Объектная парадигма провалилась

Открытое письмо Ричарда П. Гэбриэла от 6 Ноября 2002г.
Ричард П. Гэбриэл, Пятница, 16 Июль 2004, 21:49

Что это такое вообще — провал парадигмы программировани? Парадигма терпит неудачу, когда содержащийся в ней посыл утрачивает адекватность, или когда приверженцы парадигмы продолжают держаться её вопреки здравому смыслу. Причиной утраты адекватности являются сильно изменившиеся требования, предъявляемые к программному обеспечению в XXI веке, и так называемые усовершенствоваия в ОО, перечеркнувшие его первоначальные преимущества. Одержимость ООП переросла в продвижение частного решения в качестве панацеи от всех проблем, связанных с программным обеспечением, превратившись в идеологическое оружие, целью которого была промывка мозгов людям на улицах. Утверждение "всё суть " подразумевает универсальностьОО, а утверждение "объекты моделируют реальный " говорит о привилегированнм положении ОО. Весьма заманчивое приглашение к единообразной точке зрения, приводящее к замораживанию исследований и разработок альтернативных парадигм.

Когда-нибудь все программное обеспечение, которые мы уже написали, будет иметь нулевую ценность. Мы пережили три компьютерные эры: первая - кодирование в машинных кодах; вторая - ассемблеры, интерпретирующие программы и ранние компиляторы; третья - эра императивного, процедурного и функциональногопрограммировани, языков, ориентированныхна компиляторы. И вот, наступила четвёртая эра — объектно-ориентированног программировани. Все четыре эры были ориентированы на программы, работающие на одном компьютере. Хотя такие системы и сейчас представляют ценность, всё возрастающая сложность систем приведёт к тому, что они будут состоять из сотен тысяч или даже из миллионов различных компонент, приложений, служб, сенсоров и приводов, работающих на различном оборудовании, написанных разными производителями, так что не было бы ошибкой сказать — никто не знает, как это всё работает. В старом мире мы фокусировали своё внимание на эффективности, ограничении используемых ресурсов, производительноти, монолитных программах, отдельных системах, программах "одного " и математических моделях. В новом мире на первый план выходят устойчивость, гибкость, адаптируемость, распределённые системы, программы коллективных разработчиков и биологические (взятые из жизни) метафоры в вычислительной технике.

Конечно, ОО-подход представляет собой мощный инструмент, используя который, можно понять и обустроить системы в новом мире, но он не может быть единственным инструментом. В будущих системах ненадёжность (многих компонентов в целом) станет обычной, сложность выйдет за пределы поля зрения, и строить системы на тщательно "" коде станет нереально. Это похоже на город: кирпичи важны для постройки зданий, но вся разношёрстная гамма применяемых строительных материалов и компонент, и людей, которые ими занимаются — разных культур, мировоззрений, возможностей и талантов, приводит нас к мысли, что тип мышления одними только "" не будет работать в масштабах города. Кирпичи слишком ограничены, и обстоятельства, в которых они могут применяться, слишком ограничены, чтобы использоваться в качестве модели для построение чего-того такого же разнообразного и непредсказуемог, как город. Более того, город сам по себе тоже не является конечной целью, поскольку он, в идеальном случае, суть "гуманитарное образование для человеческой ", понимание этого требует введения дополнительногоуровня сложностей и

связей. Применяя эту метафору для разговора о компьютерных системах будущего, будет справедливо заметить, что ООП сегодня находится на "уровне ".

Современные тенденции в ИТ принуждают к единообразному подходу, в рамках которого предполагается, что одна парадигма устроит всех во всех ситуациях. Но, даже стараясь изо всех сил, приверженцы ОО не могут предоставить убедительное итоговое решение, подходящее для всех без исключения. У парадигмы ОО на самом деле нет привилегированнй позиции. Поэтому разработчики Java, к примеру, вместо того, чтобы использовать уже существующие компоненты, решают включить в своё решение всё, что "может понадобиться", чтобы весь компьютерный мир оказался охваченным Java.

Объекты, введённые дизайнерами таких языков, как Smalltalk и Actors - задолго до C++ и Java - были созданы для моделирования сложных, динамичных миров. Среда программировани в Smalltalk была создана с использованием его же самого, и является расширяемой для программистов-разработчиков. Поскольку философия динамических изменений была частью "мировоззрения" языков, созданных после Simula, языки и среды того времени были весьма динамичными.
Примечание переводчика: для тех кто не знаком со Smalltalk, будет откровением узнать, что при разработке программ в его среде, нет понятия этапов "дизайн-компиляция-" — изначально задуманный как интерпретирующий язык, со временем переросший в компилирующий во время исполнения (задолго до Java!), в конце концов стал "компилирующий и сохраняющий результаты компиляции до изменения программного ", с оптимизацией. Трудности оптимизации в этом языке связаны именно с его динамичностью — объекты Smalltalk имеют ограниченный набор методов (в привычном понимании адептов C++), и всё что они "умею" — посылать сообщения друг другу, соответственно всё программировани в Smalltalk сводится к "обучению" объектов реагировать на соответствующие сообщения. Причём новая реакция на новое сообщение может быть добавлена "на " во время исполнения программы (читай "добавление к объекту нового "). Однако были достигнуты приемлемые результаты оптимизации. При выпуске программы "в " из неё просто исключался "за ненужностью" модуль среды Smalltalk, поскольку программа отлажена и работает.

Но с приходом C++ и Java, динамичное мышление, воспитанное в духе ранних объектно-ориентированныхязыков, было практически полностью подавлено теологией статичного мышления, взятого из математическогонаследия, и взглядами на вычисления, восходящими к Чарльзу Бэббиджу, с мировоззрением, ориентированнымна всеведение и всемогущество.

В результате мы пришли к тому, что объектно-ориентированныеязыки поддались влиянию статично мыслящих разработчиков, которые поставили тщательное планирование выше приспособляемости программ во время исполнения, ранние решения выше поздних, и "" компилятора выше умения обработки ошибок и восстановления.

Вместо статичных типов, точных интерфейсов и математических обоснований, появилась острая необходимость в самоорганизующихся механизмах, способных к самопроверке и различной реакции на ошибки, в управляемых системах, общая сложность которых будет лежать за пределами понимания одной личности.

Кто-нибудь может подумать, что подобный постмодернистки шаг приведет к положительным последствиям, но, в отличие от Perl, подобная комбинация привела не к сложению, а к вычитанию - как будто бы ОО могло стать лучше при вырезании из него самой сути. Это могло бы сработать в книжных примерах, но идея программировани - не учить, а строить.

Частный коммерческий успех объектов и наш развивающийся в последнее десятилетие роман с большим бизнесом вылились в остановку изучения и разработки альтернативных языковых подходов и парадигм в компьютерной науке. Научные и промышленные сообщества отступили от исследований в сфере языков, пожиная лёгкие плоды с дерева ОО. Безумие большого бизнеса конца прошлого века ослепило исследователей и привело к тому, что на их взгляд, то, что существует в сфере ООП сегодня, настолько близко к идеалу, что это бесспорно. И если когда-нибудь и было время, когда Куновская "нормальная " доминировала в вычислительной технике, так именно в этот период (Томас Кун в своей книге "Структура научных революю" показал, что принятие единой парадигмы, в конце концов, обязательно приведёт к новой революции).

Я на собственном опыте перенёс это. До того, как я ушел изучать поэзию в 1995 году, моя карьера исследователя вращалась вокруг языка программировани LISP. Когда я вернулся в 1998, я обнаружил, что моя область исследований оказалась уничтоженной. Я был вынужден искать новые способы зарабатывания на жизнь в рамках новой экологии компьютерного мира, созданного языком Java, который деловито перестраивал мир программировани по своему образу и подобию.

Smalltalk, Lisp, Haskell, ML и другие языки зачахли, в то время когда C++, Java и ее почти что клон C# стали единственными языками, достойными внимания. Малые языки вроде Tcl, Perl и Python ещё собирают своих приверженцев, но не демонстрируют значительного прогресса как в языке, так и в системном дизайне.

сложившийся объектно-ориентированныйпоход неадекватен требованиям вычислительных процессов будущего;
объектно-ориентированныеязыки потеряли простоту, а можно сказать и чистоту, которая особенно их отличала в плане выразительной силы и творческой мощи;
мощные концепции, такие как инкапсуляция, были придуманы для уменьшения ошибок при разработке программ, но инкапсуляция... обременяет, когда меняются глобальные особенности, или когда необходима эволюция программы или, вообще, её коренная перестройка. Концепция открытого исходного кода лучше справляется с подобной ситуацией. Она похожа на модульность — разбиение на составные части так, чтобы люди могли их понять — вот что действительно важно в инкапсулировани;
объекты обещали повторное использование, однако мы не наблюдаем больших достижений в этой области;
несмотря на ясное понимание пионерами ОО природы разработки программ, нынешнее поколение их преемников более озабочено философией тщательного планирования, монументальногодизайна, идеей охватить всё, "", взятым из теологии Бэббиджа;
эйфория, порождённая объектами в ранних 1990, вела к ожиданию чудес, которые могли бы стать возможными при незагрязнении статичным мышлением, когда же нынешние разработчики ПО оказываются не в состоянии добиться желаемого, рушатся и

возмутительные планы этих бизнесов, результатом чего является нынешняя рецессия;
объекты требуют программировани методом создания общающихся сущностей, это означает, что программировани осуществляется путем построения структур, а не языковых выражений и описаний с помощью форм, что часто приводит к несоответствию языка предметной области;
объектный дизайн, в котором создаётся мир, где объекты "общаются" и взаимодействуют между собой, ведёт людей к выводу, что обучение объектно-ориентированном программировани легко, а фактически это стало трудно, как никогда.
Те, кто пришли в восторг от ООП, заслонили дорогу и не сходят с неё, не давая другого выбора — не по злому умыслу, а от богатства — и вот, ресурсы, которые могли быть использованы для альтернативных исследований, иссякают. Но однажды их богатство исчезнет, поскольку будет растрачено на сталкивание с дороги других.

В конце концов, никто не агитирует менять то, как мы работаем с объектами и объектно-ориентированным языками. Нет, просто необходимо разнообразие, необходимо работать над новыми парадигмами, чтобы "дать расцветать тысячам ". Самовосстанавлиающиеся, распределённые и сложные системы, самоорганизующиеся, адаптирующиеся, растущие по крупицам как живые организмы, использующие статистическое поведение, эволюционирующие, использующие избыточность ради безаварийности — систем, в основе которых могут быть ещё сотни различных перспективных идей и подходов, о которых мы еще и не думали — всё должно быть позволено и поддержано для движения вперёд.

Сейчас самое время для определения новой и смены старой парадигмы. Часто новое вообще не выглядит научно, или рационально, статьи и разговоры, объясняющие новые идеи могут звучать как пропаганда, как фантастика, или даже как поэзия; действительност же играет большую роль, чем теоремы или точные результаты. Можно сказать, что это не будет наукой в привычном понимании.

В свете всего вышесказанного, можно с уверенностью заявить, что объектная парадигма обманула наши ожидания.

Перевод Александра Майбороды ака HandleX, июль 2004г.