Труды международной конференции «Параллельные вычисления и задачи управления» (PACO'2001). - Москва: Институт проблем управления, 2001. - С.(4)-121-139. Электрон. опт. диск (CD-ROM). - ISBN 5-201-09559-3.

УДК 519.681

ПРИМЕНЕНИЕ ТЕХНОЛОГИИ КОНТЕКСТНОГО ПРОГРАММИРОВАНИЯ ДЛЯ РЕШЕНИЯ БОЛЬШИХ ПРИКЛАДНЫХ ЗАДАЧ

В.Я. Иосенкин
Приднестровский государственный университет им. Т.Г. Шевченко
Молдова, 3300, Тирасполь, 125 Октября ул., 128
e-mail: slava@tiraet.tirastel.md

В.С. Выхованец
Приднестровский государственный университет им. Т.Г. Шевченко
Молдова, 3300, Тирасполь, 125 Октября ул., 128
e-mail: srb@tirastel.md

Аннотация. Рассматривается объектно-ориентированная контекстная технология программирования. В рамках компонентной технологии разработан контекстный язык программирования Esse, который может быть успешно применен при решении задачи связанных с пониманием естественно-языковых текстов, планированием поведения, обучения, принятия решений, управления изменениями среды, в параллельных вычислительных и в суперкомпозитных системах.
Ключевые слова: контекст, контекстный язык, контекстная технология программирования, Esse, объектно-ориентированный Forth, объектно-ориентированное программирование, компонентная технология.

USE OF THE TECHNOLOGY of CONTEXT-SENSITIVE PROGRAMMING FOR SOLUTION OF THE MAJOR APPLICATIONS

V.Y. Iosenkin, *Dniester State University, 128 25th of October, Tiraspol 3300, Moldova,* e-mail: slava@tiraet.tirastel.md
V.S. Vykhovanets, *Dniester State University, 128 25th of October, Tiraspol 3300, Moldova,* e-mail: srb@tirastel.md
Abstract. The object-oriented context-sensitive technology of programming is considered. Within the framework of the component technology the context-sensitive programming language Esse is developed which can be successfully applied at solution of the task bound with understanding of the natural language texts, planning of behaviour, training, decision making, handle of changes of the environment, in parallel computing and in supercomposite systems.
Key words: context, context-sensitive language, context-sensitive technology of programming, Esse, object-oriented Forth, object-oriented programming, component technology.

1.Введение

На протяжении всего периода развития программных технологий, программная инженерия шла по пути упрощения процесса программирования – процесса без которого невозможно создание ни одного современного аппарата или механизма. Программные комплексы и компьютеры, средства телекоммуникаций, средства безопасности, средства управления и прогнозирования – основные реципиенты программных продуктов. Бурное развитие новой информационной технологии и расширение сферы ее

Труды международной конференции «Параллельные вычисления и задачи управления» (PACO'2001). - Москва: Институт проблем управления, 2001. - С.(4)-121-139. Электрон. опт. диск (CD-ROM). - ISBN 5-201-09559-3.

при-менения привели к интенсивному развитию программного обеспечения (ПО). Достаточно отметить, что в 1996 г. мировым сообществом на программное обес-печение затрачено свыше 110 млрд долларов. Причем тенденции развития ПО показывают, что динамика затрат имеет устойчивую тенденцию к росту, примерно 20% в год. Но, как и в прежние годы, в наши дни труд программиста остается наиболее сложным и трудоемким.

Процесс программирования прошел путь развития от простейших интерпретаторов и ассемблеров (не считая прямого аппаратного программирования в машинных кодах) до структурной, а затем и объектно-ориентированной технологий программирования. Программирование становилось более простым и понятным за счет приближения структуры языков программирования к естественым языкам и оперирования теми понятиями, которые наиболее близки специалистам данной области знаний. Вместе с тем росло количество языков программирования, появлялись специализированные языки, решающие узкие классы задач, менялась методология и технология програмирования, становясь доступной все большему количеству пользователей.

Вместе с развитием програмных средств параллельно идет совершенствование аппаратных средств, которые в свою очередь стимулируют развитие программной индустрии. Более мощные аппаратные платформы позволяют использовать более совершенные программные технологии. Языки пограммирования становятся все нагляднее, удобнее и интеллектуальнее. Программист при программировании начинает оперировать теми понятиями и сущностями, которые известными ему по роду его деятельности (объектно-ориентированная методология), что заметно упрощаетсам сам процесс программирования и повышает его надежность и скорость разработки продукта. То есть идет процесс «очеловечивания» языков программирования.

Основной проблемой программирования на естественном языке является языковая неоднозначность, что сваязано с контестной интерпретацией слов в естественных языках (но существующие языки прогаммирования представляются контекстно-свободной грамматикой). Существуют разные виды неоднозначности:

- Синтаксическая (структурная) неоднозначность: во фразе Time flies like an arrow для ЭВМ неясно, идет ли речь о времени, которое летит, или о насекомых, т.е. является ли слово flies глаголом или существительным.

- Смысловая неоднозначность: во фразе The man went to the bank to get some money and jumped in слово bank может означать как банк, так и берег.

- Падежная неоднозначность: предлог in в предложениях He ran the mile in four minutes/He ran the mile in the Olympics обозначает либо время, либо место, т.е. представлены совершенно различные отношения.

- Референциальная неоднозначность: для системы, не обладающей знаниями о реальном мире, будет затруднительно определить, с каким словом - table или cake - соотносится местоимение it во фразе I took the cake from the table and ate it.

- Литерация (Literalness): в диалоге Can you open the door? — I feel cold ни просьба, ни ответ выражены нестандартным способом. В других обстоятельствах на вопрос может быть получен прямой ответ yes/no, но в данном случае в вопросе имплицитно выражена просьба открыть дверь.

Труды международной конференции «Параллельные вычисления и задачи управления» (PACO'2001). - Москва: Институт проблем управления, 2001. - С.(4)-121-139. Электрон. опт. диск (CD-ROM). - ISBN 5-201-09559-3.

Помимо приближения языков программирования к естественному языку имеется и противоположное движение – разработка человеко-машинных языков – искусстыенных языков без присущих естественным языкам недостаткам. Так на земном шаре существует более трех тысяч различных языков. Объем знаний увеличивается вдвое каждые 50 лет, а объем информации удваивается каждые 10 лет. Глобальная экспансия электронно-вычислительной техники во все сферы человеческой деятельности и ее стремительное развитие обусловили необходимость создания информационных машин (комплексов), способных переводить с любого языка через язык-посредник на любой другой, хранить информацию на этом языке-посреднике, обрабатывая ее всевозможными программами (анализ, сортировка, поиск и пр.), и держать всю эту информацию на языке-посреднике в актуальном состоянии. Примером такого языка может служить человеко-машинный язык Эльюнди, разработанный Колеговым А.В. [1].

2.Исследование современного состояния комплексной проблемы взаимодействия открытых информационных систем

2.1.Открытая информационная система

Создание, сопровождение и развитие современных сложных информационных систем (ИС) базируется на методологии построения открытых систем [2, 3]. Открытые информационные системы создаются в процессе информатизации всех основных сфер современного общества: органов государственного управления, финансово-кредитной сферы, информационного обслуживания предпринимательской деятельности, производственной сферы, науки, образования и т. д. Развитие и использование открытых ИС неразрывно связаны с применением функциональной стандартизации информационной технологии (ИТ).

Термин "открытая система" сегодня можно определить как "исчерпывающий и согласованный набор международных стандартов информационных технологий и профилей функциональных стандартов, которые специфицируют интерфейсы, службы и поддерживающие их форматы, чтобы обеспечить интероперабельность и мобильность приложений, данных и персонала". Это определение, сформулированное специалистами IEEE, подчеркивает аспект среды, которую предоставляет открытая система для ее использования.

Общие свойства открытых ИС можно сформулировать следующим образом [4]:

- расширяемость/масштабируемость: обеспечение возможности добавления новых функций ИС или изменения некоторых уже имеющихся при неизменных остальных функциональных частях ИС;
- мобильность/переносимость: обеспечение возможности переноса программ, данных при модернизации или замене аппаратных платформ ИС и возможности работы с ними специалистов, пользующихся ИТ, без их переподготовки при изменениях ИС;
- интероперабельность: способность к взаимодействию с другими ИС;
- дружественность к пользователю.

Труды международной конференции «Параллельные вычисления и задачи управления» (PACO'2001). - Москва: Институт проблем управления, 2001. - С.(4)-121-139. Электрон. опт. диск (CD-ROM). - ISBN 5-201-09559-3.

Все эти общепринятые свойства современных открытых систем, взятые по отдельности, были характерны и для предыдущих поколений ИС и средств вычислительной техники. Новый взгляд на открытые системы определяется тем, что эти черты рассматриваются в совокупности, как взаимосвязанные, и реализуются в комплексе, что вполне естественно, поскольку все указанные выше свойства дополняют друг друга. Только в совокупности возможности открытых систем позволяют решать проблемы современных ИС.

Обобщенная структура любой ИС представляется состоящей из трех взаимодействующих частей:

- функциональной части, включающей прикладные программы, которые реализуют функции прикладной области;
- среды или системной части, обеспечивающей исполнение прикладных программ;
- аппаратных средств, являющихся основой для функционирования системной части.

С этим разделением тесно связаны две группы вопросов стандартизации:

- стандарты интерфейсов взаимодействия прикладных программ со средой ИС (Application Program Interface - API);
- стандарты интерфейсов взаимодействия самой ИС с внешней для нее средой (External Environment Interface - EEI).

Эти две группы интерфейсов определяют спецификации внешнего описания среды ИС - архитектуру с точки зрения конечного пользователя, проектировщика, прикладного программиста, разрабатывающего функциональные части ИС.

Спецификации внешних интерфейсов среды ИС и спецификации интерфейсов взаимодействия между компонентами самой среды, - это точные описания всех необходимых функций, служб и форматов определенного интерфейса. Совокупность таких описаний составляет модель открытых систем (Reference model).

Для сложных и ответственных ИС важно предусмотреть наличие в модели следующих средств:

- защиты информации;
- встроенных инструментальных средств, предназначенных для развития и модернизации ИС силами пользователей;
- средств интернационализации/локализации используемых программных продуктов, помогающих повторно использовать готовые программы.

Можно отметить, что современные тенденции в идеологии открытых систем направлены на освобождение потребителя от зависимости от одного определенного поставщика технических или программных средств. Прежде всего это касается обеспечения переносимости (мобильности) программного обеспечения между разными аппаратными платформами. С этим связано постепенное смещение стандартизованных решений архитектуры от нижнего к верхним уровням модели среды ИС, хотя необходимость стандартизации ряда общих вопросов аппаратуры сохранится.

Труды международной конференции «Параллельные вычисления и задачи управления» (PACO'2001). - Москва: Институт проблем управления, 2001. - С.(4)-121-139. Электрон. опт. диск (CD-ROM). - ISBN 5-201-09559-3.

2.2.Проектирование программных средств

С практической точки зрения успех проектирования открытой ИС зависит от используемой методологии программирования - интегрального, сформулированного в виде методов, методик и приемов, подхода к основным стадиям проектирования программных средств.

В связи с развитием и расширением сфер применения открытых систем весьма перспективным направлением представляется объектно-ориентированная (ОО) методология проектирования. В статье [5] Г. Буч утверждает, что логика развития программной инженерии, мотивируемая прежде всего борьбой со сложностью, через серию революционных смен парадигм программирования: алгоритмическая абстракция - абстракция данных - абстракция типов данных, привела к объектно-ориентированной парадигме, которая и воплощает в себе логичный, стабильный и окончательный организующий принцип разработки программного обеспечения.

На практике дела с ОО подходом обстоят не так хорошо. Как отмечает Т. Льюис [6] ОО модель далека от реальности. Например в объектной линии DLE-OLE-ActiveX различные версии этих продуктов столь не похожи друг на друга, и столь архитектурно ущербны, что становится ясно: в процессе разработки вместо заранее продуманного плана имели место мало управляемые мутации. Аналогичные проблемы имеются и у Microsoft. Об этом свидетельствует и статистика [7]: в 1995 году ожидалось, что только 9% проектов, выполняемых крупными компаниями, будут завершены в срок и без превышения запланированного бюджета; 52% проектов должны были стоить в среднем 189% от их первоначально оцененной стоимости; в то же время 31% всех проектов вообще ожидало приостановление или полное прекращение, причем затраты на них - ничем не компенсируемые убытки - оценивались ни много ни мало в 81 млрд. долларов.

Корни этой проблемы - в объективной трудоемкости, интеллектуальной и технологической сложности процесса программирования. По данным фирмы Software Productivity Research [8] наиболее надежной мерой для измерения производительности программистского труда (и одновременно содержательной метрикой применительно к программе) признается функциональный пункт (function point, FP) - взвешенная сумма присутствующих в блоке программного кода вводов, выводов, запросов к данным и общих обрабатывающих операций с учетом наличия распределенной обработки, степени повторного использования кода и т.п. Производительность труда программиста, выражаемая в виде усредненной стоимости одного FP возрастает в среднем на 4.6% в год и составляет сейчас приблизительно 1000 долларов на FP. Усреднение проводилось по специальной методике, позволяющей учесть различные виды программных приложений, написанные на различных языках и имеющие разные размеры.

Особенно остро проблемы проектирования программных средств выступают на фоне сверхбыстрого развития индустрии аппаратных средств, прежде всего - производства все более быстрых процессоров и средств коммуникации: в соответствии с законом Мура, производительность микропроцессоров в среднем удваивается каждые 18 месяцев при неизменном уровне цен, что означает рост со скоростью 48% в год [9].

Труды международной конференции «Параллельные вычисления и задачи управления» (PACO'2001). - Москва: Институт проблем управления, 2001. - С.(4)-121-139. Электрон. опт. диск (CD-ROM). - ISBN 5-201-09559-3.

2.2.1.Объектно-ориентированная методология

ОО методология предлагает небольшой, и в определенном смысле исчерпывающий набор из пяти принципов [10]. В совокупности они предлагают нечто большее, чем обычно упоминаемая триада из инкапсуляции, наследования и полиморфизма.

1). Децентрализация. ОО методология позволяет сфокусироваться на высокоуровневой структуре программной системы, издаваемой как сеть сотрудничающих агентов-объектов, между которыми определены только два отношения: поставщик-клиент и родитель-наследник. Именно эти ограничения обеспечивают концептуальную целостность системы (conceptual integrity). При этом уходят от таких понятий, как "основная программа", "глобальные переменные", "проектирование сверху вниз" и т.п., предполагающих, что система обязательно имеет некий центр.

2). Контракты. Отношения между клиентами и поставщиками - это наиболее уязвимый аспект архитектуры системы. Контракты - это четкое выражение этих отношений, специфицирующие ожидания и обещания каждой стороны. Для клиента формулируются предусловия, а для поставщика - постусловия, что позволяет инициировать взаимодействие в нужный момент, в нужном контексте и гарантированно получить нужный результат. Тогда поставщик гарантирует выполнение постусловия в случае если клиент гарантирует удовлетворение предусловия. Большинство коммерческих ОО языков не предлагают явных средств для реализации контрактов; соответственно, этот критический для построения надежных систем принцип используется в практике разработки не слишком часто.

3). Самодостаточность (selfishness). Как замечает Б. Мейер [10], это наиболее известная, но наименее понимаемая часть ОО методологии. Обычно используются более привычные, но и в какой то степени затуманивающие суть термины - "скрытие информации" или "инкапсуляция". Основная идея в том, что объект-поставщик раскрывает клиентам только часть своих свойств, причем в идеале именно в том объеме, который необходим. Большинство программистов привыкло считать, что этот механизм помимо возможности изменения реализации без изменения спецификации введен прежде всего для защиты поставщика от несанкционированного клиентского доступа, дабы тот не прознал тайны реализации, или не испортил ее. Это чисто программистский взгляд, имеющий мало общего с существом ОО методологии. В этом же контексте стоит рассматривать и понятие динамического связывания. Как клиент, вы отказываетесь беспокоиться о некоторых деталях, в данном случае - о том, какой вариант полиморфной операции выполнять, до самого последнего возможного момента - момента исполнения.

4). Классификация. Какие объекты (классы и их экземпляры) использовать в конкретной разработке - вот критический вопрос при проектировании программных средств. То, что в ОО технологии называется "создание иерархий с наследованием" и есть классификация, отображающая неупорядоченный реальный мир в упорядоченный мир объектов с попутным обнаружением общности в их ключевых абстракциях и реализующих поведение механизмах. Существует эмпирически выявленная закономерность [9]: 70% классов

относительно легко идентифицировать уже на стадии эскизного проекта; 25% классов возникают на стадиях технического и рабочего проектов; наконец, необходимость остающихся 5% часто выявляется только на этапе поддержки и сопровождения системы.

5). Бесшовность. Идея бесшовности (связности, цельности - seamlessness) относится к наименее понимаемым аспектам ОО методологии. Суть ее в том, что модели системы на этапах анализа, проектирования и реализации обладают такой степенью связности, что изменения в одной из них можно легко и безболезненно отобразить на другие. Это позволяет реализовать такое важное при итеративном проектировании свойство, как трассируемость (traceability). Очевидно, что при этом должно быть четкое (прежде всего - структурное) соответствие между элементами моделей на различных фазах жизненного цикла. Из бесшовности следует обратимость (reversibility), гарантирующая согласованное изменение моделей не только в прямом (от требований к реализации), но и в обратном направлении. Свойства трассируемости и обратимости являются основой для реализации эволюционных моделей жизненного цикла программных систем.

В конечном итоге, этот набор принципов обеспечивает концептуальный базис для борьбы со сложностью - тем свойством программного обеспечения, которое является его неотъемлемой частью, и от которого нельзя избавиться. Однако, сложностью можно управлять, если учитывать, что все хорошо структурированные сложные системы имеют пять общих характеристик:

- сложность принимает форму иерархии;
- выбор примитивных компонент относительно произволен;
- внутри компонентные связи сильнее межкомпонентных;
- иерархические системы состоят из небольшого числа подсистем в различных сочетаниях;
- работающая сложная система получается только как развитие работающей простой системы.

Таким образом, ОО методология является не столько методологией проектирования программных средств, сколько фундаментальным базисом для анализа и проектирования сложных систем произвольной природы.

2.2.2.Объектно-ориентированные технологии

Успех применения ОО методологии зависит от используемой технологии - системы методов, способов и приемов разработки программных средств. От иных, в частности структурных подходов, ОО технологии отличаются прежде всего тем, что необходимой фундаментальной единицей архитектурной абстракции служит класс. В конце 80-х - начале 90-х наблюдался бум в создании новых ОО технологий. Эти подходы в новой тогда области ОО анализа и проектирования разрабатывались, как правило, независимо друг от друга и практически без учета опыта конкурентов.

По сложившейся практике, наряду со смысловыми названиями (или вместо них), технологии получали имена своих создателей. Среди них Booch, OMT (Object Modeling Technique), Object-Oriented Systems Analysis and Recursive Design (Shlaer/Mellor), CRC (Class, Responsibility, and Collaborations), Responsibility-Driven Design (Wirfs-Brock), Object-Oriented Analysis and Design

(Coad/Yourdon) и другие. Почти все технологии имели оригинальные графические нотации и нашли поддержку в виде соответствующих CASE-средств. Появились и CASE-средства, поддерживающие одновременно несколько технологий.

О единой ОО технологии говорят с осторожностью, как о технологии будущего [11]; пока же работа концентрируется на определении как концептуальных, так и исполняемых ОО артефактов (artifacts), под которыми понимаются "реальные продукты-сущности процесса разработки", а также взаимосвязей между ними. Создана и систематически описана базисная метамодель [12], определившая семантику всех моделирующих понятий, необходимых на этапах анализа, проектирования и конструирования сложных систем. Удалось выработать единую нотацию - унифицированный моделирующий язык (Unified Modeling Language, UML). Это будет означать, что UML может быть использован для описания процессов в рамках различных технологий.

Значительно более амбициозной, направленной не просто на традиционный процесс анализа, проектирования и реализации программного продукта, но имеющей целью охватить полный жизненный цикл программного проекта, включая стратегии стратегического планирования, управления и оценки качества, повторного использования унаследованных систем, сопровождения и др., является методология OPEN (Object-oriented Process, Environment and Notation) [13]. Неформальный коллектив, состоящий из двух десятков всемирно признанных экспертов, вознамерился создать не просто новую технологию, а скорее "метатехнологию" в виде открытой ОО инфраструктуры, которая инкорпорирует большое количество методов, способов и приемов из практически всех существующих ОО технологий.

Лежащая в основе технологии OPEN модель жизненного цикла сама является полностью ОО и представляется как сеть взаимодействующих объектов, воплощающих процессы разработки на макроуровне. Эта сеть процессов конфигурируется для конкретной организации и/или конкретного проекта. Принципиально важно, что переходы между процессами управляются контрактами; поэтому войти в новую фазу разработки - перейти к соответствующему процессу можно лишь при удовлетворении предусловия данного процесса. Удовлетворение постусловия процесса является гарантией качества его выполнения. Каждый процесс на микроуровне ассоциируется с рядом задач. Пары процесс/задача - это, в сущности, ОО образцы, о которых уже шла речь, и определение набора таких пар является составной частью настройки ОО методологии на разработку конкретного проекта. Наконец, для выполнения каждой задачи необходимо задействовать методики - конкретные способы и механизмы выполнения. Их набор весьма широк - уже сейчас можно выбирать их более чем сотни.

Разработчики полагают, что реализация OPEN технологии и ее принципиальная открытость гарантируют ей долгую жизнь независимо от грядущих потрясений в программной инженерии.

Труды международной конференции «Параллельные вычисления и задачи управления» (PACO'2001). - Москва: Институт проблем управления, 2001. - С.(4)-121-139. Электрон. опт. диск (CD-ROM). - ISBN 5-201-09559-3.

2.2.3.Объектно-ориентированные языки

При всем кажущемся благополучии в области построения открытых ИТ-систем узким местом остается, казалось бы давно решенная проблема, - стандартизация языков программирования и их библиотечных окружений, составляющих основную часть прикладного интерфейса. Существует более сотни ОО языков; впрочем, лишь немногие имеют такой набор средств, который позволяет с их помощью вести промышленные разработки.

Стоит отметить, что ввод в практику новых языков программирования не слишком принципиально изменяет отношение SLOC/FP (Source Lines Of Code per Function Point - количество строк на функциональный пункт), которое уменьшается в среднем на 11% в год (для C++ это отношение около 100, для Smalltalk - около 20) [8]. Это означает, что мощность языков и средств программирования растет быстрее, чем производительность программистского труда, но гораздо медленнее производительности аппаратуры. При этом прогноз относительно дальнейшего прогресса этого показателя в обозримом будущем не выглядит утешительным: показатель мощности языка SLOG/FP не может быть значительно улучшен. Приведем из [9] список распространенных ОО языков.

Наиболее известным языком является C++. Его достоинства и недостатки, проистекающие из его гибридности (сосуществования в нем наряду с объектами процедурных сущностей), хорошо известны. Отмечается негативное отношение к C++ многих классиков программирования, полагающих, что именно доминирование C++ обусловило многие беды индустрии программирования. Соперничает с C++ Smalltalk, который сегодня лидирует в разработках для ряда вертикальных рынков, особенно это заметно в финансовом секторе (где так важно быстрое изготовление системы), а также в архитектуре клиент-сервер, обычно с клиентской стороны.

Наиболее элегантным признается Eiffel, являющийся, пожалуй, самым концептуально чистым ОО языком. Особо отмечается, что в отличие от других языков (и прежде всего от C++), в этом языке явно присутствует контрактная модель, поддержанная предусловиями, постусловиями, инвариантами и механизмом исключений. Это позволяет резко повысить надежность разрабатываемых программных средств, особенно в контексте повторного использования. Как следствие, Eiffel наиболее подходит для изучения в качестве первого ОО языка.

Самым нетребовательным к ресурсам является язык Oberon [14]. Отмечается лишь принципиальное нежелание разработчика языка Н. Вирта использовать такие фундаментальные ОО термины, как "наследование", "класс" или "метод", которые, по его мнению, ничего не добавляют к понятиям "совместимость", "тип" и "процедура". Коммерческие перспективы этого во многих отношениях замечательного языка связаны с судьбой недавно заявленной технологии Juice [15].

Наиболее надежным для разработки сверхсложных систем является язык Ada. Указывается, что Ada 9X - это не просто объектный, но полностью ОО язык.

Наиболее гибким признан язык CLOS. Будучи интегрированным расширением языка Common Lisp и обладая мощными средствами определения полиморфных операций, он дает программисту очень большую свободу

действий во время исполнения, позволяя динамически вводить новые классы и добавлять новые методы. Еще одна уникальная для коммерческих ОО языков особенность - отсутствие механизмов предотвращения доступа одного класса к деталям реализации другого. Такое пренебрежение к сокрытию внутренней структуры класса - одного из базисных принципов ОО программирования - принято считать крайне нежелательным. Существует отличающаяся от ОО парадигма программирования - "определительное программирование" (definitive programming), где ослаблены принципы инкапсуляции и целостности. Здесь объектам позволяется не просто "заглядывать" внутрь друг друга, но и достаточно радикально изменять значения атрибутов как численные, так и функционально или процедурно заданные, а то и саму структуру другого объекта. Это может быть особенно полезно при моделировании небольших по размеру, но сложных по поведению систем с сильным межобъектным взаимодействием. Впрочем, реализация систем определительного программирования пока не вышла за рамки академических разработок.

Наиболее любопытным с точки зрения результата использования является ОО язык Cobol. Отмечается, что около 80% всего программного обеспечения написано именно на нем (50 миллиардов строк). Более 2 миллионов программистов продолжают разрабатывать и поддерживать Cobol-программы.

Самый перспективный и рекламируемый - это язык Java. Как указывает Э. Йодан [16], концептуальная ценность языка Java не в синтаксисе и даже не в органической связи с Web-технологией, а в определении вектора движения к компонентной технологии. Главное следствие - прекращение имевшей место последние годы слишком объемного программного обеспечения и рост индустрии программирования, которую можно ассоциировать с созданием Java-апплетов, прикрепленных к Web-страницам. Эти транзакционно-ориентированные апплеты можно будет арендовать вместо покупки монолитных программных пакетов. Подход, реализованный в языке Java важен даже не сам по себе, а как провозвестник принципиально новой компонентной технологии.

2.3.Компонентная технология программирования

В отличие от нынешних ОО языков, где фундаментальным средством абстракции является класс, в новых языках в этом качестве выступает более крупное понятие - специфическая для отдельного приложения интегрированная среда (application-specific framework); а вычислительная модель опирается не на стандартное для ОО языков взаимодействие типа "метод/сообщение", а на взаимодействие "событие/обработчик".

Нынешние интегрированные ОО среды, такие как Taligent или Open Step, могут рассматриваться как прообразы будущего. В соответствии с принципом децентрализации, останется в прошлом подход "главная программа плюс компоненты". Вместо этого, программы будут конструироваться настройкой существующей интегрированной среды с помощью компонентов - модулей, которые не просто инкапсулируют функциональность и данные, но могут быть динамически встраиваемыми (plug-in).

Для программиста это означает, что ему не нужно писать основную программу и затем начинать поиск повторно используемых компонентов, чтобы

Труды международной конференции «Параллельные вычисления и задачи управления» (PACO'2001). - Москва: Институт проблем управления, 2001. - С.(4)-121-139. Электрон. опт. диск (CD-ROM). - ISBN 5-201-09559-3.

их в эту программу встроить; теперь он ищет компоненты, содержащие такие обработчики событий, которые могут быть использованы для перегрузки существующих методов внутри готовой к исполнению среды, чья задача - послать сообщение к объектам с тем, чтобы обработать событие. Некоторые из этих событий будут обрабатываться самой средой, другие - методами из встроенных компонентов. Эти методы-апплеты обрабатывают сгенерированные пользователем события путем перегрузки методов среды.

При этом принципиально важна динамическая природа ситуации: функциональность системы подгружается на арендной основе в тот самый момент, когда она пользователю потребовалась, и сразу же происходит исполнение. По существу, это означает смерть технологии клиент-сервер в ее нынешнем виде. В пределах следующего десятилетия на первый план выйдут динамически компилируемые и исполняемые на различных платформах языки с компонентами в качестве базисных сущностей и методами - обработчиками событий. Языки Java, Lingo, Telescript - их теперешние прообразы.

Принципиальное отличие этих, содержащих апплеты-обработчики событий, от их теперешних ОО образцов в том, что технология их производства (где профессионалы-программисты выполняют сложное инженерное проектирование сред и компонентов) будет отделена от технологии их потребления. Именно пользователи будут управлять динамическими вычислениями, определяя порядок обработки и продолжительность исполнения (что, собственно, и определяет необходимость вычислительной модели "событие/обработчик").

3.Сложность программного обеспечения, как фактор понижения надежности, борьба с ней

В ближайшие пять лет ожидается резкое увеличение сложности программного обеспечения, предназначенного для информационных систем различного класса [17]. Следствием этого станет с одной стороны ужесточение требований к характеристикам компьютеров, сетевого оборудования, пропускной способности каналов связи, а также определение оптимального распределения нагрузки в узлах информационных систем, в которых ресурсы закрепляются за конечным пользователем по принципу «ровно столько, сколько нужно», а с другой стороны - необходимость создания новых программных технологий и методологий, позволяющих эффективно бороться со сложностью, для которых базой будут являться объектно-ориентированная и компонентная технология.

По сравнению с программной методологией и инструментальными средствами роль языка в программировании традиционно принижается; и не только преуменьшается, но и полностью отвергается, когда утверждают, что хорошо разработанная система может быть одинаково хорошо реализована на любом языке [18]. Но языки программирования — это не просто инструментальное средство; это тот «материал», из которого создается программное обеспечение, то, что мы видим на наших экранах большую часть дня. Язык программирования — один из наиболее, а не наименее важных факторов, которые влияют на окончательное качество программной системы.

Труды международной конференции «Параллельные вычисления и задачи управления» (PACO'2001). - Москва: Институт проблем управления, 2001. - С.(4)-121-139. Электрон. опт. диск (CD-ROM). - ISBN 5-201-09559-3.

С этим недостатком знания связаны две серьезные проблемы разработки программного обеспечения. Первая — крайний консерватизм в выборе языков программирования. Несмотря на бурное развитие компьютерной техники и сложности современных программных систем, большинство программ все еще пишутся на языках, которые были разработаны около 1970 г., если не раньше. Многие исследования в области языков программирования никогда не подвергались проверке практикой, и разработчики программ вынуждены с помощью различных инструментальных средств и методологий компенсировать устаревшую языковую технологию.

Вторая проблема состоит в том, что языковые конструкции используются без должного отбора, практически без учета надежности и эффективности, Это ведет к созданию ненадежного программного обеспечения, которое невозможно поддерживать, а также к неэффективности, которая устраняется скорее путем кодирования отдельных фрагментов программ на языке ассемблера, чем совершенствованием алгоритмов и парадигм программирования.

Традиционно при разработке программного обеспечения больших систем основное внимание уделяется разработке управляющего процесса, выполняемого на универсальном процессоре или наборе специализированных процессоров, основными требованиями к которому являются обеспечение необходимых уровней производительности, надежности и устойчивости.

Рассмотрим требования к большим прикладным системам, выявленные при решении практических задач.

Сопровождаемость. Современные системы разрабатываются с учетом их использования в течение достаточно большого срока. В этот период неоднократно возникает необходимость в ее модификации, вызываемая различными причинами (расширение функциональных возможностей, изменение стандартов и правил взаимодействия и т.д.).

Управляемость процессом разработки. Большие прикладные системы, как правило, отличаются значительным объемом и структурной сложностью аппаратных средств и решаемых задач и большим набором выполняемых функций. Всё это требует разработки большого объёма программного обеспечения. Достигая определенных границ, сложность программного обеспечения влечет утрату контроля над его разработкой и вынуждает разработчиков совершать ошибки, которые впоследствии достаточно тяжело исправить. Привлечение к разработке большого числа специалистов влечет за собой достаточно объемную работу по координации их совместной деятельности. Одной из наиболее важных задач здесь является достижение как можно более полного взаимопонимания между различными группами вовлеченных в разработку специалистов. Получаемые на разных этапах описания и спецификации должны обеспечивать их недвусмысленное толкование. Поэтому важная роль отводится используемым формальным средствам спецификации.

Гибкость. Редко бывает так, чтобы при разработке болшой системы была точно определена структура её программно-аппаратных средств и параметры внешнего взаимодействия. Большинство систем позволяют настраиваться на эти характеристики. Для больших систем такая настройка заключается в создании базы данных, содержащей все необходимые параметры. Настройка может выполняться как статически до запуска системы, так и динамически в процессе

её функционирования. Динамическая настройка осложняется тем, что изменения в конфигурации необходимо делать “по живому”, т.е. в данных, которые могут в этот момент использоваться функциональными процессами, что требует согласования между ними и процессом настройки.

Устойчивость. Устойчивость системы означает её готовность реагировать на непредсказуемое поведение пользователей и желательно администраторов тоже. Часто от большой прикладной системы требуется также отказоустойчивость, означающая возможность системы продолжать нормальное функционирование в условиях наличия ошибок.

Стабильность. Встроенная система может находиться в одном из двух состояний: активном и пассивном. В активном состоянии осуществляется функционирование элементов (возможно, не всех) программного обеспечения системы, и система выполняет все или некоторые из своих функций. При этом полное состояние системы описывается состоянием аппаратуры и состоянием данных в вычислительной среде. При переходе системы в пассивное состояние происходит остановка программного обеспечения. При этом не все используемые им данные теряют свою актуальность, часть их должна быть сохранена до следующего запуска системы, что заставляет хранить эти данные в энергонезависимой памяти. В качестве средств обработки таких данных выступают системы баз данных, обеспечивающие организацию, хранение и доступ к данным.

Непрерывность. Для большинства систем реального времени является критичным непрерывное и надежное функционирование. Непрерывность функционирования означает невозможность остановки системы, связанную, как правило, с важностью выполняемых ею процессов и круглосуточной работой пользователей системы.

Параллельность. Параллельность является характерным свойством большинства больших прикладных систем. Наличие аппаратных элементов, работающих в реальном параллельном режиме и одновременное выполнение нескольких функциональных процессов требуют применения параллельных вычислений. Хотя во многих случаях достаточно псевдопараллельности.

Требованиями к технологии разработки программного обеспечения больших программных систем являются высокие требования к качеству, производительности и, особенно, надежности применяемых средств. Необходима также наличие целостной методологии - в основе технологии должна лежать система методов и принципов, охватывающая все этапы разработки прикладной системы.

Итак, можно заключить, что главным фактором влияющим на скорость разработки ПО, а также на его качество и надежность является сложность ПО. А важнейшей задачей, которую предстоит решить будущим технологиям программирования, является борьба со сложностью ПО, в значительной степени это позволяет сделать контекстная технология программирования.

4.Контекстная технология программирования

Создание, сопровождение и развитие современных суперкомпозитных (сверхсложных) систем (СС) базируется на методологии построения открытых систем [19], обладающих свойствами расширяемости (масштабируемости),

Труды международной конференции «Параллельные вычисления и задачи управления» (PACO'2001). - Москва: Институт проблем управления, 2001. - С.(4)-121-139. Электрон. опт. диск (CD-ROM). - ISBN 5-201-09559-3.

мобильности (переносимости), интероперабельности и дружественности к пользователю [20]. С практической точки зрения успех проектирования СС зависит от используемой методологии программирования - интегрального, сформулированного в виде принципов и методов подхода к основным стадиям проектирования программных средств.

Логика развития программной инженерии привела к объектно-ориентированной (ОО) парадигме. Успех применения ОО методологии зависит от используемой технологии программирования - системы способов и приемов разработки программных средств.

В результате исследования источников выявлено, что перспективной является компонентная объектно-ориентированная технология программирования, основанная на следующих принципах: полная поддержка объектно-ориентированной методологии путем реализации децентрализации, контрактности, самодостаточности, классификации и бесшовности; независимость программного обеспечения от аппаратной и операционной платформ; динамическая встраиваемость модулей (компонент), созданных на различных платформах; расширяемость систем за счет миграции кода в распределенной компьютерной среде; динамическая генерация кода и трансляция методов мигрирующего кода по мере их вызова; анализ мигрирующей программы на предмет нарушения правил типизации и областей видимости; реализация средств контроля за уровнем безопасности для защиты от помех и несанкционированного вторжения; поддержка разработки крупных проектов.

Как следствие перечисленных принципов выявлены требования к объектно-ориентированному языку программирования. Последний должен обладать следующими свойствами: использование вычислительной модели, опирающаяся на взаимодействие "событие/обработчик"; отказ от директив препроцессора, указателей, оператора goto, неявного преобразования типов, адресной арифметики; реализация различных видов проверок во время исполнения (проверка на пустой указатель, контроль выхода индекса за границы массива, и т.д.); реализация одинарного наследования и строгой типизации; встроенный механизм обработки исключительных ситуаций и "сборки мусора"; наличие средств многопоточного программирования; нейтральный по отношению к архитектуре формат представления кода; выполнение независимой, раздельной и распределенной компиляции; разграничение описаний интерфейсной и реализационной части модуля; наличие выразительных средств для высокоуровневого (безопасного и переносимого) и низкоуровневого (опасными и непереносимыми) программирования; возможность задания мигрирующих программных компонентов; поддержка динамической генерации кода.

Узким местом современных технологий программирования остается объективная трудоемкость, интеллектуальная и технологическая сложность процесса программирования [21, 22].

В рамках компонентной технологии предлагается использование контекстных языковых средств, позволяющих снизить расходы на программирование. Реализована контекстная система программирования, удовлетворяющая требованиям к компонентной технологии. Наиболее близким

Труды международной конференции «Параллельные вычисления и задачи управления» (PACO'2001). - Москва: Институт проблем управления, 2001. - С.(4)-121-139. Электрон. опт. диск (CD-ROM). - ISBN 5-201-09559-3.

аналогом разработанной системы является система программирования на базе языка FORTH [23].

Контекстный язык близок к естественным языкам и представлен в виде полуконтекстной формальной грамматики G = <T,N,P,I>, где T(N) - терминальный (нетерминальный) алфавит, $T \cap N = \varnothing$; P - множество продукций вида: $\alpha A \rightarrow \beta$, где $\alpha, \beta \in (T \cup N)^*$, $A \in N$; I - аксиома.

Основными синтаксическими единицами языка является определение словаря <имя> и словарной статьи <слово>:

<словарь> ::= vocabulary <имя> <базовый> <статьи> end
<базовый> ::= <имя> <видимость>
<видимость> ::= public | private | protected
<статьи> ::= <статья> | <статья> <статьи>
<статья> ::= <слово> (<контекст> --- <контекст>) <определение> <видимость>
<контекст> ::= <имя> | <имя> <контекст>
<определение> ::= <лексема> | <лексема> <определение> | <лексема> (<контекст>) <определение>
<лексема> ::= <имя> | <слово>

Словари образуют иерархическую структуру, соответствующую иерархии классов при ОО программировании. Поиск определения слова осуществляется по первому имени в текущем контексте, которое задает точку входа в дерево словарей. Выбор слова осуществляется по левому контексту. После компиляции слова, левый контекст заменяется на правый, что определяет семантику применения слова и изменяет контекст для следующего слова.

Контекст, задаваемый в определении слова обрабатывается компилятором и соответствует контексту, который должен быть после применения слова. Это позволяет конкретизировать выбор слова при полисемии, служит мощным средством для комментирования программы, а также обеспечивает со стороны компилятора контроль представлений программиста о выполняемой обработке данных.

Динамическое связывание реализуется путем поиска слова с заданным контекстом во время выполнения. Ниже приведен пример интерфейсной части словаря для работы с логическими данными:

```
vocabulary boolean object public
   boolean ( object --- boolean ) public
   ~ ( boolean --- boolean ) public
   & ( boolean boolean --- boolean ) public
   | ( boolean boolean --- boolean ) public
   ^ ( boolean boolean --- boolean ) public
   true ( --- boolean ) public
   false ( --- boolean ) public
end
```

В качестве вычислительного механизма использован магазинный автомат, имеющий 2 (3) стека: стек операндов, стек адресов возврата (стек контекста)

[24]. Использование третьего стека позволяет реализовать динамическое связывание при произвольном контексте и произвести динамическую генерацию кода. При миграции кода подгрузка дерева прикладных словарей осуществляется в заданную точку базовой иерархии, начальная часть которой имеет вид:

```
vocabulary reference reference public
vocabulary object reference public
vocabulary boolean object public
vocabulary char object public
vocabulary memory object public
vocabulary byte memory public
vocabulary string memory public
vocabulary abstract memory public
vocabulary word abstract public
vocabulary vocabulary abstract public
```

Управляющие структуры языка (условный оператор, циклы do, while и for) определяются как слова словаря compile. Ниже приведен пример реализации слова read для словаря потокового ввода-вывода:

```
read ( size byte stream --- size )
    local stream this =             ( size byte )
    local byte ptr =                ( size )
    local size siz =                ( )
    local size len 0 =              ( )
    true                            ( boolean )
    begin
            len @ siz @ < &         ( boolean )
    while
            this cget dup           ( char char )
            ptr @ ! ptr ++ len ++   ( char )
            cr !=                   ( boolean )
    repeart
    len @ protected
```

Объектно-ориентированный подход помогает справиться с такими сложными проблемами, как:

- уменьшение сложности программного обеспечения;
- повышение надежности программного обеспечения;
- обеспечение возможности модификации отдельных компонентов программного обеспечения без изменения остальных его компонентов;
- обеспечение возможности повторного использования отдельных компонентов программного обеспечения.

Но в совокупности с применением контекстной технологии первые две из них решаются значительно проще. Применение контекстной технологии позволяет разрабатывать хорошо структурированные, надежные в эксплуатации, достаточно просто модифицируемые программные системы.

Труды международной конференции «Параллельные вычисления и задачи управления» (PACO'2001). - Москва: Институт проблем управления, 2001. - С.(4)-121-139. Электрон. опт. диск (CD-ROM). - ISBN 5-201-09559-3.

5.Язык программирования Esse

Создание систем управления безопасностью связано с использованием различных систем программирования. К программным средствам систем управления безопасностью предъявляются повышенные требования к надежности, что определяет высокую стоимость их проектирования.

Создание надежных программ базируется на методологии объектно-ориентированного (ОО) программирования [25]. Первым этапом проектирования системы является ОО анализ предметной области, выполняемый с целью создания модели системы в терминах (понятиях), близких специалисту-прикладнику.

Известные технологии ОО программирования реализуются в рамках языков программирования, порождаемых узким классом контекстно-свободных формальных грамматик, допускающих эффективные процедуры синтаксического разбора и компиляции [26].

Формальная грамматика G задается четверкой объектов <V, W, P, I>, где V - терминальный алфавит, W - алфавит синтаксических единиц, P - правила вывода (продукции), I - аксиома или начальная синтаксическая единица. Продукции контекстно-свободной грамматики имеют вид $A \rightarrow \alpha$, где A - определяемая синтаксическая единица, $A \in W$, α - строка в алфавите $V \cup W$. Это означает, что синтаксические единицы языка определяются через комбинацию лексем и других синтаксических единиц.

Ограничением выразительных качеств известных языков программирования является бесконтекстность правил вывода, что не согласуется с явлением контекстной интерпретации слов в естественных языках и, как следствие, уменьшает надежность программирования.

Полуконтекстной называется грамматика, правила вывода которой имеют вид: $\beta A \rightarrow \beta\alpha$, где β - некоторая строка в алфавите W или левый контекст правила $A \rightarrow \alpha$ [27].

Приведем грамматику языка контекстного программирования **Esse.** Язык ***Esse*** позволяет приблизить процесс проектирования программных средств к парадигмам постановки и решения многих прикладных задач.

program → '***{***' knowledge '***}***' text
knowledge → essence | essence '**.**' knowledge
essence → name '***follows***' | name '***growing***'
name '***is***' nation | name '***is***' nation '***that***' description
description → sentence | sentence '***;***' description
sentence → context words '***makes***' context '***so***' definition
context → nation | nation context
nation '***as***' local | nation '***as***' local context
words → term context | term context words
definition → local | local definition
token | token definition
term → ′ token ′ | ′ token ′ term
text → token | token text

Труды международной конференции «Параллельные вычисления и задачи управления» (PACO'2001). - Москва: Институт проблем управления, 2001. - С.(4)-121-139. Электрон. опт. диск (CD-ROM). - ISBN 5-201-09559-3.

Программа состоит из декларации знаний о предметной области ***knowledge*** и описания некоторой ситуации ***text***, которая является исполняемой частью программы

Определяемые сущности ***essence*** образуют понятия ***nation*** с именами ***name*** находящиеся в отношении наследования. Каждое понятие определяется лексическими конструкциями ***sentence.*** В последних задается контекст их применения ***context***, последовательность ***words*** терминальных знаков ***term*** и понятий ***nation***, производящих новый контекст ***context*** путем выполнения действий ***definition***. В ***definition*** используются ранее описанные лексические конструкции и локальные значения понятий ***local*** из определяемой конструкции, посредством чего формируются значения нового контекста.

Лексемы ***token*** представляют собой последовательности знаков, разделенных пробельными знаками или знаками пунктуации, которые, в свою очередь, также являются лексемами.

Для привязки программы к конкретной вычислительной системе некоторые понятия, называемые первичными, определяются низкоуровневыми средствами, например, на языке ассемблера.

Esse - интерпретируемый язык: для него определены внутреннее представление (text) и интерпретатор этого представления, который был реализован для платформы Windows. Интерпретатор упрощает отладку программ, написанных на языке Esse, обеспечивает их переносимость на новые платформы и адаптируемость к новым окружениям. Он позволяет исключить влияние программ, написанных на языке Esse, на другие программы и файлы, имеющиеся на новой платформе, и тем самым обеспечить безопасность при выполнении этих программ. Эти свойства языка Esse позволяют использовать его как язык программирования для программ, распространяемых по сетям (в частности, по сети Internet).

Использование языка контекстного программирования позволяет сократить число ошибок, возникающих из-за семантического разрыва между применяемыми языковыми средствами и математической моделью системы управления безопасностью и, тем самым, повысить надежность разработки программ.

6.Заключение

Современное развитие программно-аппаратных средств позволяет решать все более сложные задачи со все большим размером кода, которые требуют новых программных решений. Разработка больших продуктов не возможна одним программистом, требуется слаженное взаимодействие целой комманды программистов. За счет контестной привязки и более гибкого построения дерева решений ожидается существенный выигрыш в размере кода. Разделение программы на декларативную и императивную части позволяет лучше проработать каждую из них, и следовательно повысить качество кода. Контекстная зависимость императивной части позволдяет представить процесс разработки, как последовательное формирование и изменение контекста. Таким образом легко разделить задачу на подзадачи с разным состоянием контекста, то есть результирующий контекст одной подзадачи может служить начальным

контекстом другой подзадачи. Такое разделение упрощает постановку задачи и ее решение, что в итоге сказывается на качестве программного продукта.

Предлагаемая контекстная система программирования позволяет приблизить процесс проектирования программных средств к парадигмам постановки и решения многих прикладных задач.

Двумя основными проблемами построения вычислительных систем для критически важных приложений, связанных с обработкой транзакций, управлением базами данных и обслуживанием телекоммуникаций, являются обеспечение высокой производительности и продолжительного функционирования систем. Наиболее эффективный способ достижения заданного уровня производительности - применение параллельных масштабируемых архитектур. Задача обеспечения продолжительного функционирования системы имеет три составляющих: надежность, готовность и удобство обслуживания. Все эти три составляющих предполагают, в первую очередь, борьбу с неисправностями системы, порождаемыми отказами и сбоями в ее работе.

Использование языка контекстного программирования позволяет сократить число ошибок, возникающих из-за семантического разрыва между применяемыми языковыми средствами и математической моделью системы управления безопасностью и, тем самым, повысить надежность разработки программ.

Использование контекстной технологии программирования и высокоуровневой формы представления мигрирующего кода позволяет сократить семантический разрыв между математической моделью и языковыми средствами, уменьшить число ошибок программирования, улучшить условия контроля за безопасностью кода, повысить готовность, скорость разработки и сопровождения программ.

Применение технологии исполнения мигрирующего кода в рамках предлагаемой контекстной технологии позволит сократить интенсивность обмена в системах параллельной обработки данных путем предварительного согласования (подгрузки, изменения) высокоуровневых протоколов для актуализации взаимодействия в терминах (понятиях), в которых формулируется решение задачи на клиентской стороне.

На основе компонентной технологии разработан объектно-ориентированный контекстный язык «Esse», являющийся мощным средством для разработки программных комплексов и систем, связанных с пониманием естественно-языковых текстов [28], планированием поведения, обучения, принятия решений, управления изменениями среды и др. Возможно применение на параллельных вычислительных и в суперкомпозитных системах.

Список литературы

1. Колегов А.В. Грамматика языка-посредника Эльюнди. Тирасполь: РИО ПГУ, 1998.
2. Липаев В.В. Направления развития методов и стандартов открытых систем // Информатика и вычислительная техника. Науч.-техн. сборник. Выпуск 1-2. М. 1995.
3. Лезер Н. Архитектура открытых распределенных систем: Модель OSF DCE // Открытые системы. № 3. 1993. С. 10-16.
4. Филинов Е.Н. Выбор и разработка концептуальной модели среды открытых систем // Открытые системы. № 6. 1995. С. 71-77.

Труды международной конференции «Параллельные вычисления и задачи управления» (PACO'2001). - Москва: Институт проблем управления, 2001. - С.(4)-121-139. Электрон. опт. диск (CD-ROM). - ISBN 5-201-09559-3.

5. Booch G. The End of Objects and the Last Programmer. - htpp://www.rational.com/pst/tech_papers.
6. Lewis T. Software Architectures: Divine plan or digital Darwinism // Computer. № 8. 1996. P. 13-15.
7. Canceled Software Development Project costs Billions // Computer. № 8. 1995. P. 94.
8. Lewis T. The Next 10,0002 Years. Part 2 // Computer. № 5. 1996. P. 78-86
9. Аджиев В. Объектная ориентация: философия и футурология // Открытые системы.
10. Meyer B. Object Technology: The Conceptual Perspective // Computer. № 1. 1996. P. 86-88
11. Gurus share insights on objects // Computer. № 6. 1996. P. 95-98
12. Booch G. Objects Solutions. Addison-Wesley, 1996
13. Henderson-Sellers B., Graham I. OPEN: Toward Method Convergence? // Computer. № 4. 1996. P. 86-89.
14. Вирт Н. Долой жирные программы // Открытые системы. № 6. 1996.
15. Богатырев Р. Java и Juice: дуэль технологий?! // Компьютерра, № 34. 1996. С. 30-33.
16. Yourdon E. Java, the Web and Software Development // Computer. № 8. 1996. P. 25-30.
17. Общая характеристика програкммного обеспечения информационных технологий. - http://www.uspu.ru/homepages/prepod/svn/prog/prog.htm.
18. Бен-Ари М. Языки программирования. Практический сравнительный анализ. – М.: Мир, 2000.
19. Зайцев С.С., Кравиунов М.И., Романов С.В. Сервис открытых информационно-вычислительных сетей: Справочник. М.: Радио и связь, 1990. [ITU-T (CCITT), Rec. X.200-X.219, 1988].
20. Филинов Е.Н. Выбор и разработка концептуальной модели среды открытых систем // Открытые системы № 6. 1995. С. 71-77.
21. Lewis T. Software Architectures: Divine plan or digital Darwinism // Computer. N 8. 1996. P. 13-15.
22. Canceled Software Development Project costs Billions // Computer. N 8. 1995. P. 94.
23. Иосенкин В.Я. Объектно-ориентированный Forth // Материалы международной научно-практической конференции «Региональные особенности развития машино- и приборостроения, информационных технологий, проблемы и опыт подготовки кадров». - Тирасполь, 2001. С. 132-134.
24. Иосенкин В. Я., Выхованец В.С. Технология контекстного программирования в телекоммуникационных системах // Труды второй международной научно-практической конференции «Современные информационные и электронные технологии». – Одесса, 2001. С. 118-119.
25. Липаев В.В. Отладка сложных программ. - М.: Энергоатомиздат, 1993.
26. Рейуорд-Смит В. Дж. Теория формальных языков. – М.: Радио и связь, 1988.
27. Выхованец В.С. Контекстная технология программирования // Труды IV международной научно-технической конференции по телекоммуникациям (Телеком-99), 14-17 сентября 1999, Одесса, С. 116-120.
28. Иосенкин В. Я., Выхованец В.С. Контекстное программирование в моделировании // Материалы международной научно-практической конференции «Моделирование. Теория, методы и средства». Часть 7. – Новочеркасск, 2001. С. 49-51.