализующий SOA, представляет собой Web сервис;
- web сервис реализует некоторый набор услуг, заданный на стандартном мета языке WSDL (Web Services Description Language), основанном на XML;
- поведение каждого Web сервиса для приложений- потребителей определяется описанном на WSDL интерфейсом, образуя контракт;
- Web сервисы могут быть зарегистрированы в репозиториях UDDI (Universal Description Discovery and Integration) для того, чтобы предоставить потребителю возможность себя найти;
- Web сервисы взаимодействуют между собой посредством протоколов SOAP HTTP;

В дополнение к приведенным инструментам и технология для реализации SOA разработан Business Process Execution Language (BPEL) – это – основанный на XML язык описания бизнес-процессов [4].

При использовании SOA предприятие получает широкий круг возможностей, обусловленных различными характеристиками данной архитектуры. Основной из возможностей, является скорость реагирования бизнеса на изменения окружающей среды и перестроение бизнес-процессов с минимальными затратами.

Осуществим сравнительный анализ рассмотренных концепций. MDA не определяет структурные аспекты разрабатываемых систем и характер взаимодействия. Это связано с тем, что MDA – это мета-мета-модель высокого уровня, т.е. модель, по которой может быть построена метамодель. Структурные аспекты и характер взаимодействия элементов определяются мета-моделями и моделями, реализованными по стандартам MDA.

Концепция SOA является в настоящий момент одной из наиболее популярных и активно продвигаемых. Прилагаемые усилия привели к результату, и архитектурный подход стал понятен заказчику. В связи с этим термин SOA активно используется маркетологами и продукты,

реализующие EDA, часто выходят под флагом SOA.

В тоже время на уровне архитектурной концепции SOA и EDA это различные подходы. SOA предполагает, что приложению, запрашивающему услугу известен его ее провайдер, взаимодействие осуществляется по принципу запрос-ответ. В случае с EDA действия инициируются цепочкой событий, сгенерированных в среде. Отправитель события не знает, будет ли сообщение обработано и если будет, то кем. Однако, как отмечено выше, EDA совместима с SOA, и, в силу известности SOA, программные продукты, реализующие EDA часто входят в пакеты продуктов SOA. Несмотря на это, при построении прикладных систем архитекторам так или иначе придется сделать выбор из какой модели построения системы исходить.

К сожалению, в вопросах построения приложений нет единства и для части из них существуют различные трактовки. С другой стороны архитектурные парадигмы, заданные стандартами не всегда воспринимаются и реализуются производителями. Кроме того на уровне, концепции часто не рассматриваются шаги, обеспечивающие переход от уровня модели к уровню приложения и вопрос совместной работы информационных систем изначально построенных на базе разных концепций. Для архитектур, основанных на слабой связности элементов, не всегда задаются правила построения этих элементов, что с одной стороны дает свободу разработчикам, а с другой может привести к неоднородности и неуправляемости полученной архитектуры с точки зрения поддержки.

ЛИТЕРАТУРА

1. Бадд Т. Объектно-ориентированное программирование в действии / Перев. с англ. — СПб.: Питер, 1997.
2. K. Mani Chandy Event-Driven Applications: Costs, Benefits and Design Approaches, California Institute of Technology, 2006
3. Bell, Michael. "Introduction to Service-Oriented Modeling". Service-Oriented Modeling: Service Analysis, Design, and Architecture. Wiley & Sons. 2006, pp. 3. ISBN 978-0-470-14111-3.
4. Eric Newcomer, Greg Lomow. Understanding SOA with Web Services. Addison Wesley Professional 2004