Компилятор языка Zonnon: архитектура, интеграция, технология

Научно-практическая конференция
по программированию
Москва, 15-17 июня 2003

Евгений Зуев,
Institute for Computer Systems,
ETH Zürich
zueff@inf.ethz.ch
www.inf.ethz.ch/~zueff/

Содержание

- Задачи проекта.
- Компилятор Zonnon для .NET.
- Технология: пакет CCI.
- Интеграция в VS.
- Скриншоты и/или демонстрация.

Задачи проекта

- Реализовать базовую версию компилятора для платформы Microsoft .NET:
  - подмножество входного языка;
  - генерация MSIL-кода в полном формате;
  - режим командной строки.

Ноябрь 2002

- Обеспечить полную интеграцию компилятора со средой разработки MS Visual Studio .NET:
  - текстовый редактор;
  - фоновая компиляция;
  - управление проектами;
  - отладчик etc.

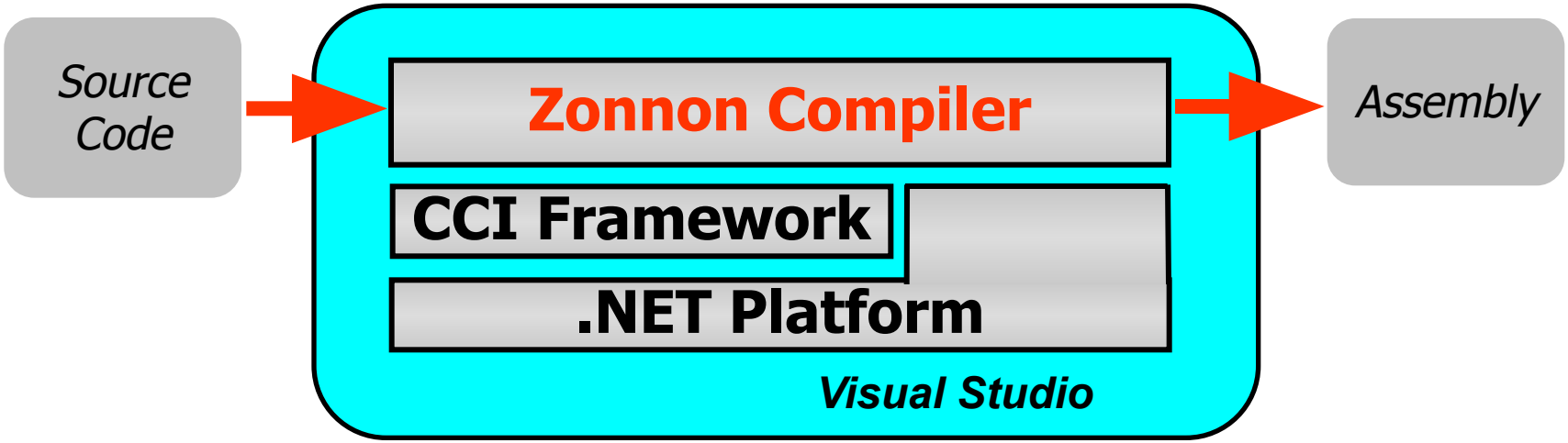
Середина 2003

?

- Выполнить “раскрутку” (bootstrapping) полного компилятора на языке Zonnon.

Компилятор Zonnon

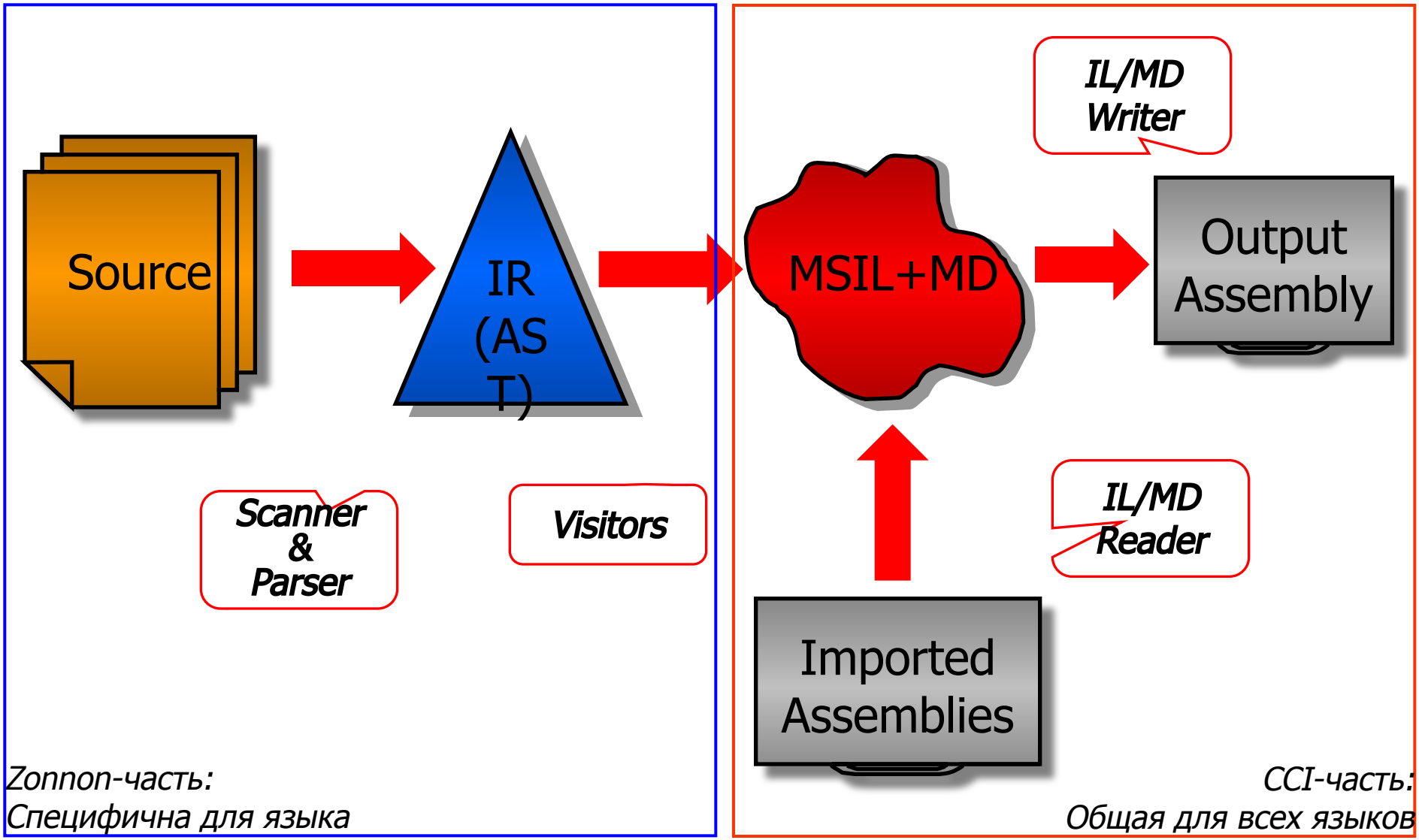
- Реализован для платформы .NET
- Генерирует стандартную сборку (Assembly)
- Реализован с использованием пакета CCI
- Интегрирован в среду MS Visual Studio .NET
- Создан в ETH Zürich, Switzerland



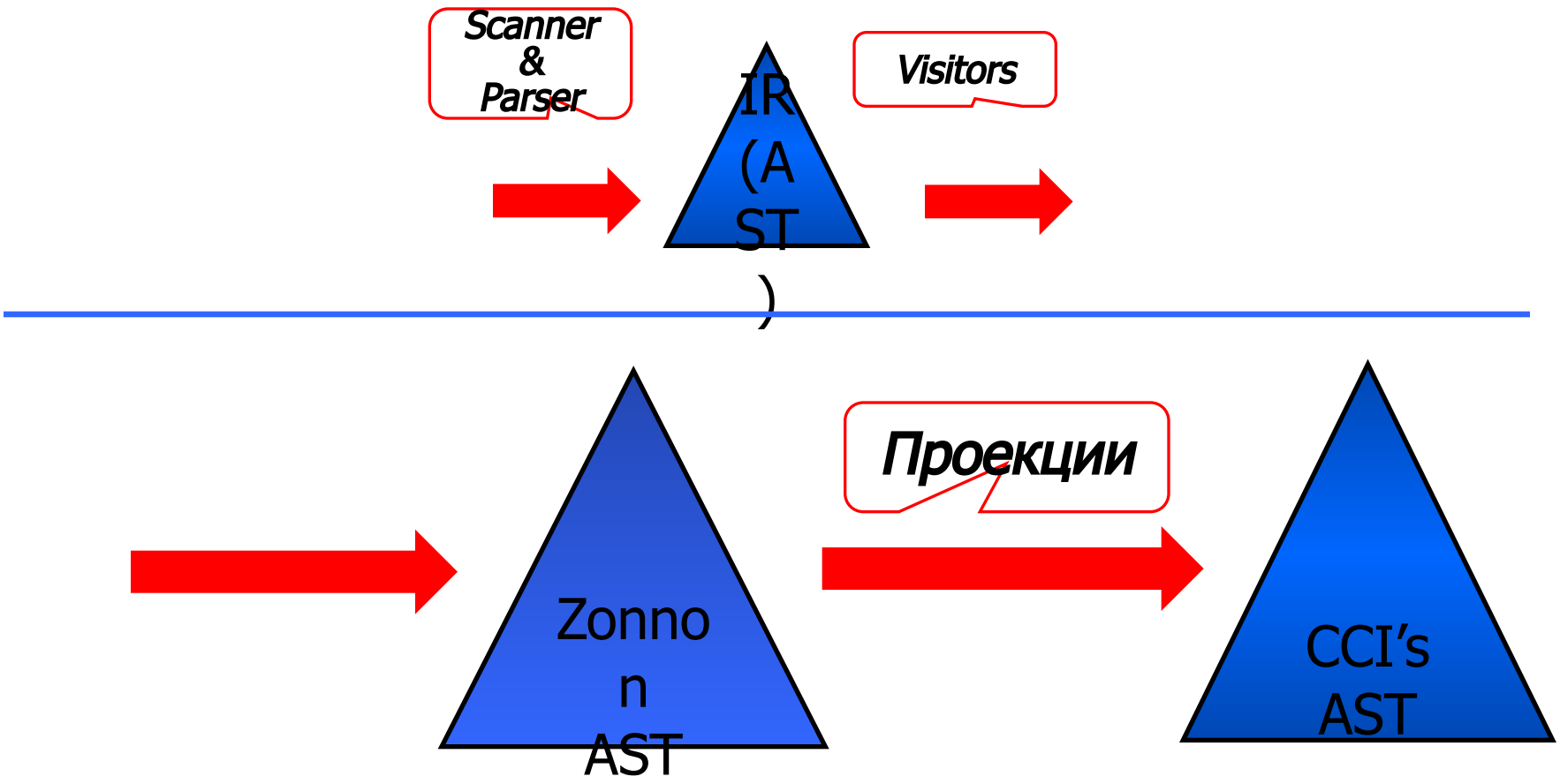
Компиляторы для .NET: возможные подходы

- Непосредственная («ручная») компиляция в MSIL/Metadata (нет примеров) или в язык ассемблера MSIL («toy compilers»).
- Использование «родного» для .NET языка (напр. C#) в качестве промежуточного (Eiffel)
- Генерация MSIL-кода средствами низкого уровня из пространств имен `System.Reflection` и `System.Reflection.Emit` (Component Pascal, авт. John Gough; Oberon.NET)
- Высокоуровневая поддержка - **CCI**: построение дерева программы с (полу)автоматической генерацией IL+MD (ASML, Zonnon for .NET).

Модель компиляции Zonnon (1)



Модель компиляции Zonnon (2)



Реализуется семантическая специфика Zonnon;

Выполняется сериализация «интерфейсной» части AST для последующей статической комплексации в собственных терминах языка

Проекции (mappings): отображение специфических свойств Zonnon на семантически эквивалентные структуры .NET

Проекции Zonnon -> .NET

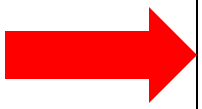
- DEFINITION
  абстрактный интерфейс; `interface`
- IMPLEMENTATION
  реализация интерфейса по умолчанию; единица агрегации; `class`
- OBJECT
  шаблон (класс), реализующий интерфейс; возможно, «активный» объект; `sealed class`
- MODULE
  контейнер ресурсов; класс, управляемый системой; `class with static members`

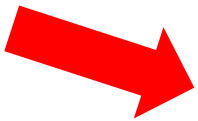
Проекции Zonnon->.NET: Definitions & Implementations

Zonnon

```
DEFINITION D;
  TYPE e = (a, b);
  VAR x: T;
  PROCEDURE f (t:T);
  PROCEDURE g ():T;
END D;

IMPLEMENTATION D;
  VAR y: T;
  PROCEDURE f (t: T);
  BEGIN x := t; y := t
  END f;
END D;
```





C#

```
interface D_i {
  T x { get; set; }
  void f(T t); T g (); };

internal class D_b {
  private T x_b;
  public enum e = (a, b);
  public T x {
    get { return x_b };
    set { x_b = ... }}};
public class D_c: D_b {
  T y;
  void f(T t)
  { x_b = t; y = t; } };
```

Проекции Zonnon->.NET: Objects

Zonnon

```
OBJECT X IMPLEMENTS D;
  IMPORT D;
  VAR y : T;
  PROCEDURE g (): T IMPLEMENTS D.g;
  BEGIN y := D.x; RETURN D.y
  END g;
END X;
```

C#

Проекция с отдельным helper-классом

```
public sealed class X: D_I
{ D_c d; T y;
  public override T g() {
    y = d.x; return d.y; } }
```

Проекция с базовым классом

```
public sealed class X: D_i, D_c d;
{ T y;
  public override T g() {
  y = x_b; return y_b }}
```

Проекции Zonnon->.NET: Active Objects

Zonnon **C#**

Исходная конструкция	Проекция на C#
`ACTIVITY S END`	Метод: `void body() { S };` Поле: `Thread thread;`

Создание активного объекта (неявный запуск «активности»)	`x.thread = new Thread(`
	`  new ThreadStart(body))`
	`x.thread.Start()`

`AWAIT cond;`	`while ( !cond )` `{ Monitor.Wait(this); }`

`BEGIN { LOCKED }`
`S`
`END`
	`Monitor.Enter(this);`
	`S;`
	`Monitor.PulseAll(this);`
	`Monitor.Exit(this);`

CCI: Основа Zonnon-компилятора

- **CCI** = **C**ommon **C**ompiler **I**nfrastructure.
- CCI – набор ресурсов (классов), предоставляющих поддержку реализации компиляторов и других языковых инструментов для .NET
  - **Реализация** компиляторов;
  - **Интеграция** компиляторов.
- Концептуально, CCI является частью .NET Framework SDK.
- Спроектирован и реализован в Microsoft; автор - **Herman Venter.**

CCI: сценарии использования

- Интеграция в VS.NET существующих (“не-CCI”) компиляторов.
- Интеграция в VS.NET компиляторов, полностью реализованных на основе CCI
- Расширение существующих .NET-языков и компиляторов (C#, VB etc.).
- Создание процессоров для посткомпиляционной обработки.
- Учебные компиляторы!

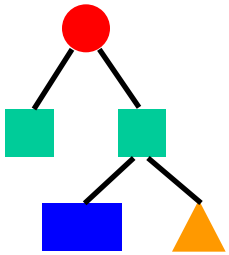
CCI: Три проблемы

- (***Общая***)
  Разработка компилятора – непростая задача; **интеграция** компилятора в среду программирования – целый спектр дополнительных проблем.
- (***CCI***)
  CCI реализует **существенно отличный от традиционного** подход к процессу компиляции.
- (***Техническая***)
  CCI имеет объемный и нетривиальный интерфейс, набор правил и «контрактов».

Общие принципы использования CCI

- Все сервисы CCI представлены в виде **классов**. Чтобы воспользоваться этими сервисами, необходимо определить собственные классы, производные от классов CCI.
- В производных классах необходимо обеспечить реализацию некоторых **абстрактных методов** классов-прототипов (они образуют «унифицированный интерфейс» с окружением).
- Производные классы содержат функциональность, реализующую **собственную семантику** компилятора.

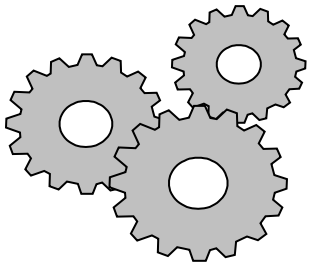
Компоненты CCI



Intermediate Representation (IR) –

Развитая иерархия C#-классов, представляющих наиболее общие и типичные понятия современных ЯП

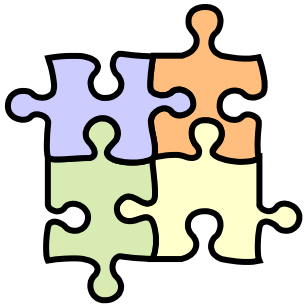
`System.Compiler.dll`



Преобразователи ("Visitors") –

Набор классов, реализующих последовательные преобразования IR ⇒ MSIL

`System.Compiler.Framework.dll`



Поддержка интеграции –

Совокупность классов и методов, обеспечивающих интеграцию в среду Visual Studio (дополнительная функциональность для редактирования, отладки, фоновой компиляции etc.)

IR: промежуточное представление (1)

```
Node                    Node
  Expression              Member
    UnaryExpression         TypeNode
    BinaryExpression          Class
    NaryExpression            DelegateNode
      MethodCall            EnumNode
      Indexer               Interface
    AssignmentExpression          . . .
    Literal                 TypeParameter
    Parameter               Pointer
      This                  Reference
  Statement               Event
    AssignmentStatement     Method
    If                  InstanceInitializer
    For                 StaticInitializer
    ForEach                 Field
    Continue                Property
    ExpressionStatement     Namespace
    VariableDeclaration     CompilationUnit
```

Часть дерева наследования IR

IR: промежуточное представление (2)

Пример:

```
public class If : Statement
{
    Expression condition;
    Block      falseBlock;
    Block      trueBlock;
      . . .
}

public class Block : Statement
{
    bool          hasLocals;
    StatementList statements;
      . . .
}
```

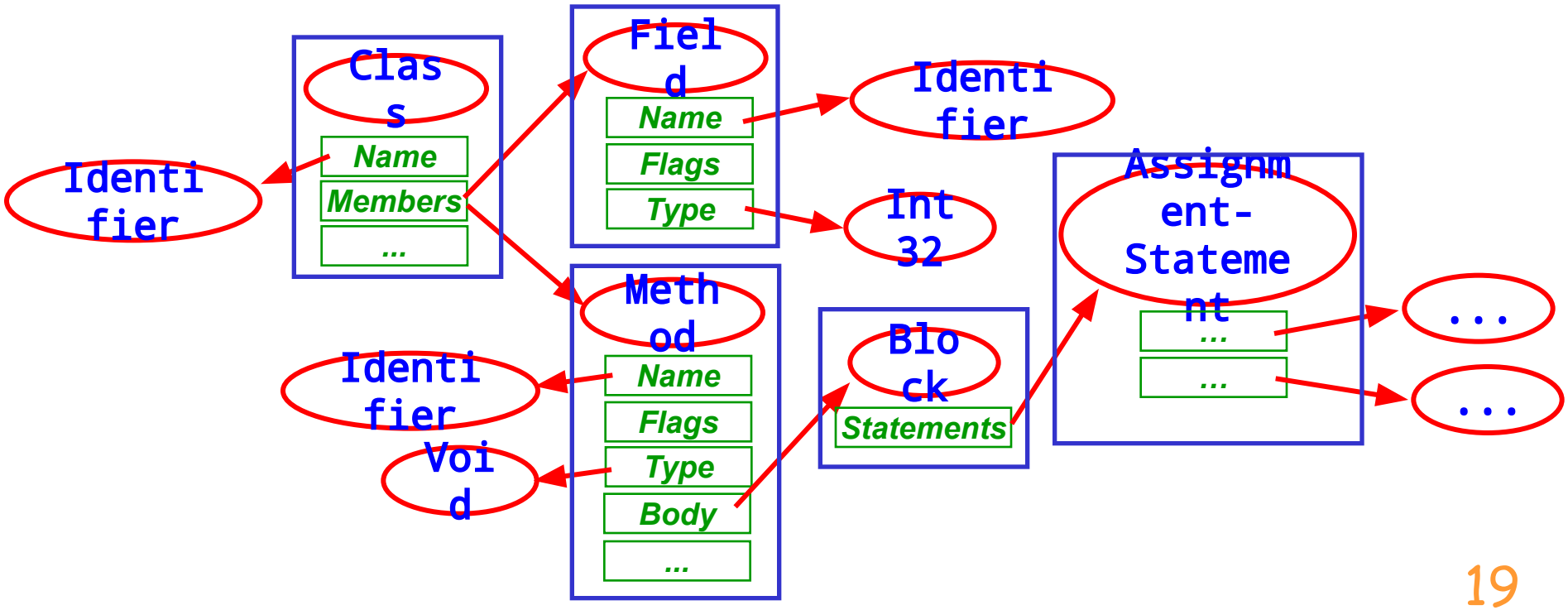
Характеристики IR:

- Весьма прямолинейный подход.
- IR почти полностью повторяет **иерархию понятий языка C#**.
- Включает поддержку некоторых языковых черт, **отсутствующих** в C#.
- Поддерживает некоторые **будущие** свойства C# (напр., generics).
- **Вывод**: архитектура IR достаточна для представления широкого спектра языков с традиционной парадигмой.

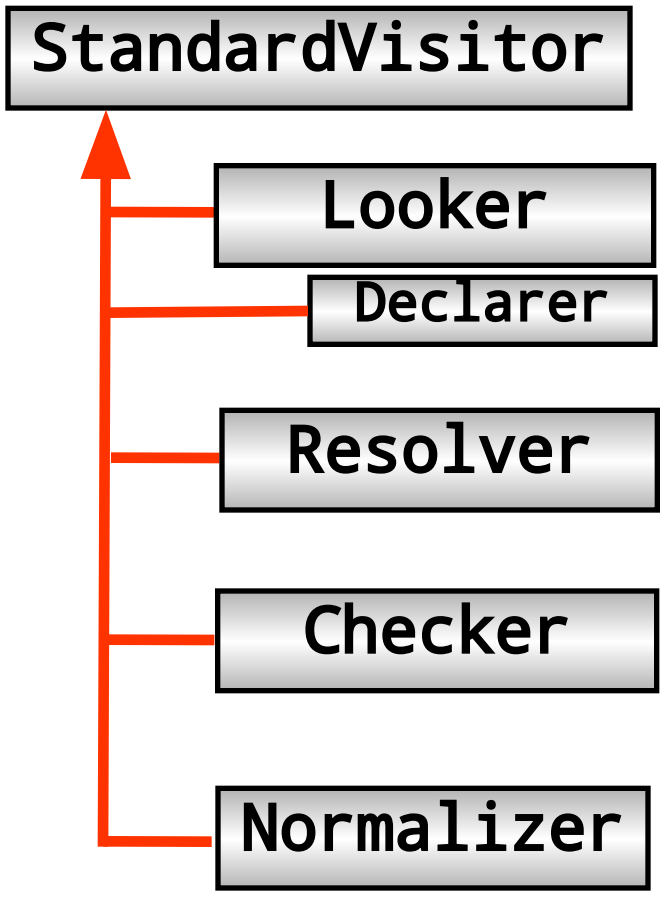
IR: промежуточное представление (3)

Пример: класс C#

```
public class C
{
    public int m1;
    public void f ( ) { m1 = 0; }
}
```



Система трансформаций IR в CCI



Каждый Visitor обходит дерево IR…

…**заменяя** узлы `Identifier` ссылками на сущности, которые обозначает идентификатор;

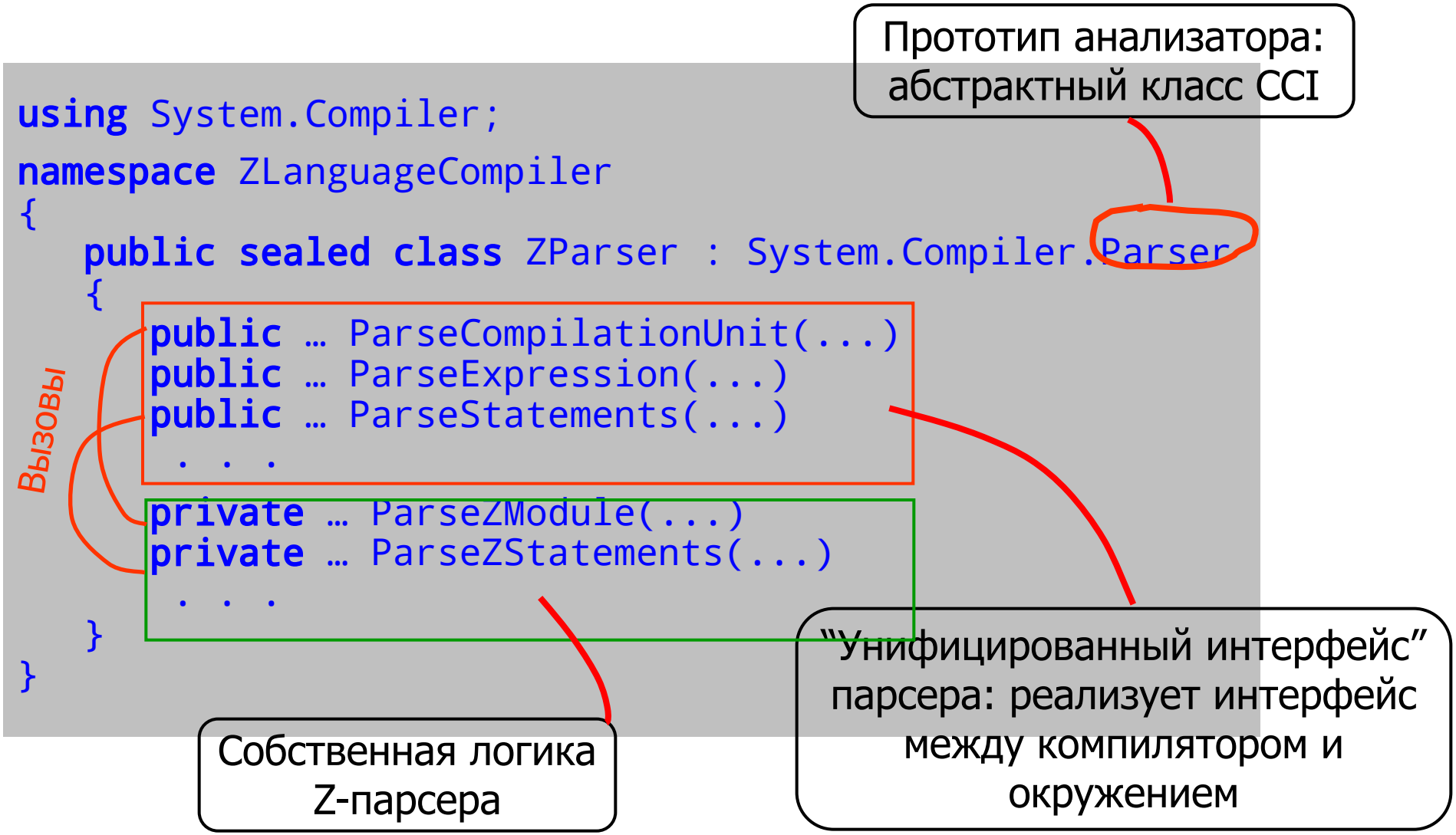
…**разрешая** случаи совместного использования **(overloading)** and **вычисляя типы** выражений;

…выполняя **семантические проверки**;

…готовя дерево к **сериализации** (генерации IL+MD).

- **Можно модифицировать стандартные Visitor'ы и/или**
- **Написать собственные**

Организация синтаксического анализа



Работа с IR: расширение Visitor'ов

Пример расширения Looker'а

```
using System.Compiler;

namespace ZLanguageCompiler
{
  public sealed class ZLooker : System.Compiler.Looker
  {
    public override Node Visit( Node node )
    {
      switch ( node.NodeType ) {
        case ZNodeType.NewStmt:
          return this.VisitNewStmt((NewStmt)node);
        default:
          return base.Visit(node);
      }
    }
    public Node VisitNewStmt( NewStmt node )
    {  /* Преобразование NewStmt в некоторый CCI-узел */ }
  }
}
```

Visitor-прототип: абстрактный класс CCI

Метод-"диспетчер"

Семантическая обработка узла

Обработка IR: Активация Visitor'ов

Общая схема работы с IR

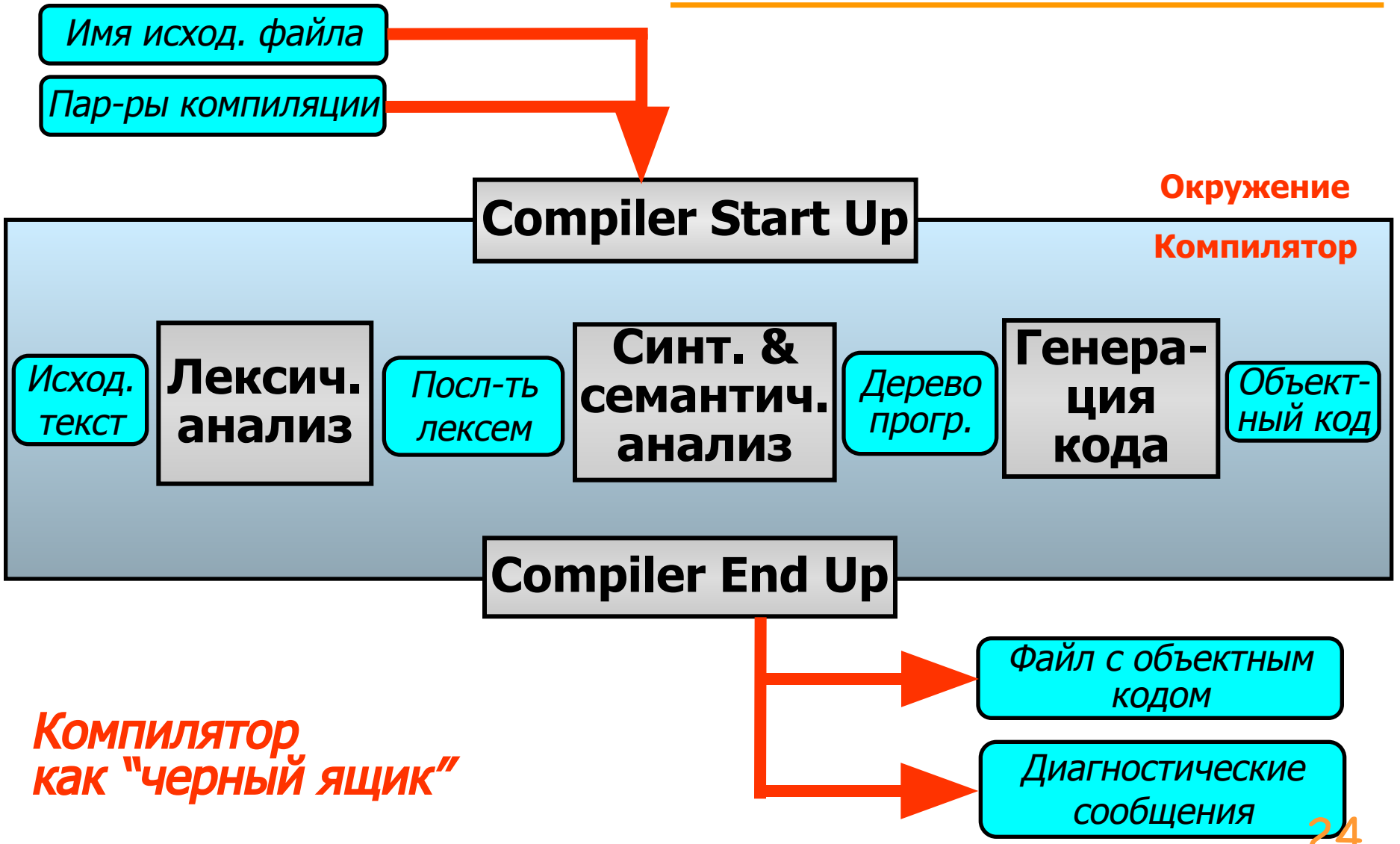
Прототип компилятора: абстрактный класс CCI

```
public class ZCompiler : System.Compiler.Compiler, ...
{
  . . .
  protected override void Compile ( CompilationUnit cu,
                                    Class           globalScope,
                                    ErrorNodeList   errors )
  {
    // Разрешение имен
    (new ZLooker(globalScope)).VisitCompilationUnit(cu);
    // Разрешение совм.использования и вычисление типов
    (new ZResolver()).VisitCompilationUnit(cu);
    // Семантические проверки; «исправление» дерева
    (new ZChecker(errors)).VisitCompilationUnit(cu);
    // Редукция дерева до узлов с предопред.отображением в MD+IL
    (new Normalizer().VisitCompilationUnit(cu);
  }
  . . .
}
```

Типы узлов IR

Запуск Visitor'ов

Архитектура компилятора: традиционный подход



Компилятор как "черный ящик"

Что подразумевается под интеграцией?

Компоненты среды Visual Studio

Менеджер проектов

Текстовый редактор

Семантическая поддержка (“Intellisense”)

Отладчик

Поведение, которое должен поддерживать компилятор

- Запуск компиляции и сборка проектов

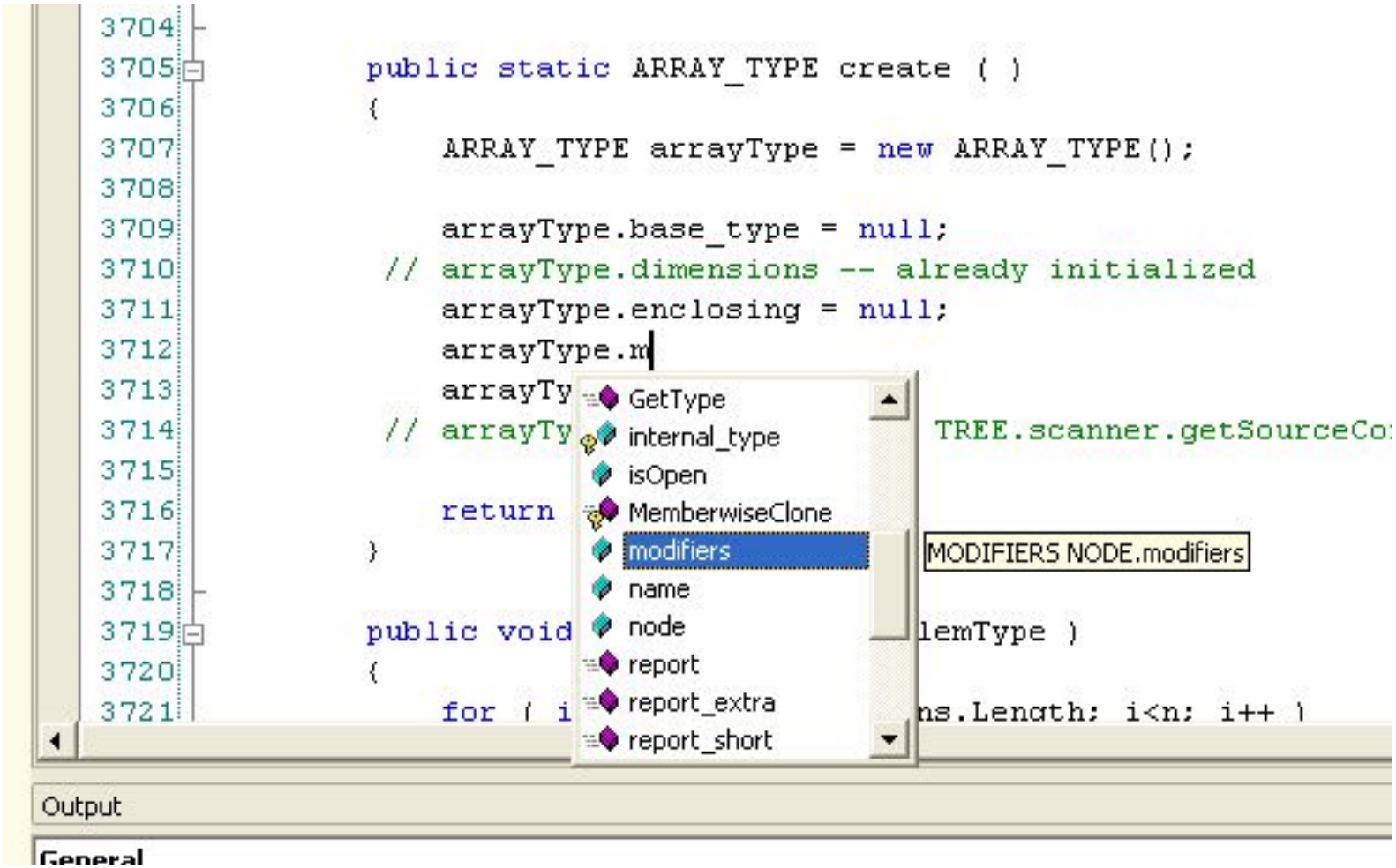
- Синтаксическая подсветка
- Автоматическое форматирование; структурн. проход по тексту { }
- Синтаксические проверки на фоне ввода текста
- «Плавающая» диагностика

- Вывод «содержимого» составного типа для переменной этого типа
- Вывод списка совместно-исполь-зуемых методов
- Вывод списка параметров

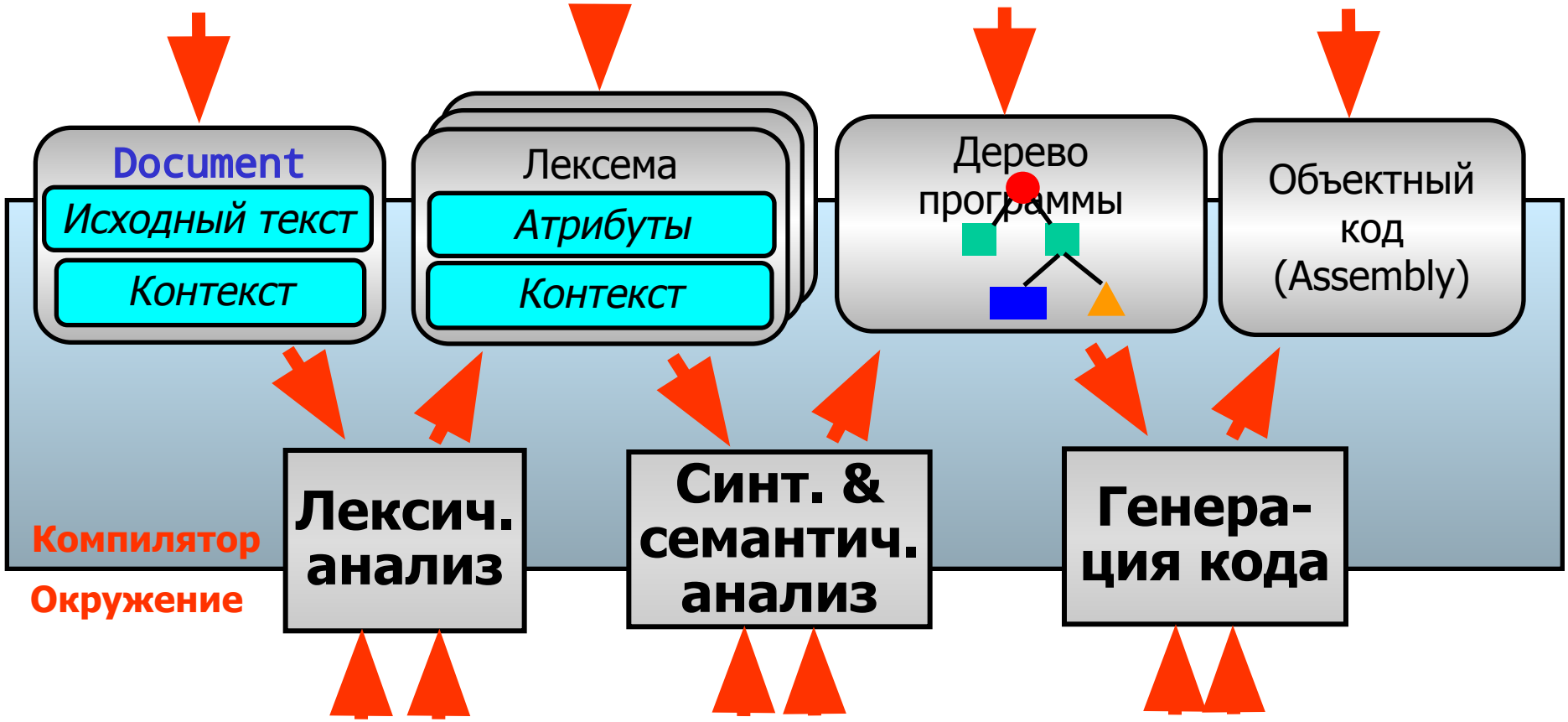
- Вычисление выражений
- Условные точки останова

Что подразумевается под интеграцией?

Пример “Intellisense”

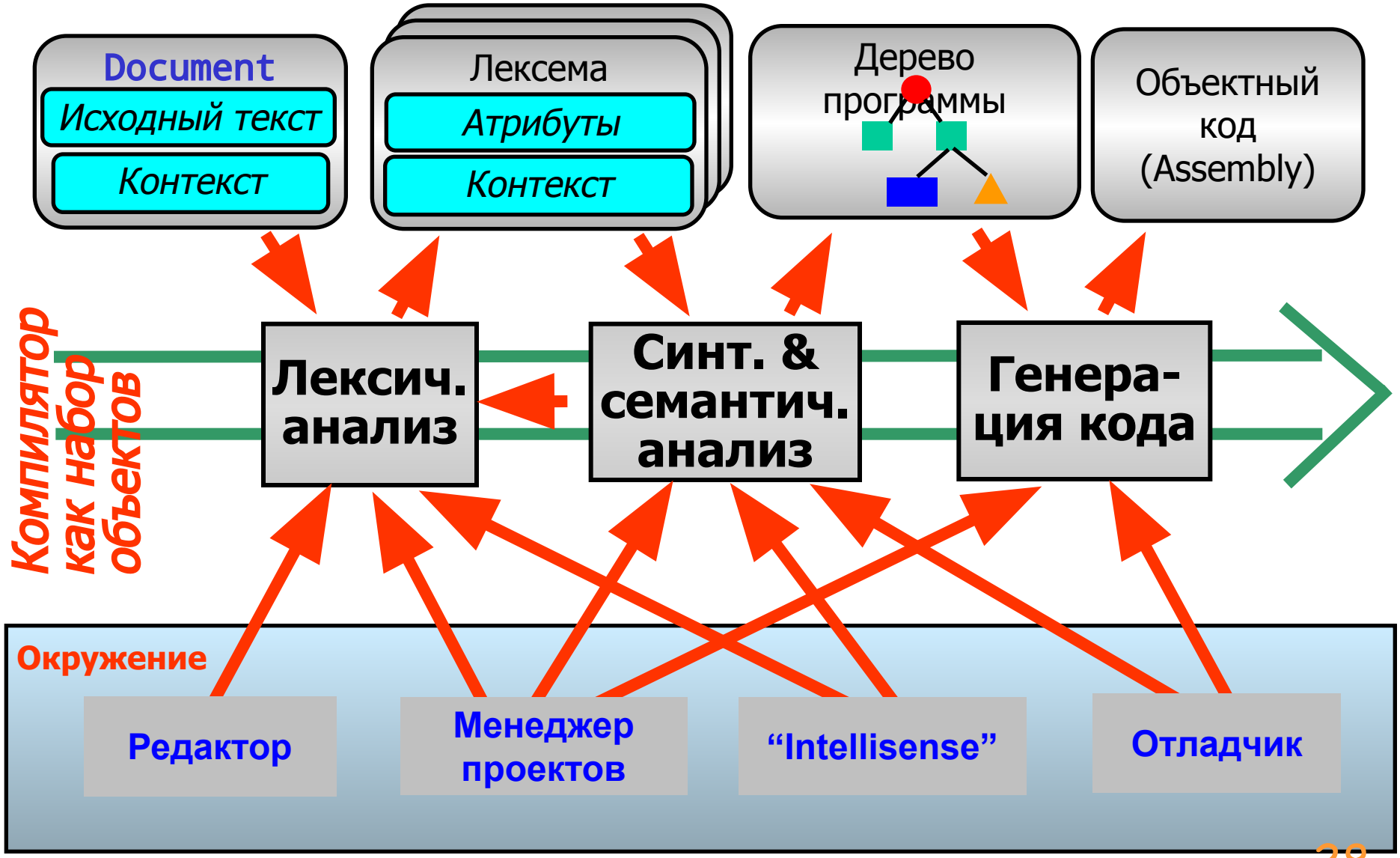


Архитектура CCI-компилятора (1)

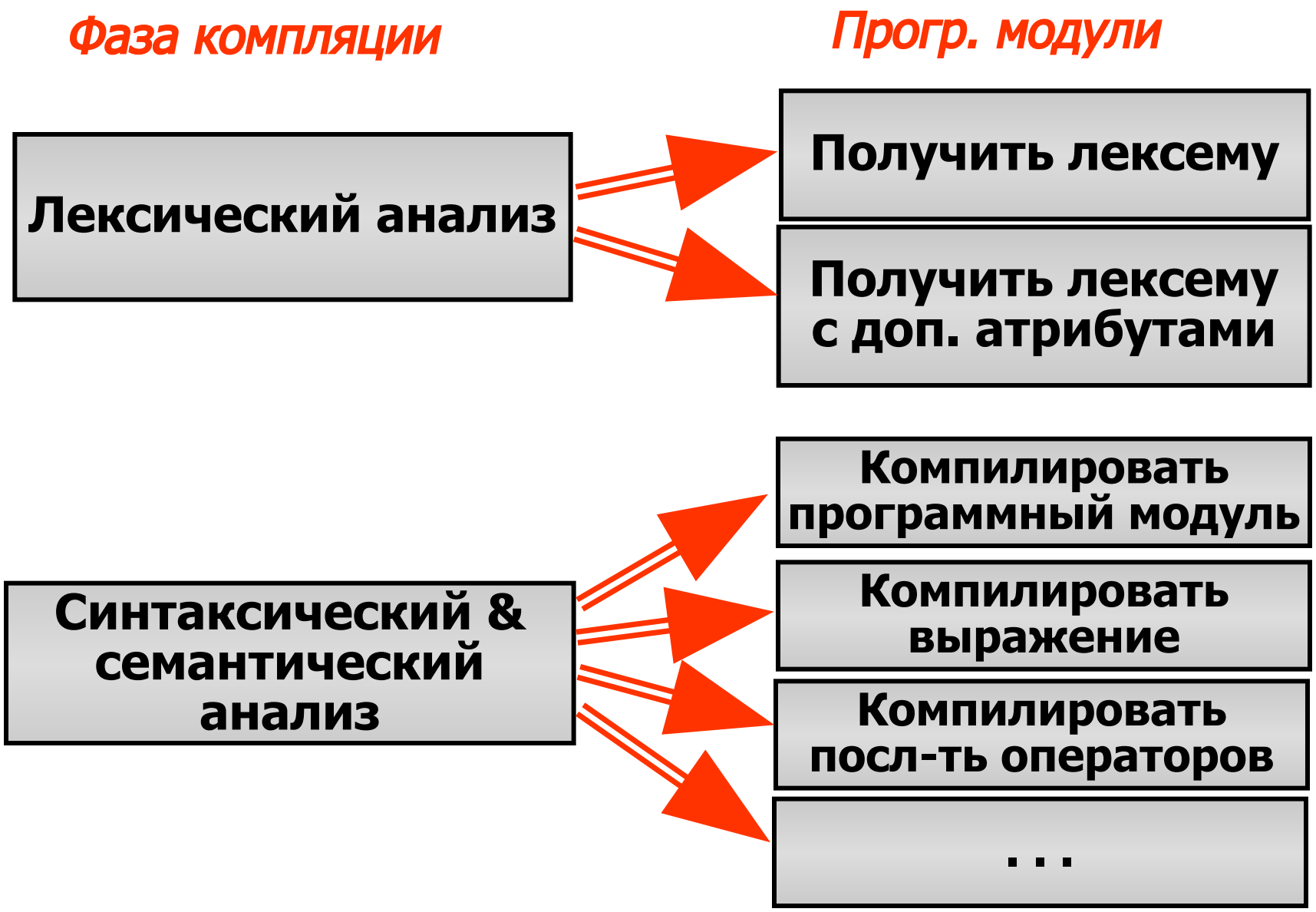


Компилятор как коллекция ресурсов

Архитектура CCI-компилятора (2)



Архитектура CCI-компилятора (3)

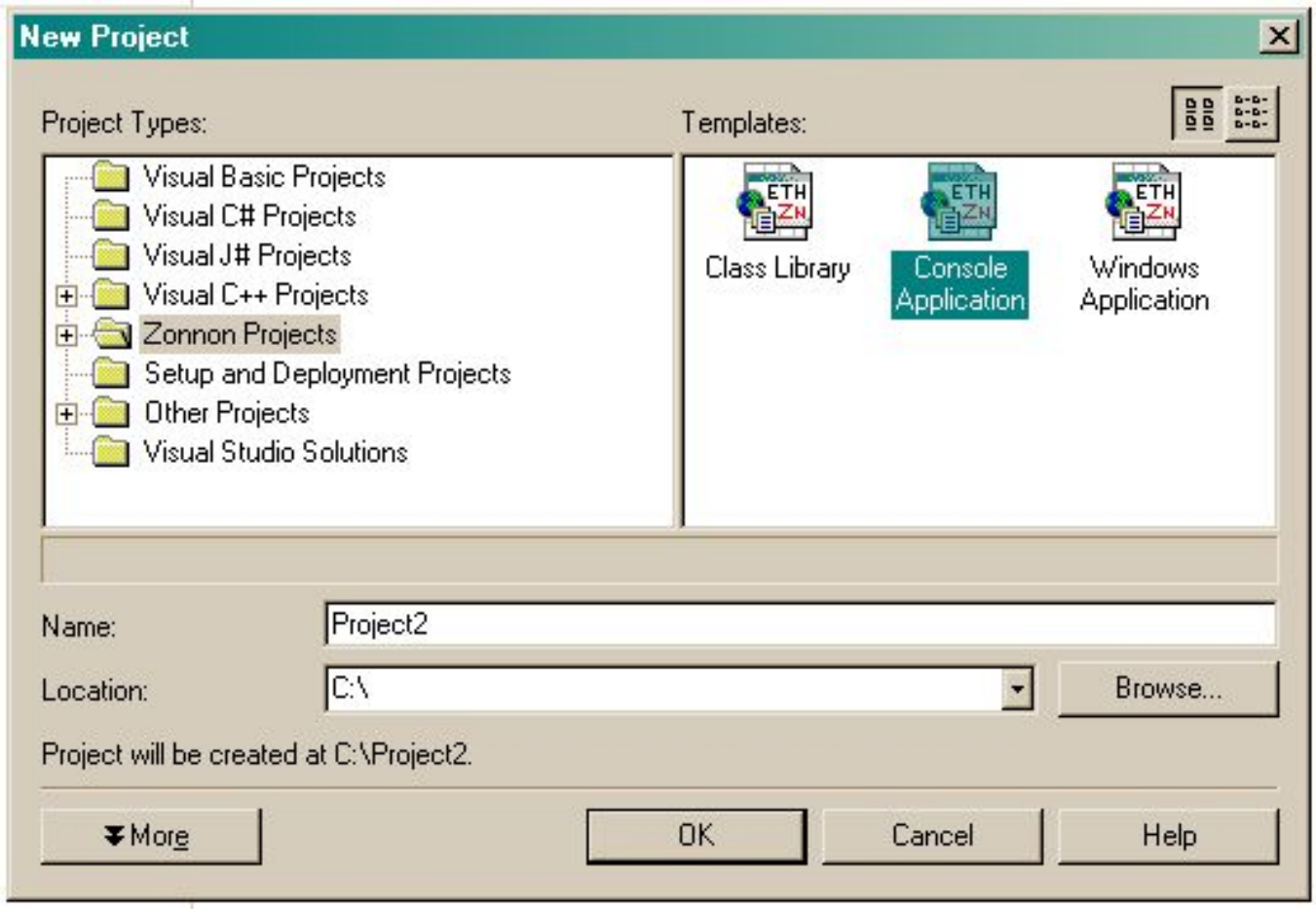


CCI: текущее состояние и статус

- Реализован почти полностью (неполная поддержка процесса отладки); не отлажен; не документирован.
- 12 июня 2003 CCI Toolkit был включен в MSDN Academic Alliance (инициатива для образовательных учреждений компьютерного профиля): www.msdnaa.net/cci
- Компилятор Zonnon – первый опыт использования CCI за пределами Microsoft.

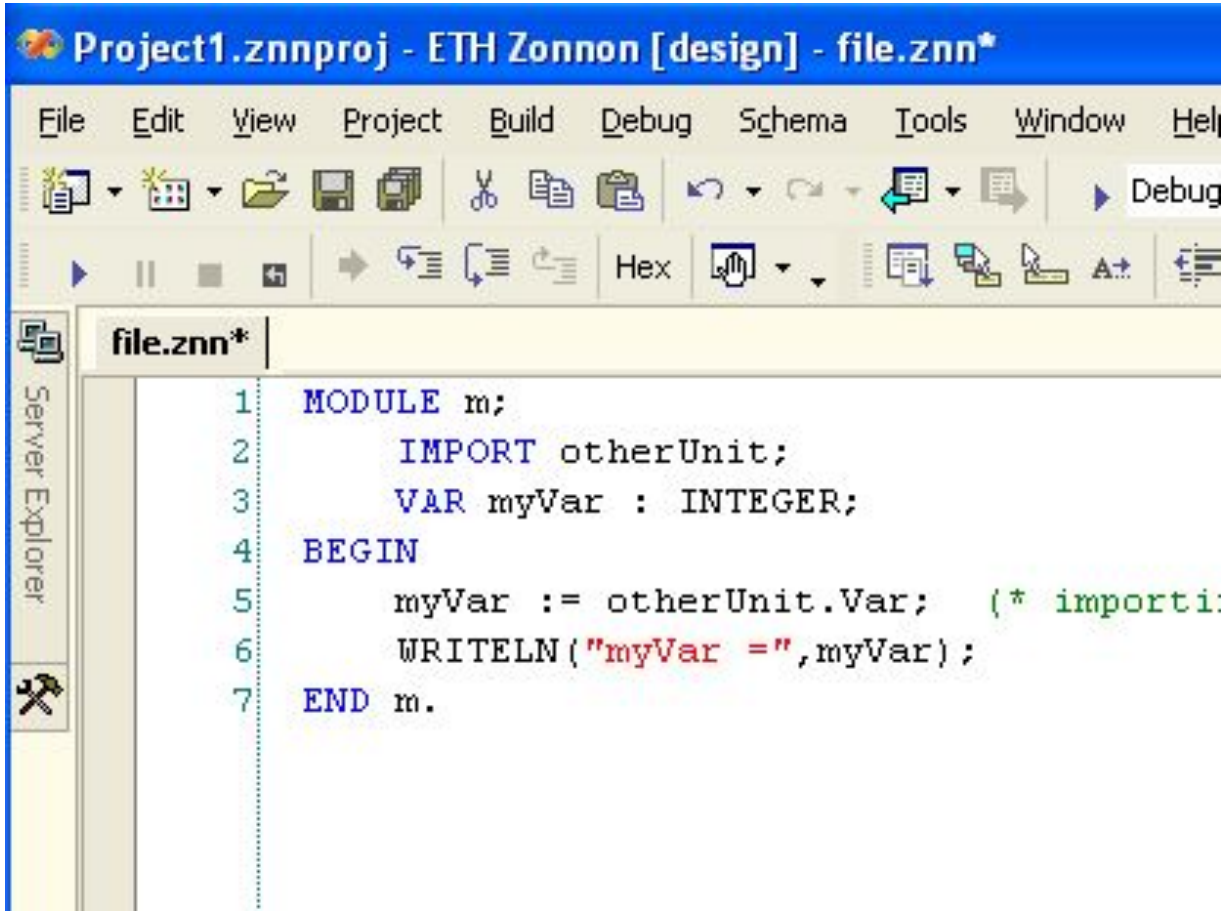
Интеграция Zonnon-компилятора в VS (1)

(Вместо демонстрации ☺)



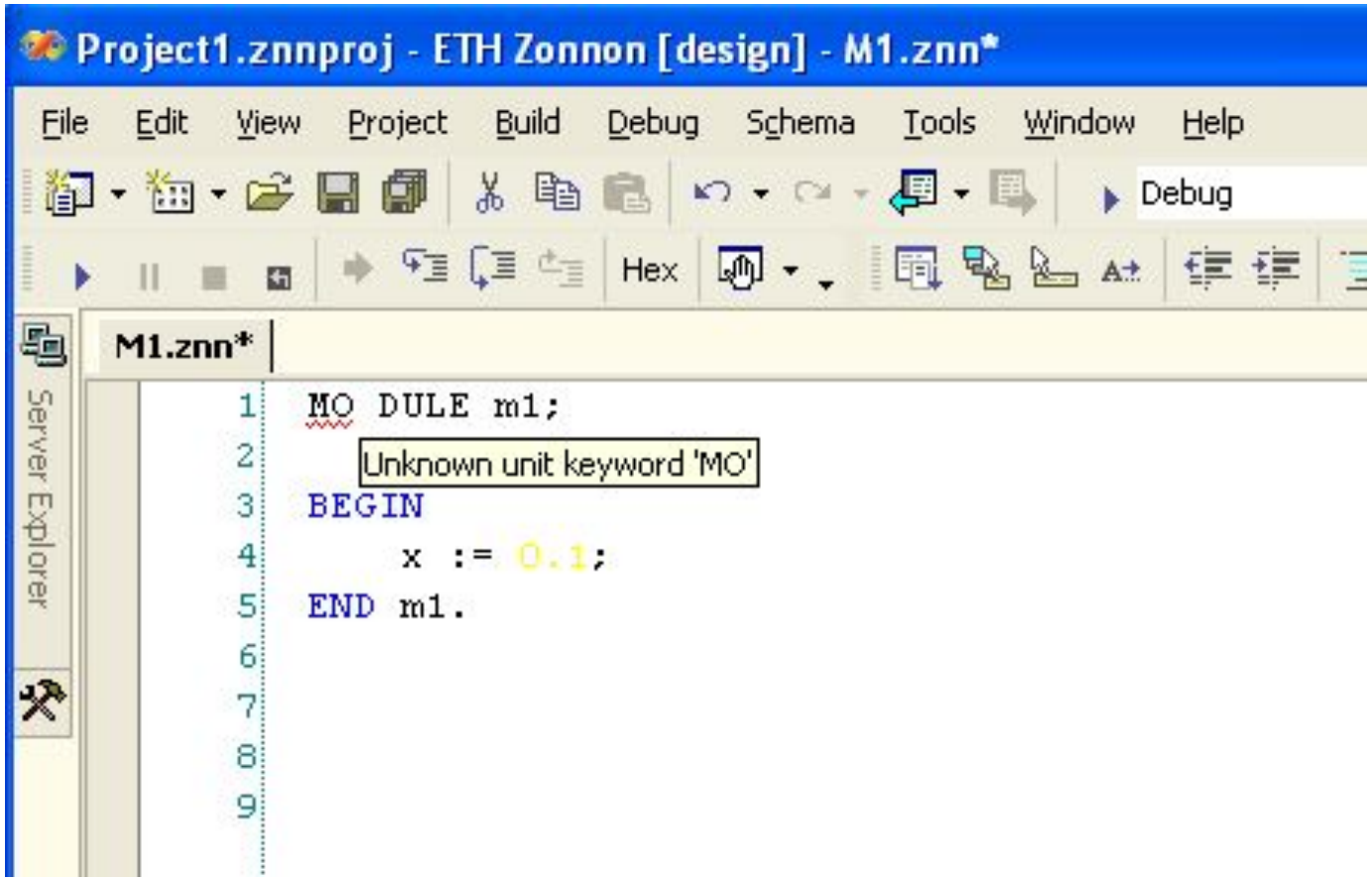
Интеграция Zonnon-компилятора в VS (2)

(Вместо демонстрации ☺)



Интеграция Zonnon-компилятора в VS (3)

(Вместо демонстрации ☺)



Интеграция Zonnon-компилятора в VS (4)

(Вместо демонстрации ☺)