Ретроспектива: 11.03.1994 - 11.03.2025

Алексей Недоря, Март 2025

Много (11,111) лет тому назад: 11 марта 1994 года, 14:30

Расширяемая переносимая система программирования, основанная на биязыковом подходе

05.13.11 - математическое и программное обеспечение вычислительный систем и машин

Диссертация на соискания ученой степени Кандидата физико-математических наук

Научный руководитель:
д.ф.-м.н. И.В. Поттосин

Новосибирск - 1994

Оппонент: д.ф.-м.н. А.Н. Терехов

Глава 1. Определение РПС и выбор языка реализации
- 1.1. Основные требования к языку
- 1.2. Сравнение языков Модула-3, Оберон, Оберон-2, C++

Глава 2. Семейство Modula-2/Oberon-2 компиляторов и трансляторов

Глава 3. Расширяемые системы на примере системы Оберон

Глава 4. Система Мифрил

Приложение А. Модели ООП
Приложение В. Аксиоматика "хороших" систем

Power of Context: есть ли жизнь после Кроноса?

- Куда двигаться?
- Как использовать сделанное на Кроносе?

1990: Новосибирск
- Знакомство с Никлаусом Виртом
- Modula-to-C (M2C), Steve Collins, RTA

1991: Zurich
- Oberon, Ceres, аспиранты Вирта
- компилятор OP2 (R. Creiler)
- редактор Write (C. Szyperski)

1993: London
- Extacy (X2C) - транслятор Модулы-2 и Оберон-2
- Oberon Guidelines

Новосибирск:
БЕТА и СОКРАТ

Расширяемые переносимые системы

Расширяемость:

- ядро системы определяет набор базовых понятий;
- этот набор понятий может быть расширен без изменения ядра (и без перекомпиляции);
- новые понятия могут быть определены на базе уже определенных понятий;
- отсутствуют различия между базовыми и вновь определенными понятиями;

Переносимость:

- все машинно-зависимые и системно-зависимые части РПС текстуально выделены;
- объем таких частей существенно меньше объема системы;
- система может быть настроена таким образом, чтобы (почти) полностью использовать ресурсы конкретной аппаратуры и ОС.

Надежность:

- использование языка программирования, обеспечивающего достаточный уровень надежности

Расширяемость - слабое понятие, любая программная система расширяема. Нет переиспользования.

Требования к языку программирования: все же ООП

… не использовать термин ОО по следующим причинам:

- несмотря на существование формальных моделей ОО, терминология ООП часто используется за рамками этих формальных моделей…
- термин ООП часто ассоциируется с программированием на языке SmallTalk или C++;
- вполне возможно, что существуют (или будут существовать) **другие модели**, более пригодные для реализации РПС.

… в модели Мейера считается существенным отождествление класса и модуля. С нашей точки зрения такое отождествление приводит к слиянию двух важных структурирующих механизмов:

- модулей – позволяющих объединить семантически связанные компоненты,
- и классов, выражающих связи по наследованию.

ООП не пригоден, надо идти дальше.

Сравнение языков программирования

C++ не подходит … так как в нем не реализованы:

- разделение концепции модуля и класса;
- сильная статическая типизация;
- динамическая типизация;
- сборка мусора.

Модула-3 поддерживает исключения, финализацию, параллелизм и весьма сложный механизм раздельной компиляции, **требующий этапа связывания всей программы**, что является абсолютно недопустимым для РПС.

Оберон-2
Некоторой проблемой является отсутствие низкоуровневых средств в этом языке. … трудоемкость переноса может быть существенно уменьшена, если система содержит еще и язык программирования низкого уровня. … заметим, что **Модула-2** обладает очень важным свойством: отсутствие собственной динамической поддержки.

Отсюда биязыковая система.

Любопытно, что выбор, в основном, делался для Мифрила, а компиляторы (исторически) были написаны на Модуле-2.

На следующем этапе (XDS) компиляторы тоже были переписаны на Обероне.

Схема трансляции

… реализовать **многоязыковую транслирующую систему**, позволяющую для N языков и для M целевых машин реализовать N фаз анализа, транслирующих некоторой язык программирования в промежуточное представление, и M фаз синтеза, транслирующих промежуточное представление в коды конкретной ЭВМ, и тем самым существенно сократить затраты на реализацию (с N*M до **N+M**). Такой общий подход принят в системе **Бета**.

Отрицательный результат экспериментов с многоязыковыми системами (не только с системой Бета) является отражением того факта, что проблема в целом неразрешима (или пока неразрешима) и для достижения успеха необходимо пойти на некоторые ограничения.

Примером удачной системы **N->1** является система **TopSpeed**, в которой реализованы языки Модула-2, Паскаль, C, C++ для IBM PC, а примером удачной системы **1->M** является Оберон-компилятор **OP2**, который в настоящее время перенесен на такие существенно различные архитектуры, как SPARC, MC 680x0, MIPS, RS/6000 и i80386.

… **успех** таких систем обусловлен тем, что в обоих случаях существует **определенность на одном конце системы**. … приходим к отображению 1+e -> M, где e определяет количество дополнительных конструкций, необходимых для реализации второго входного языка (e << 1).

До начала работы над LLVM (2000) еще 6 лет…

XDS перестал был биязыковым достаточно скоро:

- SL1
- диалект Паскаля
- далее Java

Мысль о семейства языков записана в 2018 г.

Мысль об идеальной XDS-LLVM для семейства языков еще где-то бродит…

Промежуточное представление

Процедурный интерфейс (mx)

… проблемы: для генерации хорошего кода необходим анализ достаточно большого фрагмента текста (например, для распределения регистров). Таким образом генерация вынуждена (почти) всегда восстанавливать из вызовов структуру текста.

Промежуточный язык

… фаза анализа строит текст на некотором промежуточном языке, а фаза синтеза выполняет разбор текста, а затем выполняет генерацию кода. Недостатки: фаза синтеза должна повторять (упрощенный) анализ, значительное время тратится на запись и чтение текстов на промежуточном языке. построение структуры, должно выполняться в каждом генераторе.

Промежуточная структура в памяти

… промежуточная структура есть синтаксическое дерево единицы компиляции. Внутреннее представление образовано узлами четырех видов: NODE (Узел), OBJECT (Объект), STRUCT (Структура) и VALUE (Значение).

Термин AST не используется.

Тривиль компилятор:

Узел

- Тип
- Описание
- Оператор
- Выражение

Забавные факты

C генерация: По иронии судьбы, транслятор в ANSI C разрабатывался на машине, на которой нет ANSI C компилятора. Проверка полученных текстов на инструментальной машине была невозможна, и первым большим тестом для транслятора стал перенос самого себя в среду MS-DOS. Время переноса на очередную платформу измеряется несколькими часами и, как правило, определяется **скоростью чтения флоппи-диска**.

Симфайлы: оригинальный алгоритм канонизации, который записывает объекты в порядке инфиксного обхода сбалансированного дерева. При чтении симфайла не делается никаких дополнительных действий. Дерево видимости, построенное по симфайлу, является сбалансированным.

Размер кода: размер фазы анализа для языка Модула-2 - 5345 строк, а для языка Оберон-2 - 5130. Общая часть составляет 3595 строк. Генераторы: Кронос - 4600, ANSI C - 4100, x86 - 11000.

Тривиль (C99): 10500 строк

Мифрил: прививка от ООП

… система Mithril будет использоваться как база для разработки переносимого прикладного ПО. Для того чтобы упростить разработку прикладного ПО необходимо решить несколько важных задач:
1) Необходимо создать комфортную обстановку для программиста (языки, отладка, поддержка проекта);
2) Система должна включать набор средств для создания пользовательского интерфейса (окна, мышь, конструктор интерфейсов);
3) Система должна поддерживать одинаковую обстановку на разных платформах (т.е. на разных машинах и под разными ОС).

- А. Денисов и О. Шатохин: генерация для x86
- А. Хапугин: сборка мусора, загрузчик
- А. Никитин: Мифрил - оконная система, графика

Да, но увы…

Приложение А. Модели ООП и языки CLOS и Оберон-2

Скучно

Приложение В. Аксиоматика "хороших" систем

Полувековой мечтой всего программистского сообщества (да не дрогнет рука автора, выписывающего эти слова) является создание такой среды программирования, которая будет устраивать почти всех и работать почти на всех машинах. Как и любая другая N-вековая мечта (для N>=0.5), эта мечта с трудом поддается осуществлению и любое приближение к реализации ее должно рассматриваться как важный шаг вперед.

Переведем обе составляющие этой мечты на более профессиональный язык. Будем называть систему "хорошей", если она асимптотически осуществляет мечту.

Теорема 1. Любая завершенная система не является "хорошей".

> *Хорошей иллюстрацией к данному утверждению может служить Вселенная, созданная Богом за 6 дней.*

Следствие 1. Разработка "хорошей" системы должна продолжаться всегда.

Берегитесь лжепророков, которые приходят к вам в овечьей одежде, а внутри суть волки хищные.

Мат. 7,15

Выводы

- То, что было самым важным, оказалось полезным как прививка, но не применимым.
- Переносимость - сейчас является обязательным свойством.
- Расширяемость слабое свойство, нужны заменяемость, сокращаемость, переиспользование, …
- Создание одной или нескольких “хороших” систем не имеет смысла, нужна технология изготовления. Поэтому, далее Сборочное программирование, а теперь Архитектурное программирование.

EOT