УДК 004.4

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ: ПРОЯСНЕНИЕ ПРИНЦИПОВ?

Ермаков Илья Евгеньевич,
ООО «Метасистемы»,
Технологический институт ОрёлГТУ, преподаватель,
ООО «Метасистемы», директор по науке и образованию
ermakov@metasystems.ru

1. Немного критики.

Темы, связанные с объектной ориентированностью, прошли через все этапы: от любопытной новинки, затем — массовой моды, и, наконец, — до примелькавшейся банальности. Вместе с тем, это не прибавило вопросу ясности. Такое ощущение, что за игрой слов программные инженеры так и не успели глубоко разобраться в существе дела, сразу перейдя в фазу использования — по спонтанно сформировавшимся канонам, но проскочив фазу научно-технической проработки вопроса.

Итак, взглянем правде в глаза: ни объектно-ориентированные методы, ни объектно-ориентированные инструменты (например, СУБД) так и не достигли тех рубежей, которые были намечены. Поясним, с каких позиций мы берём смелость выносить такие суждения. Если мы понимаем создание ПО как разновидность инженерной деятельности, то основной задачей должна быть систематическая организация дела: возможность создавать конструкции с предсказуемыми свойствами, иметь возможность применять строгие, при необходимости — формализованные, методы и инженерные процессы. Кроме того, инженерная деятельность подразумевает осознание двух основополагающих принципов: «принципа Калашникова» — «Избыточная сложность — всегда уязвимость» и «Римского принципа» — «Разделяй и управляй» (сформулированы проектом «Информатика-21» [2]). Подробный анализ вопросов современной программной инженерии проведён нами в [1].

Если мы посмотрим на подходы ООП в их массовом виде, то увидим полное равнодушие к этим целям и принципам. Вместо того, чтобы способствовать прояснению структуры систем, введению «инженерно осязаемых» конструктивных схем, происходит нагромождение виртуальных, неявных структур, которые совершенно выходят из-под интеллектуального контроля физически ограниченного человеческого мозга. Этой болезнью страдают и массовые языки (от C++ до C#), и объектно-ориентированные проектные средства (UML), и процессы (RUP).

В то же время, в отрасли имеются примеры значительных успехов на пути к полноценному инженерному процессу. Человечество имеет опыт создания очень крупных и в то же время надёжных программных систем, которые работают в самых разных отраслях — и заметим, что большая их часть была создана ещё в дообъектную эпоху. Качество и технологический уровень дообъектных архитектур можно видеть по системам, созданным на базе языков

Ada, Modula-2, Эль-76 и др. [3, 4, 5, 6].

Методы системного анализа, вообще говоря, «безразличны» к ООП (т.е. могут легко в него отображаться, но объектный понятийный аппарат оказывается слишком низкоуровневым для непосредственного использования системными аналитиками), зато вовсю оперируют понятиями потоков [7]. Метод анализа, навязываемый методологиями RUP/UML, скорее является попыткой поднять понятия конкретных языков программирования на уровень системного анализа (обычный программист, конечно, чувствует себя при этом весьма комфортно, если забыть о сильной примитивизации реальности).

Иллюзия выражения в ОО-модели приложения «полной картины проблемы» оказывается устойчивой — она выражается в кочующих по книгам и статьям ничем не обоснованных утверждениях о широком повторном использовании бизнес-объектов, и т.п. На самом же деле как раз в прикладном программировании (в отличие от системного) повторное использование созданных для конкретной системы бизнес-объектов обычно трудно себе представить — ведь любая удачная декомпозиция, абстракция и моделирование проблемы выполняется всегда с точки зрения конкретной постановки задачи.

Наконец, отметим ещё одну наметившуюся проблему: ООП, казалось, значительно понизило роль классического понятия алгоритма (за счёт декомпозиции, разделения ролей, делегирования, принятия решений на основе виртуальных механизмов). Однако от этого фундаментального понятия убежать не удаётся: по мере роста сложности систем оно возникает на новом уровне — в протоколах взаимодействий между объектами, в логике изменения состояний объектов под влиянием внешних событий. Традиционные объектные языки программирования не предоставляют никаких средств для спецификации алгоритмов как грамматик взаимодействий объектов. Первый важный шаг в этом направлении был сделан в языках Zonnon [8] и Composita [9] Университета ETHZ, далее — в ОС Singularity от Microsoft Research [10]. Также интересен визуальный язык ДРАКОН [11, 12], который может стать отличным инструментом для спецификации протоколов объектных взаимодействий.

2. ООП и его смыслы.

Нам представляется, что существующие проблемы с пониманием ООП связаны со смешиванием его прикладного и технического смысла, которое идёт ещё со времён языка Simula-67. Как известно, эта система была создана как проблемно-ориентированное средство для имитационного моделирования. Абстракция объекта, объединяющего состояние и поведение, оказалась достаточно удобной для прикладного программирования.

Однако вопрос удобства (синтаксического сахара) никогда не является принципиальным фактором, качественно что-то меняющим, — максимум, это некоторый коэффициент к производительности труда разработчика. В том же имитационном моделировании более удобным может быть использование пассивных структур данных, с алгоритмами модельного времени над ними (особенно, если из-за объёма задачи требуется специальное управление памятью, подкачкой и т.п.).

Реальную революцию в программировании произвел не «объектный взгляд на мир» и не «выразительные возможности ООП», а конкретные языково-системные механизмы, введённые в Simula-67 и эффективно апробированные в интегрированных операционных средах Smalltalk. Эти механизмы обеспечили возможности для расширения программных систем, их долгосрочной эволюции. Вспомним эти 4 механизма:

1) расширение типа данных (описание в программе отношений «род»—«вид» между типами данных);

2) полиморфизм (параметров процедур и ссылочных переменных) — это способ во время выполнения применить сущность расширенного (видового) типа там, где статически во время написания ожидается сущность базового (родового) типа;

3) сопряжение типов данных и процедур, с возможностью обращения к процедурам как к операциям экземпляра типа;

4) виртуализация вызова процедур (динамическое связывание на основе фактического типа объекта).

Механизмы 1 и 2 являются статическими (способом описать нечто средствами языка программирования), механизмы 3 и 4 — динамическими возможностями, вытекающими соответственно из 1 и 2. Эти динамические возможности являются основой для расширения программных систем во время выполнения.

Многие специалисты выражают мнение [13], что основной идеей ООП и критерием для признания какого-либо языка объектно-ориентированным является возможность расширять программную систему новыми сущностями без переписывания уже существующего кода. В любом случае, фактом является то, что в ряде языков программирования (Oberon-2, Modula-3, Ada-95, Component Pascal) все 4 механизма обеспечены сочетанием модульности и обычной системы структур данных, без введения понятий класса и объекта. В базовом же Обероне и его версии Оберон-07 Н. Вирт обеспечил объектную ориентированность без введения связанных процедур; виртуальные вызовы выполняются посредством процедурных полей объекта. (Причиной продолжения поддержки этого подхода является не отрицание более практичного и надёжного ООП в стиле Компонентного Паскаля, а то, что базовый Оберон, с одной стороны, играет роль эталонного языкового ядра, на основе которого могут строиться промышленные варианты, с другой — имеет в качестве традиционной целевой ниши встроенные системы).

Размытый смысл имеет термин «инкапсуляция» (сомнительна вообще полезность его использования). Одна из его смысловых нагрузок — механизм 4, однако этот смысл смешан с принципом сокрытия внутреннего устройства. Последний является дообъектным принципом, сформулированным Д. Парнасом и воплощённым в модульных языках программирования (Модула-2, Cedar, Ада, Оберон и др.) В модульных языках сокрытие информации возлагается на конструкцию модуля, а не на конструкцию класса. Такой подход целесообразен, особенно в сочетании с динамической модульностью — ведь только

динамическая загрузка модулей позволяет ООП заиграть в полную силу при обеспечении эволюции системы [18].

3. Расширяемое ООП.

Объектно-ориентированные механизмы являются необходимой, но не достаточной основой для построения надёжных расширяемых систем.

Именно поэтому в 90-х гг. была заявлена такая парадигма, как компонентно-ориентированное программирование (КОП), одним из основных идеологов которого является Клеменс Шиперски [14]. В его формулировке КОП = «Полиморфизм + Реальное позднее связывание компонентов + Реальное сокрытие информации в компонентах + Безопасность». В конкретной реализации Оберон-систем это обеспечивается абстракциями расширяемых записей, полноценной модульностью с раздельной компиляцией и динамической загрузкой, герметичной системой типов данных и сборкой мусора.

Значительную роль в развитии настоящей «объектной инженерии» сыграла книга «банды четырёх» [16], посвящённая паттернам объектно-ориентированного проектирования. Паттерн ООП — это, фактически, схема соединения нескольких компонентов (имеющих абстрактные ОО-интерфейсы) и распределения между ними ролей (для заимствованного термина «паттерн» нами предложен [1] вариант замены: «конструктивная схема»). Дальнейшее развитие идей авторов книги паттернов получило в проекте EthOS, а затем — в системе программирования BlackBox Component Builder и компонентном каркасе BlackBox Component Framework. Ход мысли, лёгший в основу этого направления, подробно раскрыт в диссертации К. Шиперски [15]. В ней вводится понятие **расширяемого ООП** (extensible OOP) и подробно разбираются все необходимые для этого ограничения, накладываемые на «простое ООП». За подробностями мы отсылаем читателя к этой работе, а также к публикациям на OberonCore [17, 18, 19, 20, 21].

Здесь коротко обрисуем основные особенности, характерные для расширяемого ООП — и вообще для систематической инженерии на основе ООП.

Во-первых, имеются отличия в самой «философии»: если популярная точка зрения заключается в том, что «программа — это театр, а объекты — это актёры», то программный инженер обязан помнить о разграничении между самой системой, им создаваемой, и теми данными, которые она обрабатывает. В частности, объект как компонент архитектуры, имеющий соединения с другими компонентами, и объект как часть модели проблемной области и элемент хранилища данных — это принципиально разные роли ООП, которые не должны смешиваться. Прикладной объект не может «сам себя оживлять», должен существовать уровень архитектурных объектов, которые выполняют манипуляции над прикладными, передавая их друг другу через свои интерфейсы. Этому же способствует наличие отдельной конструкции модуля. Таким образом, если и сравнивать хорошую объектную программу с театром, то — с кукольным (отдельно — модель задачи, отдельно — оживляющие её

модули и архитектурные объекты, которые могут рассматриваться как «мультимодули»).

Во-вторых, настойчиво рекомендуется отказ от наследования реализации. В работах К. Шиперски и К. Пфистера [14, 15, 17] описана проблема хрупкого базового класса (fragile base class problem), которая заключается в том, что неабстрактный класс, от которого имеются расширения у сторонних разработчиков, практически невозможно изменить. Наследование реализации ведёт к запутанным и неявным схемам соединения компонентов и передачи управления. Его можно сравнивать с оператором GOTO и считать средством, сильно ухудшающим структуру программной системы. Полный запрет на наследование реализации характерен также для языковонезависимых компонентных моделей 90-х гг., подобных COM – хотя там этот отказ сделан вынужденно (в силу трудностей с поддержкой такого наследования), но можно видеть, что эффект от этого наблюдался только положительный.

Таким неявным механизмам, как наследование реализации, стоит предпочесть обычную композицию абстрактных компонентов. Компоненты делаются как можно более простыми, обобщаются до абстрактных интерфейсов, через которые устанавливается их соединение друг с другом. Наследование является плоским (так называемые гомогенные иерархии) — реализация расширяется от базового абстрактного интерфейса и является скрытой, создание экземпляров происходит посредством фабрик. Композиция объектов — это явный, прозрачный, «инженерно осязаемый» способ организации системы, который хорошо проектируется, документируется, анализируется, поддаётся расширению. С точки зрения этапа проектирования он соответствует известному проектному плану «компоненты—соединители».

Заметим, что при таком подходе структура системы сближается, по существу, с той, которая характерна для программ на функциональных языках и для dataflow-архитектур: имеется ряд долгоживущих объектов, которые передают между собой потоки обрабатываемых полиморфных данных. Многие поклонники функционального программирования резко критикуют ООП за недостаточную гибкость, «хрупкость», на самом же деле эти недостатки связанны именно с наследованием реализации и преодолены в подходе компонентно-ориентированного программирования.

Необходимо помнить про **принципиальное влияние сборки мусора на архитектуру, на структуры данных и на алгоритмы**. Это влияние до сих пор недооценивается, в силу консерватизма мышления разработчиков, даже при использовании таких языков, как Java и C#, где имеется сборка мусора. Из императивных систем полностью задействуют преимущества сборки мусора, пожалуй, только Оберон-архитектуры. С другой стороны, поклонники функциональных и интерпретируемых языков часто даже не осознают, что основной фактор их эффективности — это именно автоматическое управление памятью и возможность сложных ссылочных структур данных, всё остальное — второстепенно и дискуссионно.

Основные архитектурные приёмы Оберон-систем рассмотрены в публикации [20] — абстрактные разъёмы, инсталлируемые фабрики (directories)

динамическое переключение реализаций, подключение платформенно-зависимых модулей без перекомпиляции.

Особое место занимает эффективный приём передачи полиморфного сообщения через VAR-параметр [20]. Он возможен благодаря тому, что расширение типов данных в Обероне допускается не только для указательных типов, а для любых записей. Этот механизм можно рассматривать как безопасный вариант процедур со свободными параметрами, при этом он не требует сколь-нибудь значительных накладных расходов. Примером использования этого механизма при организации структур данных и алгоритмов является реализация на Компонентном Паскале транслятора функционального (марковского) языка Рефал-0, описанная в работе [21].

4. Выводы.

Объектно-ориентированное программирование является основным методом и средством создания программных систем, пригодных для расширения и длительной эволюции. Оценка прикладной роли ООП чрезвычайно завышена, в то время как осознание его возможностей для архитектуры систем и системного программирования, наоборот, в полной мере ещё не пришло. Это осознание должно быть связано с ясным пониманием главных механизмов ООП и способов их обеспечения, с наработкой систематических инженерных подходов к организации объектных систем.

Литература

1. Ермаков, И.Е. От ремесла — к инженерии, от мастерской — к заводу программных систем. // OberonCore. Web: http://metasystems.ru/download/science/a-042-sudak-lection-2010-eie.pdf; обсуждение — http://forum.oberoncore.ru/viewtopic.php?f=86&t=2574

2. Принцип Калашникова. // Информатика-21. Web: http://www.inr.ac.ru/~info21/princypKalashnikova.htm

3. Бар, Р. Язык Ада в проектировании систем. / Р. Бар. — Пер. с англ. — М.: Мир. — 1988. — 320 с. Web: http://317.metasystems.ru/books/System-Design-With-Ada.djvu

4. Янг, С. Алгоритмические языки реального времени. / С. Янг. — Пер. с англ. — М.: Мир. — 1986. — 400 с. Web: http://317.metasystems.ru/lib/exe/fetch.php/knowledge:young.pdf

5. Пентковский, В.М. Язык программирования Эль-76. Принципы построения языка и руководство к пользованию. — 2-е изд. испр. и доп. / В.М. Пентковский. — М.: Наука. — 1989. — 367 с. Web: http://store.oberoncore.ru/lib/book/pvm1989r.djvu

6. Сафонов, В.О. Языки и методы программирования в системе «Эльбрус» / В.О. Сафонов. — Под ред. С.С. Лаврова. — М.: Наука. — 1989. — 392 с. Web: http://store.oberoncore.ru/lib/book/svo1989r.djvu

7. Маторин, С.И. Анализ и моделирование бизнес-систем: системологическая объектно-ориентированная технология. / С.И. Маторин. — Харьков: ХНУРЭ. — 2002. — 322 с.

8. Zonnon Programming Language. Web: http://www.zonnon.ethz.ch/

9. Composita Programming Language. Web: http://www.jg.inf.ethz.ch/wiki/ComponentLanguage/Front. Обсуждение: http://forum.oberoncore.ru/viewtopic.php?f=21&t=459

10. OS Singularity RDK. Web: http://singularity.codeplex.com/

11. Паронджанов, В.Д. Как улучшить работу ума. Алгоритмы без программистов — это очень просто! / В.Д. Паронджанов. — М.: Дело. — 2001. — 360 с.

12. Визуальный язык «Дракон». // OberonCore. Web: http://oberoncore.ru/wiki/drakon/start
13. Рыбин, С.И. Современное ИТ: индустрия и образование. Обзор языка Ada. // OberonCore. — 2006. Web: http://store.oberoncore.ru/lib/paper/rybin.pdf
14. Szyperski, C. Component Software. Beyond Object-Oriented Programming. — Addison Wesley Longman. — 1998.
15. Szyperski, C. Insight EthOS — On Object Orientation In Operating Systems. — ETH Diss. No 9884. — Zurich, Switzerland. — 1992. — ISBN 3-7281-1948-2. Web: http://research.microsoft.com/en-us/um/people/cszypers/books/insight-ethos.htm
16. Гамма, Э. Приёмы объектно-ориентированного проектирования. Паттерны проектирования. / Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес. — СПб.: Питер. — 2001. — 368 с.
17. Пфистер, К. Компонентное ПО. — Пер. с англ. И.Е. Ермакова. // OberonCore — 2005. — Web: http://oberoncore.ru/blackbox/articles
18. Губанов, С.Ю. Секреты модульных систем. // OberonCore. — 2006. Web: http://oberoncore.ru/programming/articles
19. Ермаков, И.Е. Оберон-технологии: что это такое? // OberonCore. — 2006. Web: http://oberoncore.ru/programming/oberon-technology
20. Ермаков, И.Е. Некоторые идеи архитектуры Оберон-систем. // OberonCore. — 2007. Web: http://oberoncore.ru/programming/oberon-technology
21. Ермаков, И.Е. Встраиваемый язык обработки текстов Рефал-0 и разработка его транслятора на Компонентном Паскале. // ООО «Метасистемы». — 2008. Web: http://metasystems.ru/download/science/Cot-r-001-refal0-2008-eie.pdf