УДК 681.3.06

# МОДЕЛИРОВАНИЕ ПРОГРАММНОЙ АРХИТЕКТУРЫ

**К. В. Ахтырченко**[1], **В. В. Леонтьев**[1]

Рассматривается моделирование программных архитектур — научно-практическая дисциплина, являющаяся составной частью программной инженерии. Дается определение программной архитектуры, целей ее применения при разработке программных средств. Обсуждаются существующие подходы к моделированию программной архитектуры, включая формализованные методы анализа и проектирования. Определяются перспективы дальнейших исследований в области моделирования программной архитектуры.

**1. Введение.** Становление моделирования программной архитектуры программных средств (в дальнейшем программной архитектуры [2, 7]) как научно-практической дисциплины приходится на период конца 1980-х – начала 1990-х гг. и может рассматриваться как результат эволюционного развития методологий и технологий разработки программных средств, сопровождающийся сменой парадигм в программной инженерии. Впервые термин "программная инженерия" был введен в 1967 г. Исследовательской Группой по компьютерным наукам Комитета по науке НАТО (the Study Group on Computer Science of the NATO Science Committee) для удовлетворения потребностей производства программных средств, основанного на тех видах теоретических, фундаментальных основ и практических дисциплин, которые являются традиционными в области инженерии. Программная инженерия имеет дело с разработкой и реализацией крупномасштабных программных систем на компьютерах различного назначения. Охватывает широкий диапазон тем, относящихся к управлению проектированием и разработкой высококачественных компьютерных программных средств, включая методологию программирования и управление проектами создания программных средств [1]. Неправильно было бы утверждать, что до этого периода программные средства не имели архитектуры. Программы, создаваемые в 1960-х г., делились на модули, программистами определялась логика их взаимодействия, идентифицировались глобальные свойства, которыми должна была обладать программа в целом [2]. Уже тогда при построении программ разработчики руководствовались определенными стратегиями организации программ, которые можно было бы рассматривать как архитектурные образцы [2, 7]. Однако применение этих образцов было преимущественно неформальным, поскольку не существовало методологий и средств, требуемых для их формального описания, использования и анализа.

Ситуация изменилась в связи с появлением в 1970-х г. повышенного интереса к проектированию программных средств (Software Design) со стороны исследователей. Это было обусловлено проблемами, возникшими у разработчиков при создании крупномасштабных программных систем в 1960-х г. [3]. В этот период явного акцента на решение проблем, связанных с моделированием программной архитектуры, не наблюдалось, поскольку основное внимание исследователей было направлено, в первую очередь, на определение проектирования как деятельности, отличной от реализации программных средств, специализированных методик, инструментальных средств и систем обозначения (в дальнейшем — нотаций или языков описания). Тем не менее, результатом исследований стало появление первых средств автоматизации проектирования (CASE (Computer-Aided Software Engineering) tools), применение которых упорядочивало процесс проектирования, но не обеспечивало четкую организацию процесса разработки программных средств в целом.

Поэтому в 1980-х гг. основной акцент переместился на создание таких подходов к разработке программных средств, которые охватывали бы все этапы жизненного цикла программной системы, а не только этап ее проектирования. Этот этап эволюционного развития программной инженерии характеризовался появлением большого количества нотаций и методик, разработанных для проектирования программных средств и нашедших успешное применение на уровне языков реализации (языков программирования), например, при структуризации программ. Другим значимым результатом стало возникновение и использование в промышленных разработках методов описания и анализа программных систем, позволяющих судить об их целостности, конформности какому-либо решению и т.п.

[1] Научно-исследовательский вычислительный центр, Московский Государственный Университет, 119899, Москва; e-mail: Akhtyrchenko@unis-ru.com

Конец 1980-х – начало 1990-х гг. характеризовались появлением новых информационных технологий, обеспечивающих организацию распределенных вычислений в неоднородных средах, что предоставило разработчикам множество альтернатив при разработке программного обеспечения. Примерами могут служить: технология промежуточного программного обеспечения, с 1989 г. разрабатываемая и стандартизируемая Object Management Group (OMG), технология менеджеров транзакций (transaction manager technology), промежуточное программное обеспечение, ориентированное на обмен сообщениями (message oriented middleware) [29]. Причина появления новых технологий во многом определялась той ролью, которую стали отводить информационным технологиям в различных компаниях, а также появлением и бурным распространением персональных компьютеров, их объединением в рамках корпоративных компьютерных сетей. Для многих организаций наличие соответствующих программных средств являлось необходимым условием доминирования на рынке, а иногда и выживания. Многие компании достигли конкурентных преимуществ благодаря использованию информационных технологий. В то же время экспансия персональных компьютеров в различной степени сделала доступными программные средства практически каждому сотруднику организаций. В 1990 г. в мире насчитывалось около 30 тысяч больших универсальных компьютеров, в то время как число персональных компьютеров составляло свыше 30 миллионов [23]. Результаты использования информационных технологий для персональных компьютеров делались очевидными: легкий в применении графический интерфейс пользователя, фактически неограниченный доступ к ресурсам персонального компьютера с индивидуального рабочего места и новое представление о компьютерах и программах, выполняемых на них.

Как результат, существенно расширился спектр задач, которые могли быть эффективно решены с использованием информационных технологий, что способствовало их широкому проникновению в различные сферы деятельности. Во многих организациях информационная система приобрела статус корпоративной, охватывая большое количество существующих в организации процессов. В условиях роста значимости информационных технологий при достижении конкурентных преимуществ и распространения персональных компьютеров произошло увеличение доли средств, выделяемых в организациях для информатизации своей деятельности. Так, если в 1970 г. организации в среднем тратили на информатизацию около 1 % своего бюджета, то в 1990 г. соответствующая доля бюджета составляла уже 3 % и к настоящему моменту продолжает увеличиваться [23]. Высокая стоимость проектов информатизации определила стремление организаций минимизировать вероятность их провала. Для разработчиков такое стремление проявилось в виде требования организаций-заказчиков предоставить им гарантии успешного выполнения проектов. Это делалось возможным, если уже на самых ранних этапах разработки были известны свойства (характеристики), которыми будет обладать создаваемое программное средство. С учетом увеличения размера и сложности программных средств это привело к тому, что проектирование, спецификация и анализ высокоуровневой структуры программного средства становились одним из самых важных аспектов процесса разработки, отодвигая на второй план многие другие виды деятельности, как, например, выбор алгоритмов и структур данных. В то же время в процессе исследований было установлено [7], что высокоуровневая структура системы позволяет осуществить не только анализ соответствия программных средств выдвинутым требованиям, но может выступать также в качестве базиса при решении различных проблем, возникающих при проектировании крупномасштабных программных систем, управлении и оценке процесса разработки, повторном использовании решений и др. Именно в этот период моделирование программной архитектуры стало выделяться в самостоятельную научно-практическую дисциплину, оставаясь при этом составной частью программной инженерии.

Хотя к настоящему моменту уже имеется солидный теоретический и практический базис, связанный с моделированием программных архитектур [2, 7, 11, 20], до сих пор многими специалистами-практиками в области информационных технологий, особенно в России, программная архитектура понимается исключительно на интуитивном уровне. Происходит упрощение и/или смешение как данного понятия, так и той роли, которую оно играет при разработке программных средств. Проектирование программной архитектуры в большинстве своем носит неформальный характер и часто подменяется использованием той или иной технологии. В связи с этим данная статья содержит: определение программной архитектуры, целей ее применения при разработке программных средств, обзор существующих подходов к моделированию программной архитектуры, включая формализованные методы анализа и проектирования, описание проблем и перспектив дальнейших исследований в области моделирования программной архитектуры.

**2. Определение программной архитектуры**. К сожалению, не существует универсального общепризнанного определения программной архитектуры. Данное понятие определяется различными способами [4]:

Криспен (Crispen): “Архитектура, как мы намереваемся использовать этот термин, состоит из (а) стра-

тегии разделения и (б) стратегии объединения. Стратегия разделения заключается в делении монолитной системы на дискретные, полностью покрывающие ее части (или компоненты). Стратегия объединения заключается в явном определении интерфейсов между этими компонентами".

Хэйез-Рос (Hayes-Roth): "Программная архитектура есть абстрактная спецификация системы, состоящая из основных функциональных компонентов, описываемых в терминах их поведения, их интерфейсов и межкомпонентного взаимодействия".

Клементс (Clements): "Программная архитектура определяется как организационная структура информационной системы, и включает компоненты, соединения, ограничения и логические обоснования. Компонентами могут быть маленькие кусочки кода, такие, как модули (процедуры), или большие составляющие системы, такие, как система управления базами данных. Соединения в архитектуре есть абстракции для описания того, как взаимодействуют компоненты в системе, например, через локальные вызовы процедур, каналы, вызовы удаленных процедур. Архитектура имеет различные ограничения и логические обоснования, ассоциированные с ней, включая ограничения на выбор компоненты и обоснование выбора того или иного специфичного компонента в конкретной ситуации".

Галэн и Пэрри (Garlan and Perry): "Архитектура есть структура компонентов программы/системы, их взаимосвязи, правила и руководящие принципы организации ее проектирования и дальнейшей эволюции".

Но несмотря на отличия, имеющиеся в определениях, в каждом из них делается акцент на структурные аспекты организации программных средств. Так, в соответствии с большинством определений программная архитектура определяет структурные элементы системы и механизмы их взаимодействия. Согласно [5] при моделировании программной архитектуры в сферу рассмотрения попадают высокоуровневая организация и структура системы, протоколы взаимодействия элементов системы, доступ к данным, декомпозиция функциональности по элементам проектирования, физическое распределение, композиция элементов проектирования, масштабирование и производительность, выбор проектных решений из множества альтернатив и т.п.

Приведенные определения отличаются тем, на решение каких проблем делается основной упор, что, в свою очередь, обусловлено теми свойствами (характеристиками) систем, которые интересуют в наибольшей степени исследователей и разработчиков при моделировании архитектуры. Это является одной из главных причин отсутствия унифицированного определения программной архитектуры. Если пытаться вывести "обобщающее и объединяющее" определение, то оно получится чрезвычайно громоздким и будет подвержено частым изменениям, поскольку спектр интересующих свойств программных средств постоянно расширяется.

Другой подход к унификации определения программной архитектуры заключается в выделении общей фундаментальной составляющей, определяющей природу понятия "программная архитектура". Поскольку из большинства определений следует, что программная архитектура определяет принципы организации программных средств и логику взаимодействия их составных частей, то наиболее релевантным понятием из теории систем является структура системы. Согласно [6] структура есть "строение и внутренняя форма организации системы, выступающая как единство устойчивых взаимосвязей между ее элементами, а также законов данных взаимосвязей". Связи, определяющие структуру системы, обеспечивают энерго-, массо- и информационный обмен между элементами системы, который, в свою очередь, определяет функционирование системы в целом и способы ее взаимодействия с внешней средой [28].

Тогда можно ожидать, что, независимо от становления и развития моделирования программной архитектуры как научно-практической дисциплины, архитектура будет определять основные элементы программного средства, взаимосвязи между ними. Учитывая, что обеспечиваемый связями энерго-, массо- и информационный обмен определяет функционирование системы в целом и способы ее взаимодействия с внешней средой, можно также ожидать, что посредством спецификации программной архитектуры могут быть описаны интерфейсы, определяющие поведение как программного средства в целом, так и его элементов в рамках некоторого сценария взаимодействия. Кроме того, можно сделать вывод, что при спецификации программной архитектуры основное внимание уделяется именно взаимодействию элементов, а не принципам их внутренней организации. Если в качестве общей фундаментальной составляющей взять структуру системы, то наиболее адекватным описанному подходу является определение, представленное в [7]:

"Программная архитектура программы или вычислительной системы есть структура или структуры системы, которые охватывают программные компоненты, видимые извне свойства компонентов, а также связи между компонентами".

Из определения следует, что программная архитектура определяет совокупность компонентов[2], образующих систему. Архитектура предоставляет информацию о том, как эти компоненты взаимодействуют друг с другом. При этом не рассматриваются выполняемые компонентами действия (вычисления), которые не характеризуют их взаимодействие. Любое программное средство, являясь системой, имеет архитектуру, так как может быть представлено как совокупность (композиция) компонентов и отношений между ними. Из определения также следует, что поведение компонента входит в состав архитектуры, поскольку оно обозримо с точки зрения остальных компонентов и позволяет им взаимодействовать друг с другом. И наконец, из определения вытекает, что программное средство может иметь более чем одну структуру. Действительно, распределенная информационная система как объект моделирования, с одной стороны, может быть рассмотрена как совокупность программных модулей, а с другой стороны, как совокупность параллельно выполняющихся процессов или потоков управления. Здесь важнейшим моментом системного подхода к моделированию объекта выступает категория цели [28], которая определяет интересующие свойства программного средства, а следовательно, создаваемую модель и соотносимую с моделью структуру. Свойство множественности системного (модельного) описания объекта в зависимости от целей этого описания является определяющим при моделировании программной архитектуры. В соответствии с возникающими потребностями могут быть введены различные структуры, требуемые для моделирования интересующих свойств (характеристик) программных средств. Применительно к множественности структур правомерна аналогия с человеческим организмом. Для него, в зависимости от предмета исследования, также определяют различные структуры, например, структуру кровеносной системы, структуру системы пищеварения, структуру нервной системы и т.д.

**3. Цели использования программной архитектуры**. Наличие тех или иных структур, являющихся составной частью программной архитектуры, обусловлено целями, которые исследователи и/или разработчики планируют достичь в процессе ее моделирования.

**3.1. Описание целей**. Обобщая [3, 7, 8, 9], можно утверждать, что использование программной архитектуры позволяет достигнуть следующих целей при разработке программных средств.

Программная архитектура обеспечивает достижение понимания программного продукта, коммуникации на высоком уровне проектирования, коммуникации между заинтересованными лицами. Для различных групп заинтересованных лиц, каждая из которых имеет свои интересы в проекте, чрезвычайно важно достигнуть понимания программного средства, осуществить коммуникации в рамках единого языка общения. Например, будущих пользователей интересует полнота и практическая значимость требований, которым будет удовлетворять система. Заказчика (покупателя) волнует возможность реализации проекта за приемлемое время, в рамках определенного бюджета. Менеджеру при управлении проектом необходимо обеспечить регламентированное и контролируемое взаимодействие команд разработчиков, например, посредством достижения слабой связанности выполняемых командами работ. В случае создания больших (крупномасштабных) программных средств каждому разработчику сложно достигнуть детального понимания всех программных компонентов системы, и поэтому его интересует, как минимум, понимание системы в целом. Программная архитектура обеспечивает достижение высокоуровневого понимания программного средства различными заинтересованными лицами (представителями со стороны заказчика, менеджерами высшего звена руководства, членами кросс-функциональной команды (маркетинг, качество и др.)), одновременно выступая в качестве языка общения (средства обмена знаниями) для различных групп заинтересованных лиц.

Программная архитектура воплощает получаемые на самых ранних этапах разработки решения, определяющие успех проекта и являющиеся базисом для принятия других проектных решений.

*Эффективное управление проектом* (*распределение работы*). Программная архитектура предусматривает декомпозицию системы на относительно независимые (слабо связанные) подсистемы/компоненты, для каждой из которых явно определены обеспечиваемая ей функциональность и интерфейсы взаимодействия с другими подсистемами/компонентами. Декомпозиция системы на слабо связанные составляющие позволяет организовать параллельное выполнение работ по созданию различных подсистем/компонентов, делает возможным рассмотрение деятельности по их реализации как самостоятельные задания (work tasks). При этом команды разработчиков взаимодействуют друг с другом в терминах интерфейсов, поддерживаемых основными компонентами. Это увеличивает вероятность правильного формирования организационной структуры (организационной архитектуры) проекта. Несмотря на то, что не только программная архитектура определяет организационную структуру, последняя часто является отраже-

[2] Здесь и далее, если это специально не оговорено, компонентами системы, в зависимости от уровня абстракции и детализации, а также интересующих свойств системы, могут быть как процедуры (функции), экземпляры классов, так и программные модули, являющиеся единицами компиляции, программные модули, выполняемые в рамках отдельного пользовательского процесса операционной системы, и, возможно, что-то другое.

нием программной архитектуры. Показано [3, 7], что программная архитектура упрощает оценку стоимости проекта, управление и оценку процесса разработки.

*Определение оптимальной структуры системы и ограничений на реализацию (достижение гибкой декомпозиции системы на составляющие).* Хорошая программная архитектура делает возможным гибкое распределение системы на взаимодействующие составляющие (подсистемы/компоненты), позволяя тем самым осуществить оптимальную декомпозицию как функциональности по подсистемам/компонентам, так и подсистем/компонентов по вычислительным узлам. При этом имеется возможность альтернативных декомпозиций без перепроектирования распределенных составных частей системы. Определяя структуру системы, архитектура накладывает ограничения на реализацию системы. Например, все решения, принимаемые на уровне проектирования с учетом реализации, должны быть конформны архитектурным решениям.

*Удовлетворение требованиям к системе и достижение поставленных целей.* Архитектура определяет возможность/невозможность достижения целевых системных характеристик качества (quality attributes), позволяет оценить и сопоставить требования, определить для них различные приоритеты. Если на архитектурном уровне какие-либо нефункциональные требования не были зафиксированы, то возрастает риск того, что созданная система не будет им удовлетворять. Программная архитектура делает возможным анализ свойств создаваемых программных средств, способствуя тем самым верификации на предмет удовлетворения требований. Программная архитектура позволяет оценить целостность создаваемого программного средства.

*Базис для обучения.* Архитектура может выступать в качестве базиса при обучении разработчиков, пользователей и др. Например, при подключении в проектную команду нового разработчика последнему нет необходимости изучать все детали, чтобы понять организацию программного средства. В большинстве случаев ему достаточно лишь изучить описание программной архитектуры.

*Уменьшение стоимости сопровождения системы.* Программная архитектура способствует минимизации стоимости сопровождения системы, так как она позволяет судить об изменениях, управлять этими изменениями. На основе анализа архитектуры могут быть сделаны с высокой степенью достоверности прогнозы основных видов изменений (например, локальные (уровень компоненты), архитектурные). На основе данных прогнозов могут быть разработаны мероприятия, позволяющие локализовать изменения моделей, кода, документации и др.

*Базис при прототипировании.* Программная архитектура выступает в качестве базиса при проектировании и эволюции прототипов программных средств.

Программная архитектура является базисом, используемым при разработке методологий и технологических решений, применяемых при проектировании программных средств. Программная архитектура становится базисом методологий и технологических решений, образуя класс Архитектурно-центричных (Architecture Centric) [14] процессов разработки программных средств. Программная архитектура может также выступать как инструмент исследования при разработке новых технологических решений.

Программная архитектура выступает в качестве базиса при анализе решений (программных средств). Программная архитектура может рассматриваться как инструмент исследования при анализе решений (программных средств). Анализ осуществляется при выборе решения (программного средства) из множества альтернативных, например, при выборе готовой информационной системы финансового и управленческого учета. На основе описания архитектуры может быть осуществлен анализ зависимостей между элементами программных средств, анализ характеристик программных средств (масштабируемость, расширяемость и т.д.).

Программная архитектура является распространяемой и многократно используемой абстракцией системы. Для достижения целей распространения и многократного использования решений программная архитектура

— позволяет определить неизменяемые (общие) и переменные составляющие для линии программных продуктов;

— делает возможным разделение функциональности, поддерживаемой компонентами, и механизмов межкомпонентного взаимодействия;

— делает возможным и уменьшает трудоемкость повторного использования и интеграции решений (отдельных существующих компонентов, каркасов (frameworks), библиотек классов, унаследованных приложений и программных продуктов третьих фирм), выступая в качестве обмена знаниями (средства описания решений для их повторного использования). Системы могут быть созданы из “больших”, вне проекта разработанных компонентов, которые являются совместимыми с предопределенной архитектурой. Одновременно с этим, программная архитектура поддерживает компонентную разработку на основе

шаблонов (Template-Based Component Development), например, параметризованных архитектурных стилей [20];

— способствует формированию словаря альтернатив, куда попадают решения, получаемые при проектировании архитектуры системы. Например, альтернативы могут быть описаны как архитектурные стили.

**3.2. Классификация целей**. Перечисленные выше цели можно разделить на организационные и технические, т.е. цели, достижение которых отражается на организации проекта, и цели, достижение которых отражается на свойствах программного средства. Например, цель "программная архитектура обеспечивает достижение понимания программного продукта, коммуникации на высоком уровне проектирования, коммуникации между заинтересованными лицами" является организационной, т.к. соотносится с эффективной организацией коммуникаций и обмена знаниями в рамках проекта. В то же время цель "определение оптимальной структуры системы и ограничений на реализацию (достижение гибкой декомпозиции системы на составляющие)" является технической, поскольку связана с определением структуры системы, ее программной реализацией. Отметим, что в рамках данной классификации некоторые из целей можно классифицировать и как технические, и как организационные. Примером может служить цель "уменьшение стоимости сопровождения системы". Она, с одной стороны, может рассматриваться как техническая цель, т.к. ее достижение предполагает минимизацию изменений в программном продукте. С другой стороны, достижение данной цели влияет на организацию процесса сопровождения, внесения изменений на уровне проектной команды и/или служб поддержки. С учетом описанных ограничений, классификация целей представлена в табл. 1. Подцели цели "распространение и многократное использование абстракций системы" не представлены в классификации, т.к. каждая из них является технической целью применения программной архитектуры, что соответствует классификации объемлющей их цели.

Таблица 1

Классификация целей применения программной архитектуры

| **Организационные цели применения программной архитектуры** | **Технические цели применения программной архитектуры** |
|---|---|
| Достижение понимания программного продукта, коммуникации на высоком уровне проектирования, коммуникации между заинтересованными лицами | Распространение и многократное использование абстракций системы |
| Эффективное управление проектом (распределение работы) | Формирование базиса, используемого при разработке методологий и технологических решений, используемых при проектировании программных средств |
| Базис для обучения | Формирование базиса при анализе решений (программных средств) |
| Удовлетворение системных требований и достижение поставленных целей | Определение оптимальной структуры системы и ограничений на реализацию (достижение гибкой декомпозиции системы на составляющие) |
| Уменьшение стоимости сопровождения системы | Удовлетворение системных требований и достижение поставленных целей |
| | Уменьшение стоимости сопровождения системы |

**4. Связь внешней среды и программной архитектуры**. Программная архитектура, являясь совокупностью структур программного средства, зависит от характеристик внешней среды, в рамках которой разрабатывается программное средство [7]. В свою очередь, сама программная архитектура созданного программного средства оказывает воздействие на внешнюю среду, реализуя принцип обратной связи. Действительно, цели организации, где программное средство разрабатывается и/или использу-

Таблица 2

Влияние внешней среды на программную архитектуру

| N | Источник возникновения внешних факторов | Влияние, оказываемое на программную архитектуру |
|---|---|---|
| 1 | Пользователи программного средства | Пользователи, которых в большей степени интересуют свойства программного продукта, в различной степени формируют функциональные требования к программному средству, определяют его качественные характеристики (надежность, интероперабельность, модифицируемость и др.) |
| 2 | Заказчик (покупатель) программного средства | Заказчик, интересы которого, в первую очередь, направлены на стоимость продукта и сроки его разработки, также формирует функциональные требования к программному средству и определяет его качественные характеристики |
| 3 | Организация, разрабатывающая программное средство | Для организации, разрабатывающей программное средство, можно выделить как минимум три различных случая возможных влияний на программную архитектуру:<br>— построение программного средства приемлемого качества за минимальный срок, что определяет ориентацию на уже существующие архитектуры;<br>— разработку линии программных продуктов, которая требует высокой степени повторного использования создаваемых программных компонентов, что в любом случае отразится на программной архитектуре;<br>— в ряде случаев, например, при организации разработки через субподрядчиков, требуется представить программное средство как совокупность в максимальной степени независимых подсистем, что, безусловно, сказывается на программной архитектуре |
| 4 | Технологическая инфраструктура | Технологическая инфраструктура, охватывая инструментальные средства, готовые программные средства от других производителей, стандарты, различные информационные технологии и методологии и т.п., накладывает ограничения на программную архитектуру |
| 5 | Разработчики программного средства | От знаний и опыта разработчиков программного средства, и, в первую очередь, программного архитектора, зависит, будет ли осуществлен поиск нового архитектурного решения, или вся деятельность ограничится только выбором решения из уже существующих |

ется, определяют необходимые свойства (характеристики) программного средства, которые выражаются в виде требований. Требования к программному средству влияют на определение структуры программного средства, а следовательно, на его программную архитектуру. Программная архитектура определяет правила организации программного средства, вводит ограничения на его реализацию. И наконец, программное средство предоставляет организации возможность улучшения производственных процессов, способствуя достижению существующих целей и формированию новых, которые, в свою очередь, определяют новые или дополняют старые требования к программному средству.

**4.1. Влияние внешней среды на программную архитектуру**. Программная архитектура формируется под воздействием внешних факторов. Источниками возникновения внешних факторов являются будущие пользователи и заказчик (покупатель) программного средства, организация, разрабатывающая программное средство, технологическая инфраструктура, сами разработчики программного средства, обладающие определенными знаниями и опытом [7]. Существуют следующие возможные влияния,

Таблица 3

Влияние программной архитектуры на внешнюю среду

| N | Объект влияния | Влияние, оказываемое программной архитектурой |
|---|---|---|
| 1 | Организационная структура разрабатывающей программное средство организации | В соответствии с программной архитектурой, а именно — декомпозицией программного средства на компоненты, могут быть сформированы структура работ и команды разработчиков |
| 2 | Корпоративные цели разрабатывающей программное средство организации | Имея перспективную программную архитектуру, организация может поставить цель создания программного продукта с аналогичной архитектурой, который будет иметь успех на рынке продаж |
| 3 | Требования к последующим версиям программного продукта | Программная архитектура может наложить ограничения на те изменения, которые заказчик желает увидеть в новой версии программного продукта |
| 4 | Корпоративные знания и опыт | Осваивая существующие архитектуры и разрабатывая новые, организация приобретает знания и опыт построения программных средств |
| 5 | "Технологическая культура" | Разработка новых программных архитектур может привести к изменениям, в том числе смену парадигм в программной инженерии. Примером могут служить те изменения, которые произошли в технологиях построения распределенных информационных систем вследствие появления Общей Архитектуры Брокера Объектных Запросов (CORBA) [26] |

оказываемые каждым из источников возникновения внешних факторов, на программную архитектуру (табл. 2).

**4.2. Влияние программной архитектуры на внешнюю среду**. Программная архитектура, реализуя принцип обратной связи, воздействует на внешнюю среду. Объектами влияния могут быть организационная структура разрабатывающей программное средство организации, корпоративные цели организации, требования к следующим версиям программного продукта, корпоративные знания и опыт, "технологическая культура" [7]. Существуют следующие возможные влияния, оказываемые программной архитектурой на внешнюю среду (табл. 3).

**5. Структуры программных средств.** Как было определено ранее, архитектура программы или вычислительной системы есть структура или структуры системы, которые охватывают программные компоненты, видимые извне свойства компонентов, а также связи между компонентами. Тем не менее, довольно часто о структуре системы судят только по ее функциональности, хотя имеются и другие свойства системы, как например, физическое распределение, межпроцессные коммуникации и синхронизация, которые должны быть представлены на уровне программной архитектуры [9]. В зависимости от интересующих свойств (характеристик) программных средств для программной архитектуры можно определить различные структуры или "взгляды" (представления (view)), которые в совокупности описывают программную архитектуру.

**5.1. Существующие структуры**. Обобщая взгляды на программную архитектуру из [7, 9, 10], можно выделить следующие основные структуры.

Концептуальная (логическая) структура (Conceptual or logical structure). Концептуальная, или логическая, структура включает множество абстракций, необходимых для описания функциональных требований к системе на абстрактном уровне, т.е. уровне, не затрагивающем вопросы реализации. Довольно часто данная структура строится на основе анализа проблемной области и является средством коммуникации между архитектором и экспертами проблемной области. В случае применения объектно-ориентированных методов анализа и проектирования данная структура представляет собой объектную модель программного средства.

Модульная структура (Module (Development) structure). Модульная структура определяет организацию программного средства как совокупности модулей — программных единиц, с которыми могут быть соотнесены рабочие задания, выполняемые членами проектной команды. Примером связи, устанавливаемой между модулями, может быть "является подмодулем". В зависимости от того, как осуществляется выделение модулей, какие виды межмодульных связей используются, имеются различные формы модульных структур. Примерами форм являются: группировка модулей в подсистемы[3], которые выступают базисом при распределении работ в проектной команде, а также многоуровневая иерархическая организация модульной структуры.

Процессная структура (Process structure). В то время как концептуальная и модульная структура определяют, в первую очередь, статические аспекты системы, процессная структура обеспечивает ортогональный "взгляд" на архитектуру, а именно, описывает поведение системы во время ее исполнения (runtime). Здесь в качестве элементов структуры выступают процессы и/или потоки управления (threads). Процессная структура охватывает создание, реконфигурацию, уничтожение, восстановление элементов, взаимодействие и синхронизацию элементов, устанавливая между ними связи типа "синхронизируется с", "не может быть запущен без", "не может быть запущен с", "имеет приоритет" и т.п.

Физическая структура (Physical structure). Физическая структура определяет отображение элементов программного средства на аппаратное обеспечение. Как правило, в рамках данной структуры осуществляется распределение процессов по объединенным в сеть вычислительным узлам[4] — элементам структуры. Моделирование физической структуры бывает особенно важно при распределенных вычислениях и параллельной обработке информации.

Описанные выше структуры являются наиболее распространенными [7, 9–11, 13–15]. В то же время при моделировании программной архитектуры используются и другие структуры, какими являются: структура "вызовов" (Call structure), структуры, определяющие потоки данных и потоки управления (Data Flow structure and Control Flow structure), структура использования (Use structure), структура классов (Class structure) [7]. Применение каждой из них соотносится с моделированием некоторых определенных программных свойств (характеристик) и обеспечивает отличный от других взгляд на программное средство.

**5.2. Совместное использование структур и их применимость**. Концептуальная и модульная структуры описывают, в первую очередь, статические аспекты программного средства, в то время как процессная и физическая структуры отражают его динамику [9]. Характеризуя программную архитектуру, данные структуры не являются независимыми по отношению друг к другу. Элементы одной структуры могут соотноситься с элементами другой структуры, выражая определенный структурный аспект организации программного средства. Примером проявления связей между структурами являются изменения в программном средстве потока управления, которое может потребовать изменений в процессной структуре, а также в физической структуре, если последняя не предусматривала требуемую вследствие изменения потока управления параллельную обработку данных.

Один из подходов, обеспечивающий совместное согласованное использование различных структур, был предложен в [11]. Он заключается в определении, помимо основных четырех структур, пятой, элементами которой являются сценарии — последовательности действий, выполняемых программным средством и являющихся абстракцией функциональных требований. Выполняя в рамках сценария последовательность действий, система позволяет получить пользователю[5] определенный результат. Следствием введения пятой структуры становится возможность показать, насколько правильно "работают" элементы различных структур в рамках некоторого сценария. Например, при использовании объектно-ориентированных методов моделирования для каждого из сценариев будут определены последовательности взаимодействий, соответственно объектов и процессов. Данный подход применен, например, в Унифицированном Процессе Разработки Программных средств (Unified Software Development Process) [14] и Архитектурном Методе Проектирования (Architecture Based Design Method) [13].

Не обязательно, чтобы при разработке программных средств использовались все структуры. Так, в случае небольшой программы, которая будет выполняться как один пользовательский процесс операционной системы и иметь один поток управления, процессная и физическая структуры могут не применяться. Концептуальная и модульная структуры могут быть совмещены, например, в виде объектной модели программы. С другой стороны, если в рамках программного средства предполагается реализовать распределенную обработку информации, обеспечить масштабируемость и повышенную надежность

[3] Например, в соответствии с существующими в проблемной области подсистемами.

[4] Компьютерам или процессорам.

[5] Пользователем системы может выступать элемент ее внешней среды, будь то человек или внешнее программное средство.

системы, то при моделировании программной архитектуры, возможно, потребуется использовать практически весь спектр имеющихся структур, а может быть, разработать новые структуры.

**5.3. Классификация структур**. Имеется несколько подходов к классификации структур программных средств.

В основу подхода, предложенного в [9], положены два признака классификации структур.

1) Различима (наблюдаема) ли структура во время исполнения (runtime) программного средства или нет? Различимой является используемая для трассировки подпрограмм структура "вызовов". Структура, описывающая параллельно выполняющиеся процессы или нити управления, также различима. Примером неразличимой структуры может быть модульная структура программного средства.

2) Описывает ли структура продукт (программное средство), процесс построения продукта или процесс использования продукта для решения каких-либо проблем? Модульная структура, как и структура "вызовов" подпрограмм, описывает продукт. Структура, определяющая правила взаимодействия пользователей с системой, не является описанием продукта.

Другие подходы, например [10, 12], разделяют структуры в зависимости от того, на какие аспекты архитектурного моделирования в них сделан акцент. Здесь примерами признаков классификации структур могут быть:

— ориентирована на моделирование данных;

— описывает поведение элементов программного средства;

— определяет функциональную декомпозицию по элементам;

— определяет структурную организацию системы;

— описывает реконфигурацию и эволюцию программных средств;

— определяет систему как иерархию функциональных компонентов.

**6. Процесс разработки и программная архитектура**. Процесс разработки программных средств (в дальнейшем – процесс разработки) определяет организацию и управление деятельностью, основным результатом которой становится программное средство. Так, согласно [15], процесс разработки

— является руководством, определяющим последовательность выполнения работ (видов деятельности) проектной командой;

— специфицирует, какие артефакты (модели, элементы моделей, документы и т.д.) должны быть разработаны;

— специфицирует, когда должны быть разработаны артефакты;

— соотносит задания отдельных разработчиков и проектной команды в целом;

— предлагает критерии для мониторинга и оценки работ и их результатов.

Моделирование программной архитектуры является составной частью процесса разработки программных средств, а программная архитектура – его артефактом. Выделяют следующие виды деятельности в процессе разработки, которые соотносятся с программной архитектурой [7, 9].

Достижение понимания проблемной области, определение целей, экономических показателей и ограничений на проект. Данный вид деятельности является важным шагом на пути формирования требований, влияющих на программную архитектуру программных средств.

Достижение понимания требований к программному средству. Как правило, на основе созданной модели проблемной области, целей, экономических показателей и ограничений на проект осуществляется формирование требований, где под требованием подразумевается способность выполнять определенную функцию (функциональное требование) или быть конформной определенным условиям [16]. Полученные результаты являются основополагающей информацией, которая поступает на вход процесса моделирования программной архитектуры.

Создание или выбор программной архитектуры. Не существует формальных, предусматривающих строго предопределенную последовательность действий, методов создания или выбора программных архитектур. Результат зависит от многих факторов, включая цели организации, требования к программному средству и опыт архитектора. Тем не менее, существуют механизмы, поддерживающие данный вид деятельности, как, например, методы анализа программных архитектур, архитектурные стили и образцы проектирования.

Представление (описание) программной архитектуры. Программная архитектура выступает как средство, обеспечивающее понимание программного средства, коммуникации между заинтересованными лицами. С другой стороны, воплощая проектные решения, она является артефактом, являющимся основой для реализации и обеспечения последующего эволюционного развития программного средства. Поэтому описание программной архитектуры должно быть информативным, недвусмысленным, читабель-

ным и в максимальной степени формализованным. Последнее обусловлено тем, что программная архитектура является исходной информацией для формализованных методов анализа, включая симуляцию, верификацию и прототипирование. Для описания программной архитектуры применяются языки визуального моделирования, например, язык UML (Unified Modeling Language) [17, 18], и формальные языки описания программных архитектур (Architecture Description Language (ADL)) [2, 7, 19, 20].

Анализ (оценка) программной архитектуры. Данный вид деятельности предусматривает применение различных методов анализа программной архитектуры. Например, на основе Метода Анализа Программных Архитектур (Software Architecture Analysis Method (SAAM)), предложенного в [7], могут быть оценены такие характеристики качества программного средства, как переносимость (portability), повторная используемость (reusability), адаптируемость (adaptability), расширяемость (extensibility). С помощью этого метода может быть выполнен анализ программной архитектуры применительно к ее эволюционному развитию, что, в первую очередь, предусматривает прогнозирование, оценку и локализацию возможных изменений в архитектуре. В отличие от SAAM, ориентированного на анализ качественных атрибутов, формальные языки описания программных архитектур позволяют осуществить анализ свойств, различимых (наблюдаемых) во время исполнения (runtime) программного средства [20].

Реализация на базе разработанной архитектуры программного средства и обеспечение гарантий соответствия. Этот вид деятельности направлен на достижение соответствия реализации программного средства по отношению к разработанной программной архитектуре.

Существует ряд процессов разработки, где моделирование программной архитектуры является одним из доминирующих видов деятельности, являясь составной частью ядра используемой методологии. Примером таких процессов могут быть Унифицированный Процесс от фирмы Rational (Rational Unified Process) [15] и Архитектурный Метод Проектирования (Architecture Based Design Method) [13].

На диаграмме явным образом выделены виды деятельности процесса моделирования программной архитектуры и возникающие между ними связи. В качестве средства описания был использован язык UML, а именно, диаграмма вариантов использования (Use Case Diagram). Заметим, что на диаграмме не показаны причинно-следственные отношения между видами деятельности, поскольку первые могут существенно отличаться в зависимости от рассматриваемых процессов разработки и проектов создания программного средства.

Как видно из диаграммы, на нее также были вынесены виды деятельности: “Реинжениринг программной архитектуры” (включает обратное проектирование программной архитектуры) и “Повторное использование программной архитектуры”. Первый из них, применительно к обратному проектированию, осуществляется в случае, если при создании программного средства имеется унаследованная система, программная архитектура которой не описана. Второй включает деятельность по разработке и описанию повторно используемых архитектурных решений. Оба вида деятельности расширяют вид деятельности “Создание или выбор программной архитектуры”.

**7. Связанные направления**. Существует ряд близких по отношению к программной архитектуре направлений (концепций), затрагивающих вопросы структурной организации программных средств и поддерживающих моделирование программной архитектуры [2, 7, 19, 20, 21]. Такими направлениями являются:

— формальные методы (formal methods) моделирования программной архитектуры;
— образцы проектирования (design patterns) программных средств;
— архитектурные стили (architectural styles);
— эталонные модели и архитектуры (reference models and reference architectures);
— языки описания архитектур (architecture description languages) программных средств;
— языки визуального моделирования (visual modeling languages) программных средств;
— языки программирования (programming languages);
— языки определения интерфейсов (interface definition languages);
— языки описания межмодульного взаимодействия (module interconnection languages).

В последующих разделах приводится краткое описание направлений, которые в наибольшей степени соотносятся с процессом моделирования программной архитектуры.

**7.1. Формальные методы моделирования**. Формальные методы, как правило, не применяются самостоятельно при моделировании программной архитектуры, а выступают формальными базисами в рамках других направлений, например, языков описания архитектур. Так, при описании и анализе программной архитектуры используются следующие формальные методы:

— Абстрактная Химическая Машина (Chemical Abstract Machine) [30];
— Z-нотация (Z-notation) [31, 33];

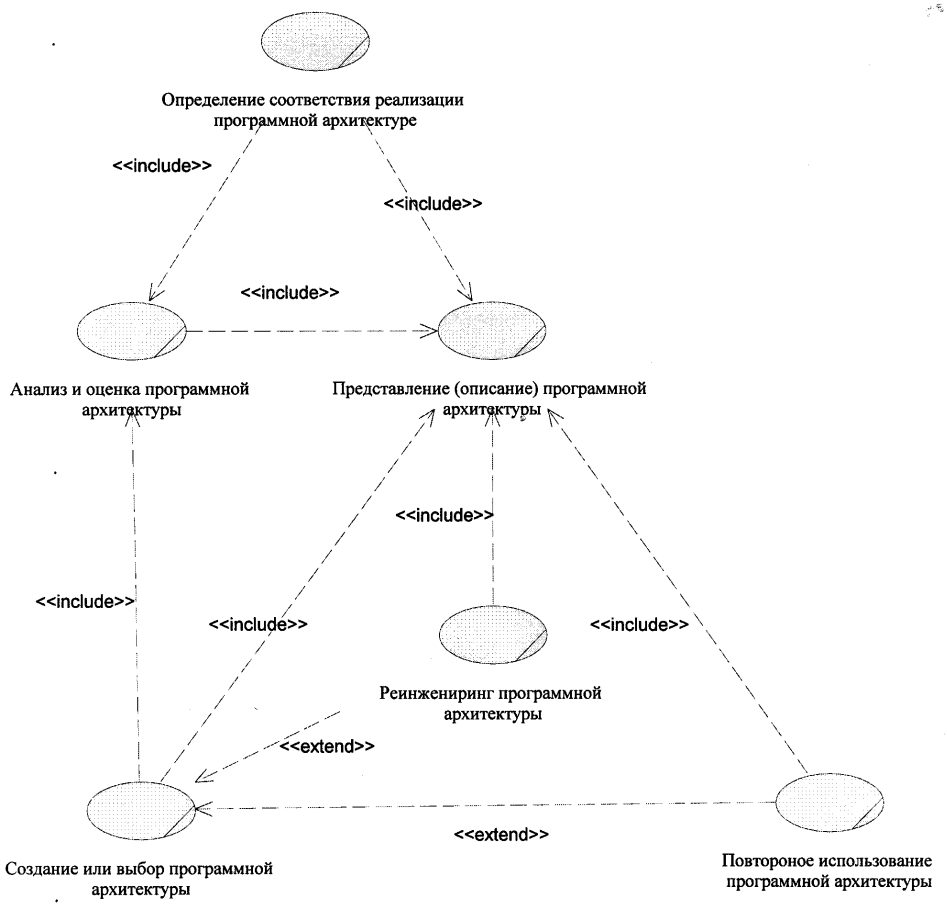


Диаграмма. Процесс моделирования программной архитектуры

— логика первого порядка (first-order logic) [32];
— Теория Графов (Graph Theory) [34, 35];
— Машина Состояний (State Machine) [20];
— Сети Петри (Petri Nets) [20, 35];
— язык CSP (Communicating Sequential Processes) [20].

**7.2. Образцы проектирования программных средств**. Одной из задач, встающих перед разработчиками программных средств, является проблема приобретения опыта и передачи знаний. Направление образцов проектирования (Design Patterns) реализует подход, предусматривающий организацию передачи между разработчиками программных средств экспертных знаний, выраженных в специальном представлении — образцах проектирования. Каждый из них описывает решение какой-либо проблемы, возникающей при проектировании программных средств. В общем случае, образец имеет следующие четыре основных элемента [21]:

— название образца проектирования;

— описание проблем, возникновение которых делает целесообразным использование образца проектирования;

— описание решения, включающее описание элементов, связей между ними, соотносимых с элементами обязанностей[6] (responsibilities), схем взаимодействия элементов; поскольку образец, по сути, является шаблоном (template), решение не включает описание деталей проектирования и реализации;

[6] Обязанность (responsibility) — контракт или обязательство, принимаемое на себя типом или классом [18].

— описание результатов применения образца проектирования.

Образцы проектирования реализуют достаточно универсальный механизм передачи знаний, в частности, при повторном использовании решений. Поэтому они используются и для описания решений проблем, возникающих при моделировании программной архитектуры.

**7.3. Архитектурные стили, эталонные модели и архитектуры**. Создание эталонных моделей является близким к образцам проектирования направлением. Эталонная модель — стандартная декомпозиция известной проблемы на части, в совокупности решающие эту проблему [7]. Данная декомпозиция, как правило, предусматривает выделение функциональных блоков, определение потоков данных между ними. Обобщая опыт, эталонные модели соотносятся с некоторой проблемной областью и часто являются результатом целенаправленного анализа этой проблемной области или какой-либо другой групповой работы, например, процесса стандартизации.

В отличие от эталонной модели архитектурный стиль есть:

— множество типов компонентов (например, хранилище данных, процесс, процедура);

— топология, определяющая взаимосвязи времени исполнения (runtime) между компонентами этих типов;

— множество семантических ограничений (например, информационное хранилище не позволяет изменять хранящиеся в нем данные);

— множество "соединителей" (connectors) — абстракций, которые обеспечивают коммуникацию и координацию при взаимодействии компонентов. Примерами "соединителей" могут быть remote procedure call (RPC) (вызов удаленных процедур), data stream (поток данных), sockets (сокеты).

Архитектурные стили, как и образцы проектирования, и эталонные модели, направлены на решение задач приобретения опыта и передачи знаний. Архитектурный стиль определяет правила организации семейства систем в терминах образцов структурной организации [2]. Имеется большое количество описанных архитектурных стилей [36], примерами которых могут быть: "client/server" (клиент/сервер), "pipe/filter" (канал/фильтр) и "data-centered" (централизованные данные).

Эталонные архитектуры можно рассматривать как отображение эталонных моделей на компоненты программных средств и потоки данных между этими компонентами. В то время как эталонные модели воплощают стандартную декомпозицию функциональности, эталонные архитектуры определяют отображение функциональности на компоненты. Одной из возможных связей между эталонными моделями, архитектурными стилями, эталонными и программными архитектурами может быть следующая [7]: результатом применения некоторого архитектурного стиля по отношению к эталонной модели является эталонная архитектура, которая может быть конкретизирована до уровня программной архитектуры.

Иногда программная архитектура, а чаще всего концептуальная структура, обобщается до уровня, когда ее область применимости уже не ограничивается конкретным программным средством, а соотносится с некоторым классом схожих по своей проблематике областей. Поэтому некоторые исследователи, например в [12], выделяют так называемый класс каркасных моделей (Framework models), которые по своей природе схожи с эталонными моделями и эталонными архитектурами. Такие обобщенные структуры, являясь представителями данного класса, могут быть просто взяты за основу при создании новых программных средств. Примерами каркасных моделей являются предметно-специфичные программные архитектуры (Domain Specific Software Architectures), Общая Архитектура Брокера Объектных Запросов (CORBA), Enterprise Java Beans (EJB) [27].

**7.4. Языки описания архитектур программных средств**. Языки описания архитектур (Architecture Description Languages (ADLs)) [2, 7, 19, 20] образуют одно из наиболее развитых направлений формализованного моделирования программных архитектур. Примерами языков являются: Wright, UniCon, SADL, Rapide, ACME, ArTek, AESOP, Modechart. Согласно [19], язык описания архитектур — это язык, предоставляющий средства для моделирования концептуальной программной архитектуры. Основными элементами таких языков, как правило, являются:

— "компоненты", для которых могут быть определены поддерживаемые ими интерфейсы (или в терминах языка ACME [22] "порты");

— "соединители" (connectors), реализующие протоколы взаимодействия "компонентов" и также поддерживающие определенные интерфейсы (или в терминах языка ACME [22] "роли");

— архитектурные конфигурации, которые являются композицией "компонентов" и "соединителей" и могут быть представлены в виде иерархической структуры;

— "ограничения" на композицию "компонентов" и "соединителей" в рамках архитектурных конфигураций.

Большая часть языков описания архитектур предусматривает типизацию компонентов и соедини-

телей[7], позволяет описывать их "семантику" (динамику поведения). Ряд языков, например, Rapide и Wright [20], имеют средства описания и повторного использования архитектурных стилей. Такие языки, как ACME, AESOP и UniCon [19], делают возможным описание на уровне языка нефункциональных требований.

Практически каждый из языков описания архитектур имеет как текстуальный, так и графический синтаксис представления элементов. В случае текстуального представления для всех элементов языка определен формальный синтаксис и семантика. В рамках приведенного ниже примера дается простейшее текстовое формализованное описание системы клиент/сервер на языке ACME [22]. Текстовое формализованное описание системы $\text{simple}_{\text{c}}\text{s}$ включает:

— определение компонентов client и server;

— спецификацию для каждой из компонентов порта, определяющего интерфейс компоненты;

— определение соединителя, реализующего взаимодействие компонентов через удаленные вызовы процедур (Remote Procedure Call (RPC));

— спецификацию в рамках соединителя ролей caller и receiver;

— определение связей между портами и ролями.

```
System simple_cs = {
        Component client = {Port send-request}
        Component server = {Port receive-request}
        Connector rpc = {Roles {caller, receiver}}
        Attachments: {
                client.send-request to rpc.caller;
                server.receive-request to rpc.receiver}
}
```

Пример. Текстовое формализованное описание системы клиент/сервер

На рисунке представлено для этой же системы графическое представление [22].

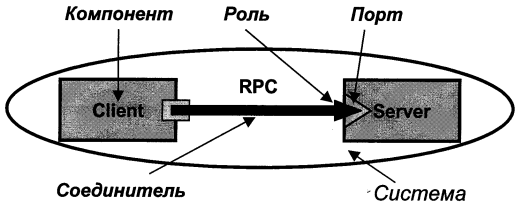


Рис. Графическое представление системы клиент/сервер

Языки описания архитектур предоставляют средства анализа свойств моделируемого программного средства, позволяют описывать и анализировать на архитектурном уровне различные виды межкомпонентного взаимодействия, включая потоки данных и потоки управления. Например, язык Wright [20] предоставляет практические средства описания и анализа образцов дискретного асинхронного взаимодействия на уровне программных архитектур и архитектурных стилей. Как было сказано ранее, анализ свойств программных средств требует формализованного описания программной архитектуры, в том числе — семантики компонентов и соединителей. Для этой цели в языках описания архитектур используются [20]: State Machine, CSP, Z-нотация, сети Петри, логика первого порядка, теория графов и другие формальные методы.

**7.5. Языки визуального моделирования программных средств**. Языки визуального моделирования, используемые для описания и анализа программной архитектуры, можно разделить на три категории:

— языки, являющиеся визуальным представлением формализованного текстуального описания программной архитектуры, например, на языке описания архитектур [22].

[7]Некоторые языки описания архитектур, например Wright, поддерживают параметризацию типов компонентов и соединителей.

— языки, неформально описывающие программную архитектуру, например, с помощью "box and line" диаграмм [20] и используемые без каких-либо поддерживающих их формальных методов.

— общецелевые языки моделирования, т.е. допускающие в различной степени моделирование как программных архитектур, так и других моделей, возникающих в процессе разработки программных средств.

Примером языка третьей категории является UML (Unified Modeling Language), который определяется в [18] не только как язык визуализации, но и как язык спецификации, конструирования и документирования. В основу методологии моделирования, поддерживаемой этим языком, положен подход [11], описанный в данной статье (см. раздел "Совместное использование структур и их применимость") и обеспечивающий совместное согласованное использование различных структур программного средства. Язык UML, с одной стороны, имеет строго определенную семантику, описанную на базе четырехуровневой метамодели языка, а с другой стороны, является расширяемым, предоставляя средства введения новых элементов языка [17, 18]. Имеются отдельные примеры интеграции языка UML с другими средствами и методами моделирования программных архитектур [24, 25]. В настоящий момент он стандартизируется OMG.

**8. Заключение**. Данная статья носит скорее обзорный, чем исследовательский характер. Тем не менее, можно выделить следующие области в научно-практической дисциплине "Моделирование программных архитектур", которые требуют исследования и решения возникающих в них проблем.

Создание или выбор программной архитектуры. Требуется разработка методов проектирования программной архитектуры, введения новых структур, позволяющих моделировать на архитектурном уровне структурные аспекты, которые становятся жизненно важными в связи с развитием информационных технологий, изменением роли, отводимой им в организациях, использующих или создающих программные средства. Примером таких структурных аспектов может служить транзакционная обработка информации, которая практически не отражена в рамках существующих структур, определяющих архитектуру программных средств.

Представление (описание) программной архитектуры. Актуальной является разработка, с одной стороны, методов формализованного описания программной архитектуры для целей ее последующего анализа, а с другой стороны, методов визуального представления, где языки описания были бы в максимальной степени близки к естественному языку, но в то же время имели бы формальный синтаксис и семантику.

Анализ и оценка программной архитектуры. Требуется разработка методов анализа свойств программного средства. Как и в случае создания или выбора программной архитектуры, необходимость в анализе тех или иных свойств обусловлена направлениями развития информационных технологий, изменением роли, отводимой им в организациях, использующих или создающих программные средства. Особенно актуальными являются вопросы эволюции программной архитектуры.

Реинжениринг программной архитектуры. Чрезвычайно важной становится разработка методов обратного проектирования, методов прямого проектирования в условиях присутствия унаследованных систем, поскольку последние являются атрибутом практически каждого проекта информатизации.

Определение соответствия реализации программного средства программной архитектуре. В условиях появления новых технологий реализации программных средств требуется создание новых и развитие существующих методов [13–15, 37] определения соответствия реализации программным архитектурам.

Процесс моделирования программной архитектуры. Многие существующие методы моделирования программной архитектуры, например, используемые в языках описания архитектур, не интегрированы с использующимися на практике процессами разработки программных средств, что делает их недоступными для практического использования сообществом разработчиков. В настоящее время такая интеграция, безусловно, является актуальной.

СПИСОК ЛИТЕРАТУРЫ

1. *Spencer D.* Webster's new world dictionary of computer terms. New York: Prentice Hall, 1993.
2. *Shaw M., Garlan D.* Software architecture. London: Prentice-Hall, 1996.
3. *Perry D.E., Wolf A.L.* Foundations for the study of software architecture // ACM SIGSOFT Software Engineering Notes. 1992. **17**, N 4. 40–52.
4. *W*hat is software architecture? URL: http://www.sei.cmu.edu/architecture/definitions.html, Carnegie–Mellon University.
5. *Garlan D., Shaw M.* An introduction to software architecture // Advances in Software Engineering and Knowledge Engineering. Vol. 2, pp. 1–39. Singapore: World Scientific Publishing Company, 1993.

6. *Философский словарь*/ Под ред. И.Т. Фролова. М.: Политиздат, 1980.
7. *Bass L., Clements P., Kazman R.* Software architecture in practice. Reading: Addison Wesley, 1998.
8. *Morris C.R., Ferguson C.H.* How architecture wins technology wars // Harvard Business Review. 1993. 86–96.
9. *Clements P.C., Northrop L.M.* Software architecture: an executive overview. Technical Report CMU/SEI-96-TR-003, ESC-TR-96-003. Pittsburgh, 1996.
10. *Budgen D.* Software design. Reading: Addison-Wesley, 1994.
11. *Kruchten P.B.* The 4+1 view model of architecture // IEEE Software. 1995. **12**, N 6. 42–50.
12. *Shaw M., Garlan D.* Formulations and formalisms in software architecture // Computer Science Today. 1995. N 1000. 307–323.
13. *Bachmann F., Bass L., Chastek G, Donohoe P., Peruzzi F.* The architecture based design method. Technical Report CMU/SEI-2000-TR-001, ESC-TR-2000-001. Pittsburgh, 2000.
14. *Jacobson I., Booch G., Rumbaugh J.* The unified software development process. Reading: Addison Wesley, 1999.
15. *Kruchten P.* The rational unified process: an introduction. Reading: Addison Wesley, 1999.
16. *Grady R.B.* Practical software metrics for project management and process improvement. London: Prentice Hall, 1992.
17. OMG Unified Modeling Language specification. Version 1.3. June 1999. http://www.omg.org
18. *Booch G., Rumbaugh J., Jacobson I.* The Unified Modeling Language User Guide. Reading: Addison Wesley, 1999.
19. *Medvidovic N.* A classification and comparison framework for software architecture description languages. Technical Report UCI-ICS-97-02. Irvine, 1996.
20. *Allen R.J.* A formal approach to software architecture: Thesis, Pittsburgh, 1997.
21. *Gamma E., Helm R., Johnson R., Vlisssides J.* Design patterns: elements of reusable object-oriented software. Reading: Addison Wesley, 1995.
22. *Garlan D., Monroe R., Wile D.* Acme: an architecture description interchange language // Proceeding of CASCON'97, pp. 169–183. Toronto, 1997.
23. *Vaskevitch D.* Client/server strategies. A survival guide for corporate reengineers. Foster City: IDG Books Worldwide, 1995.
24. *Robbins J.E., Medvidovic N., Redmiles D.F., Rosenblum D.S.* Integrated architecture description languages with a standard design method. University of California, Irvine, 1997.
25. *Egyed A.* Integrated architectural views in UML. Technical Report USC/CSE-99-TR-514. Los Angeles, 1999.
26. *Hoque R.* CORBA 3 developer's guide. Foster City: IDG Books Worldwide, 1998.
27. *Orfali R., Harkey D.* Client/server programming with Java and CORBA. New York: John Wiley & Sons, 1998.
28. *Хомяков Д.М., Хомяков П.М.* Основы системного анализа. М.: Изд-во механико-математического факультета МГУ, 1996.
29. *Orfali R., Harkey D., Edwards J.* The essential distributed objects. New York: Wiley Computer Publishing, 1996.
30. *Wermelinger M.* Specification, testing and analysis of (dynamic) software architecture with the chemical abstract machine. Departamento de Infomatica, Universidade Nova de Lisboa, Portugal, 1998.
31. *Abowd G., Allen R., Gralan D.* Formalizing style to understand descriptions of software architecture. Technical Report CMU-CS-95-111. Pittsburgh, 1995.
32. *Moriconi M., Qian X., Riemenschneider R.* Correct architecture refinement // IEEE Transactions on Software Engineering. 1995. **21**, N 4. 356–372.
33. *Spivey J.M.* The Z-notation: a reference manual. London: Prentice Hall, 1992.
34. *Murphy G.C., Notkin D., Sullivan K.* Software reflexion models: bridging the gap between source and high-level models // Proceedings of the Third ACM SIGSOFT Symposium on the Foundations. pp. 18–28. Washington, 1995.
35. *Брой М.* Информатика. Часть 3. М.: Диалог-МИФИ, 1996.
36. *Gacek C.* Detecting architectural mismatches during systems composition: Thesis. Los Angeles, 1998.
37. *Collofello J.S.* Introduction to software verification and validation. SE Curriculum Module SEI-CM-13-1.1. Pittsburgh, 1988.

Поступила в редакцию
05.04.2001