

Компилятор языка Zonnon: архитектура, интеграция, технология

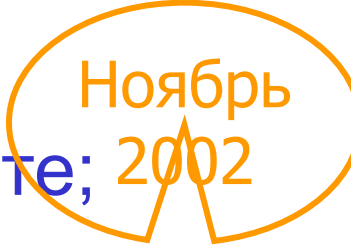
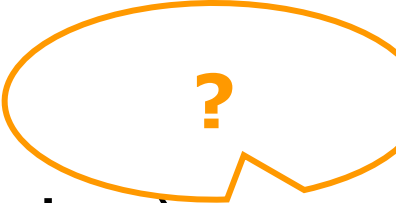

Научно-практическая конференция
по программированию
Москва, 15-17 июня 2003

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

Содержание

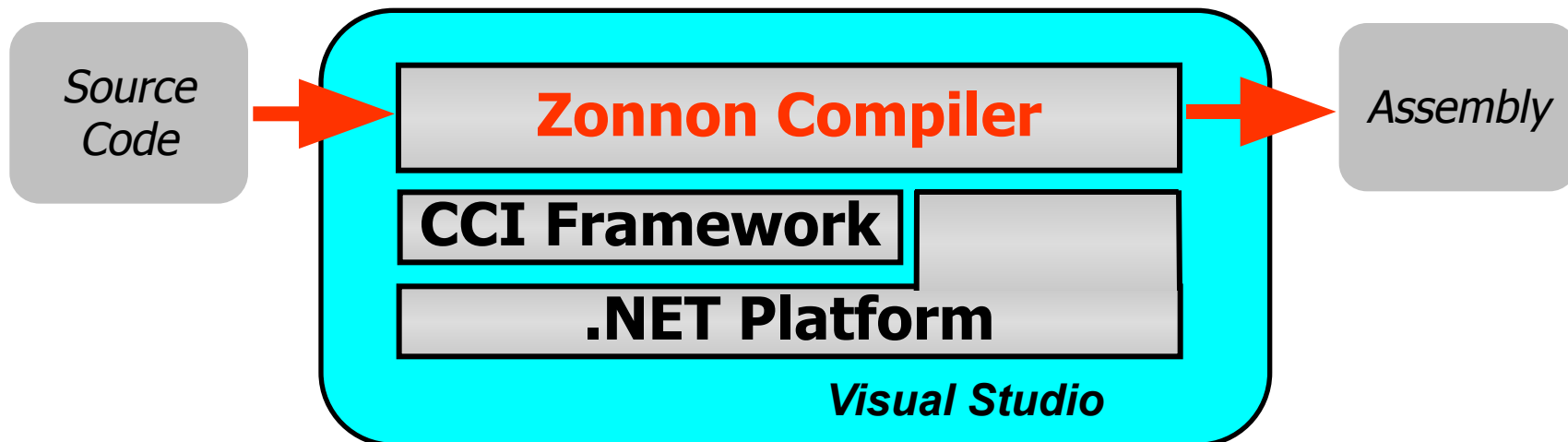
- Задачи проекта.
- Компилятор Zonnon для .NET.
- Технология: пакет CCI.
- Интеграция в VS.
- Скриншоты и/или демонстрация.

Задачи проекта

- Реализовать базовую версию компилятора для платформы Microsoft .NET:
 - подмножество входного языка;
 - генерация MSIL-кода в полном формате;
 - режим командной строки.
- Обеспечить полную интеграцию компилятора со средой разработки MS Visual Studio .NET:
 - текстовый редактор;
 - фоновая компиляция;
 - управление проектами;
 - отладчик etc.
- 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**

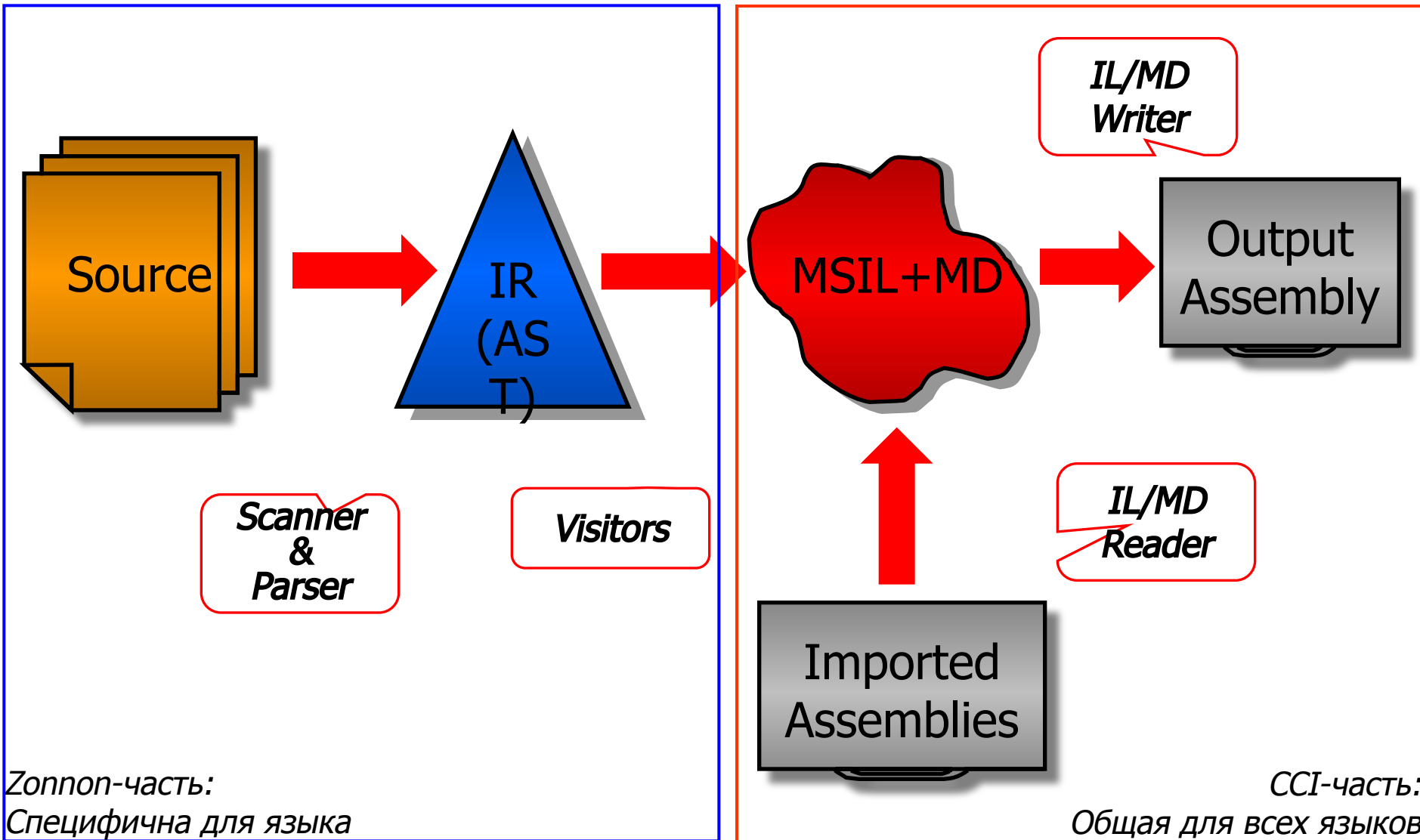


Компиляторы для .NET:

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**).

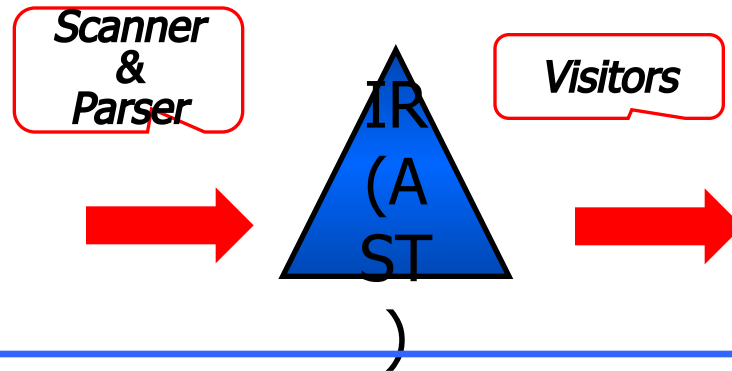
Модель компиляции Zonnon (1)



Zonnon-часть:
Специфична для языка

CCI-часть:
Общая для всех языков

Модель компиляции Zonnon (2)



Реализуется **семантическая специфика** Zonnon;
Выполняется **сериализация** «интерфейсной»
части AST для последующей статической
компиляции в собственных терминах языка

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

Проекция Zonnon -> .NET

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

Проекция 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; } };
```

Проекция 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 }}
```

Проекция Zonnon -> .NET: Active Objects

Zonnon

C#

Исходная конструкция	Проекция на C#
ACTIVITY S END	Метод: <code>void body() { S };</code> Поле: <code>Thread thread;</code>
Создание активного объекта (неявный запуск «активности»)	<code>x.thread = new Thread(new ThreadStart(body))</code> <code>x.thread.Start()</code>
AWAIT cond;	<code>while (!cond)</code> <code>{ Monitor.Wait(this); }</code>
BEGIN { LOCKED } S END	<code>Monitor.Enter(this);</code> <code>S;</code> <code>Monitor.PulseAll(this);</code> <code>Monitor.Exit(this);</code>

CCI: Основа Zonnon-компилятора

- **CCI = Common Compiler Infrastructure.**
- 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!

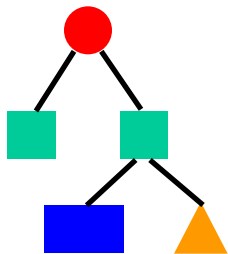
ССИ: Три проблемы

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

Общие принципы использования ССИ

- ◆ Все сервисы ССИ представлены в виде **классов**. Чтобы воспользоваться этими сервисами, необходимо определить собственные классы, производные от классов ССИ.
- ◆ 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`

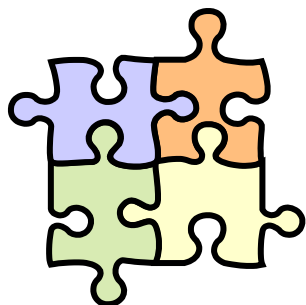
Преобразователи (“Visitors”) –

Набор классов, реализующих последовательные преобразования
IR \Rightarrow 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;
    . . .
}
```

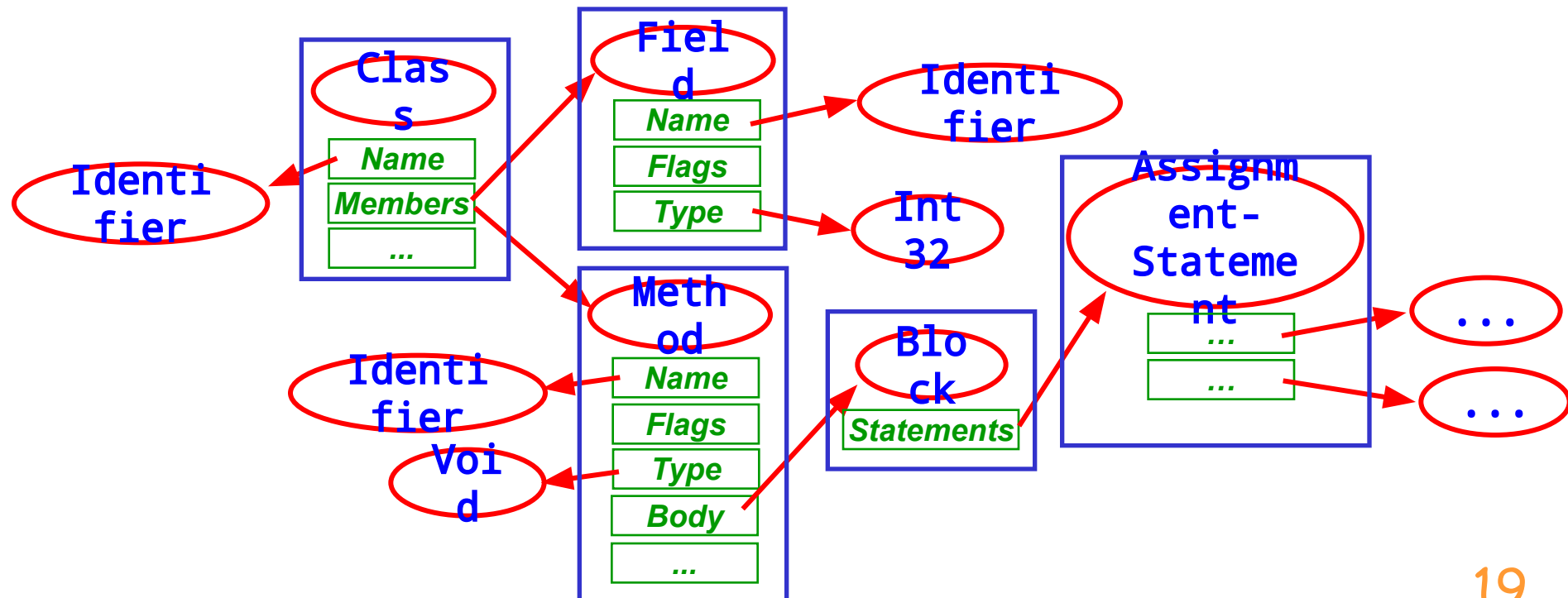
Характеристики IR:

- ◆ 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 достаточно для представления широкого спектра языков с традиционной парадигмой.

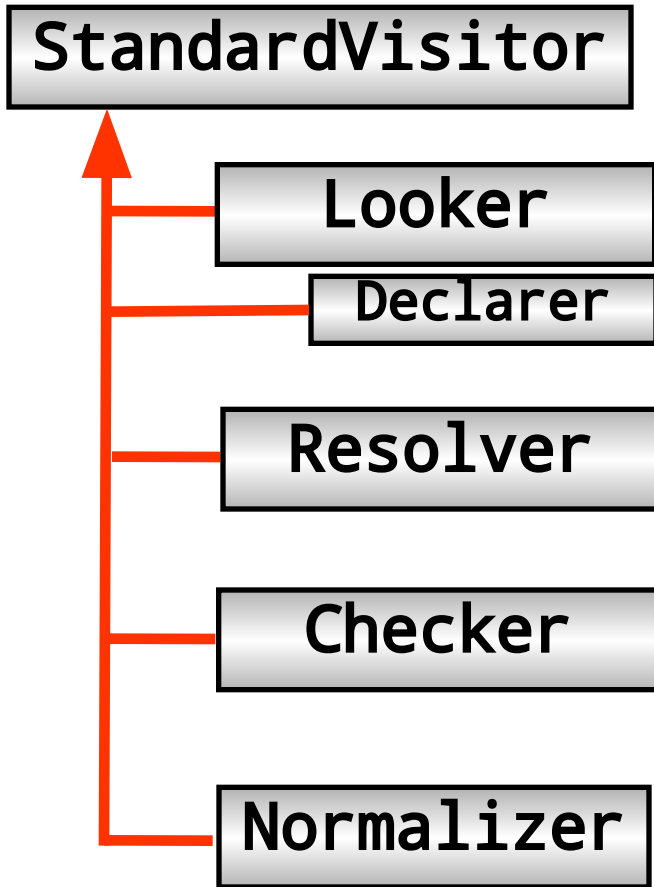
IR: промежуточное представление (3)

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

```
using System.Compiler;
namespace ZLanguageCompiler
{
    public sealed class ZParser : System.Compiler.Parser
    {
        public ... ParseCompilationUnit(...)
        public ... ParseExpression(...)
        public ... ParseStatements(...)
        . . .
        private ... ParseZModule(...)
        private ... ParseZStatements(...)
        . . .
    }
}
```

Вызовы

Прототип анализатора:
абстрактный класс CCI

Собственная логика
Z-парсера

“Унифицированный интерфейс”
парсера: реализует интерфейс
между компилятором и
окружением

Работа с IR: расширение Visitor'ов

Пример расширения Looker'a

```
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

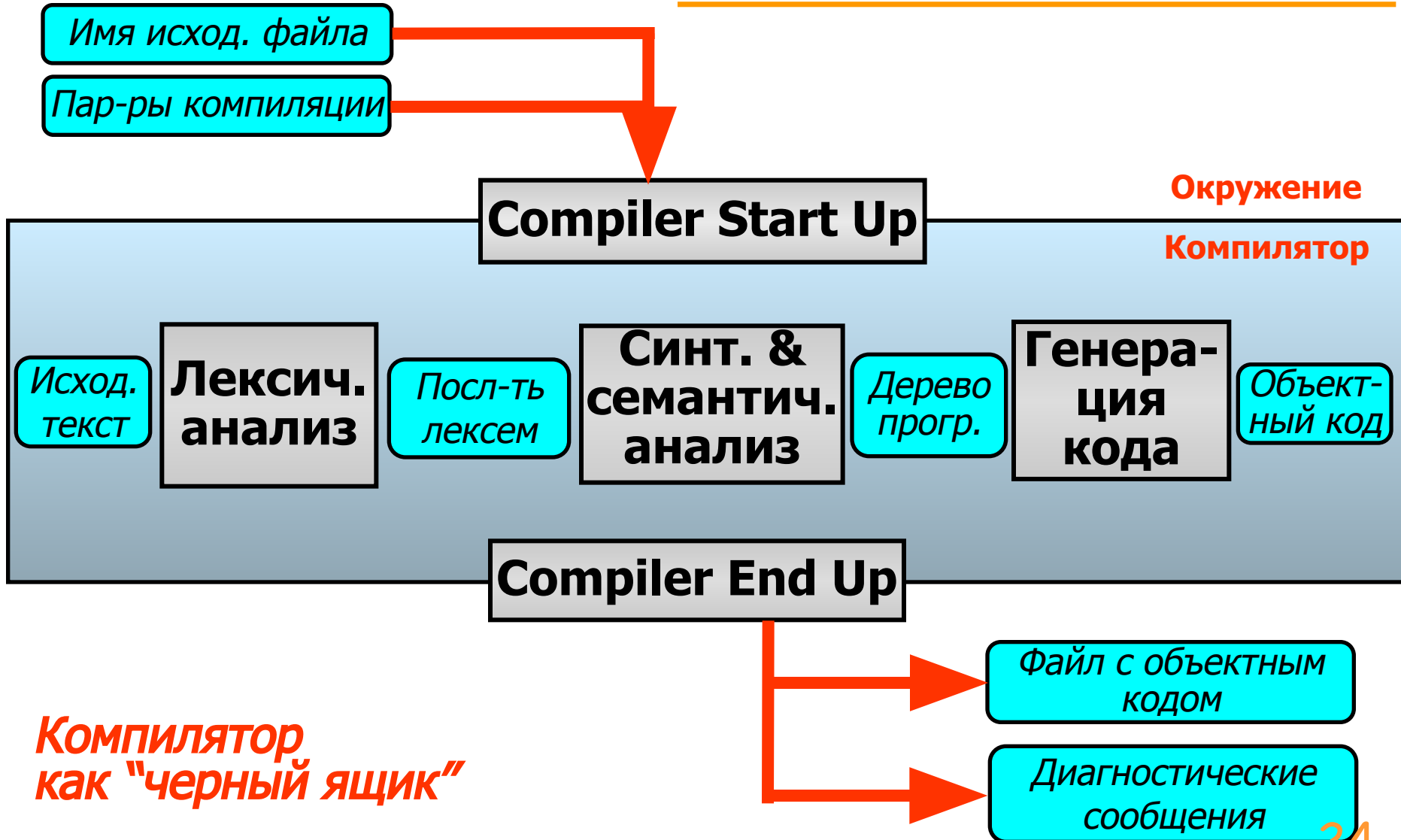
Прототип компилятора:
абстрактный класс CCI

```
public class ZCompiler : System.Compiler.Compiler, ...
{
    . . .
    protected override void Compile ( CompilationUnit cu,
                                     Class
                                     ErrorNodeList
                                     globalScope,
                                     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



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

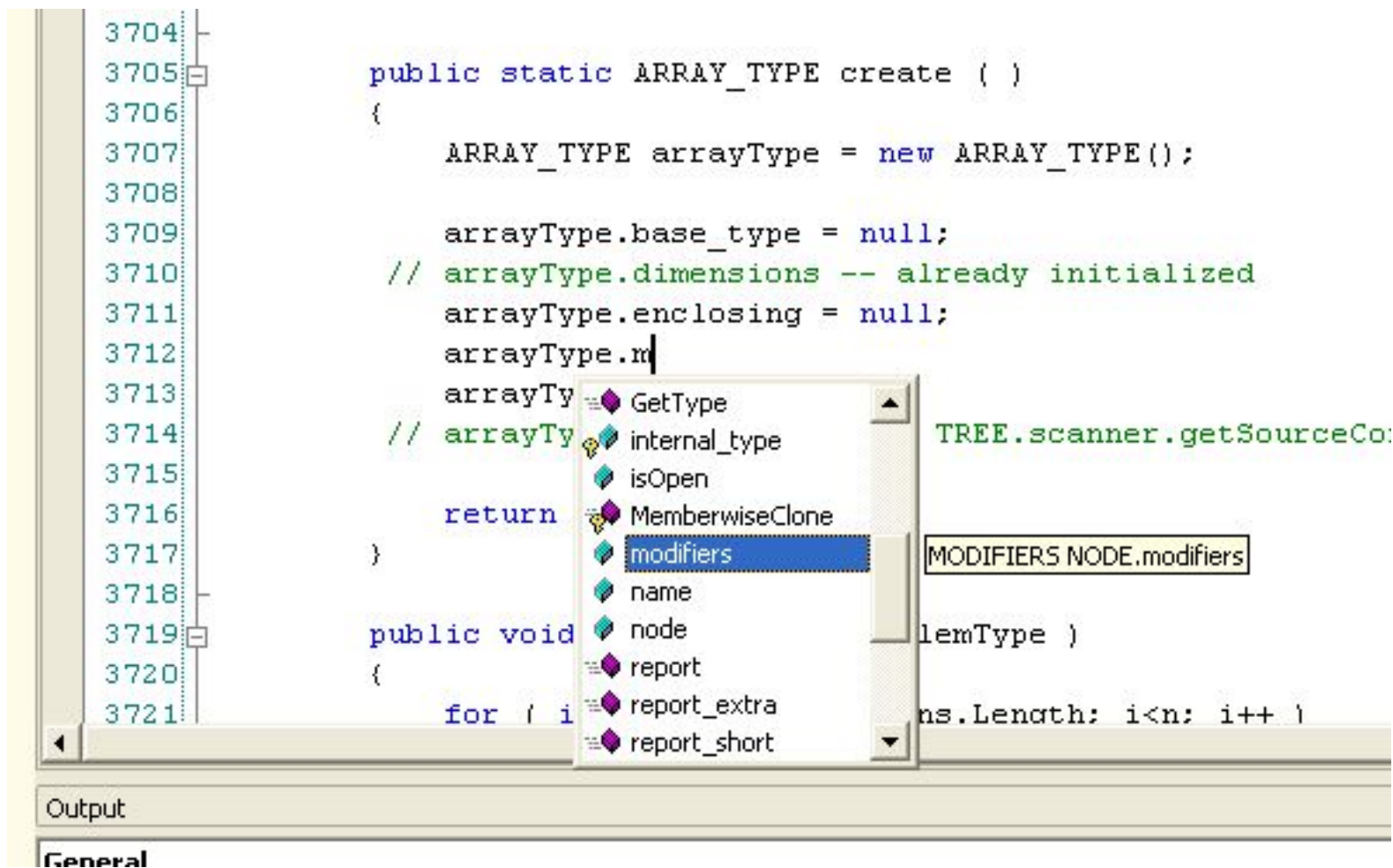
- Синтаксическая подсветка
- Автоматическое форматирование; структурн. проход по тексту {□}
- Синтаксические проверки на фоне ввода текста
- «Плавающая» диагностика

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

- Вычисление выражений
- Условные точки останова

Что подразумевается под интеграцией?

Пример “Intellisense”



The screenshot displays a code editor with the following C# code:

```
3704  
3705 public static ARRAY_TYPE create ( )  
3706 {  
3707     ARRAY_TYPE arrayType = new ARRAY_TYPE();  
3708  
3709     arrayType.base_type = null;  
3710     // arrayType.dimensions -- already initialized  
3711     arrayType.enclosing = null;  
3712     arrayType.m  
3713     arrayTy  
3714     // arrayTy  
3715  
3716     return  
3717 }  
3718  
3719 public void  
3720 {  
3721     for ( i
```

An Intellisense popup menu is visible over the code, listing the following members for the `arrayType` object:

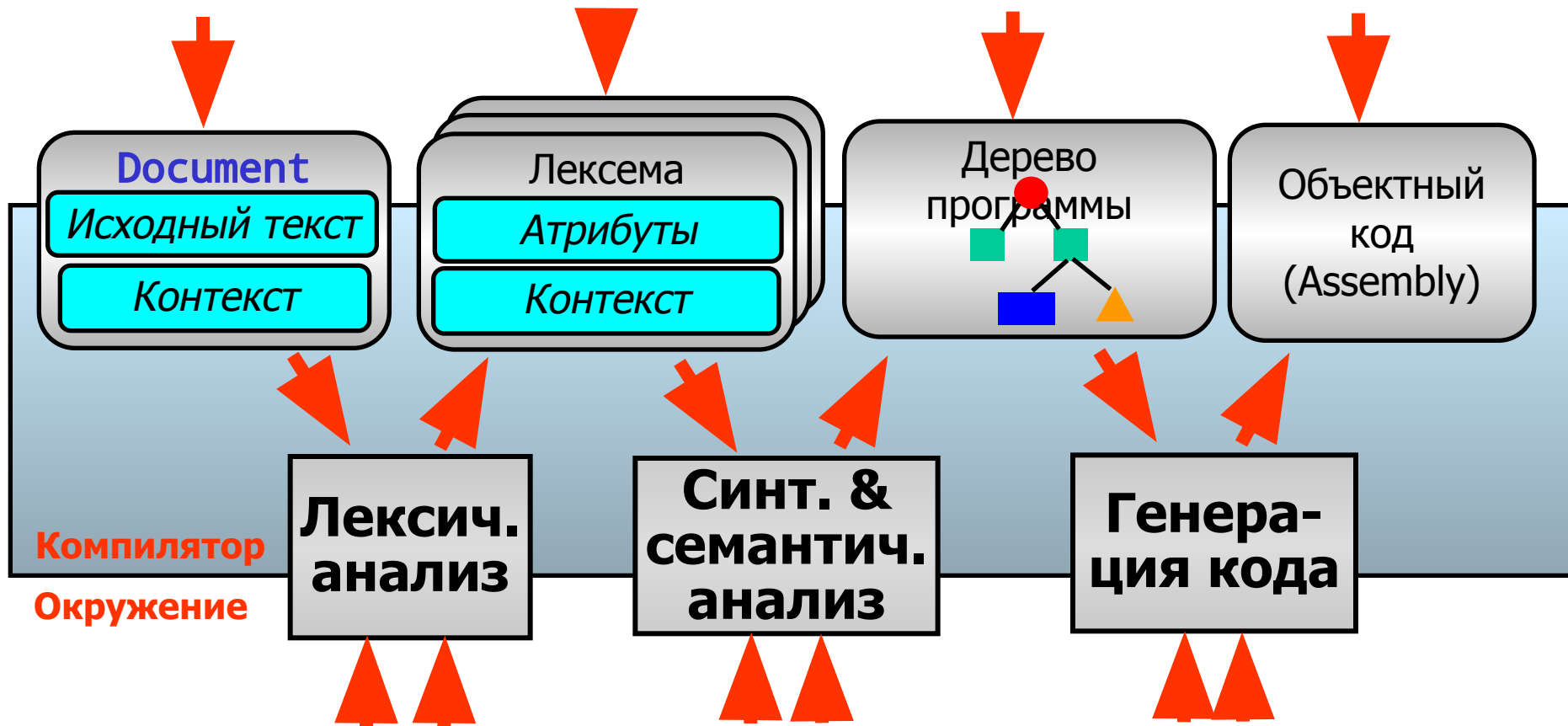
- GetType
- internal_type
- isOpen
- MemberwiseClone
- modifiers** (highlighted)
- name
- node
- report
- report_extra
- report_short

Additional context is provided on the right side of the popup:

- `TREE.scanner.getSourceCo` (next to `internal_type`)
- `MODIFIERS NODE.modifiers` (next to `modifiers`)
- `lemType)` (next to `node`)
- `ns.Length: i<n: i++)` (next to `report_extra`)

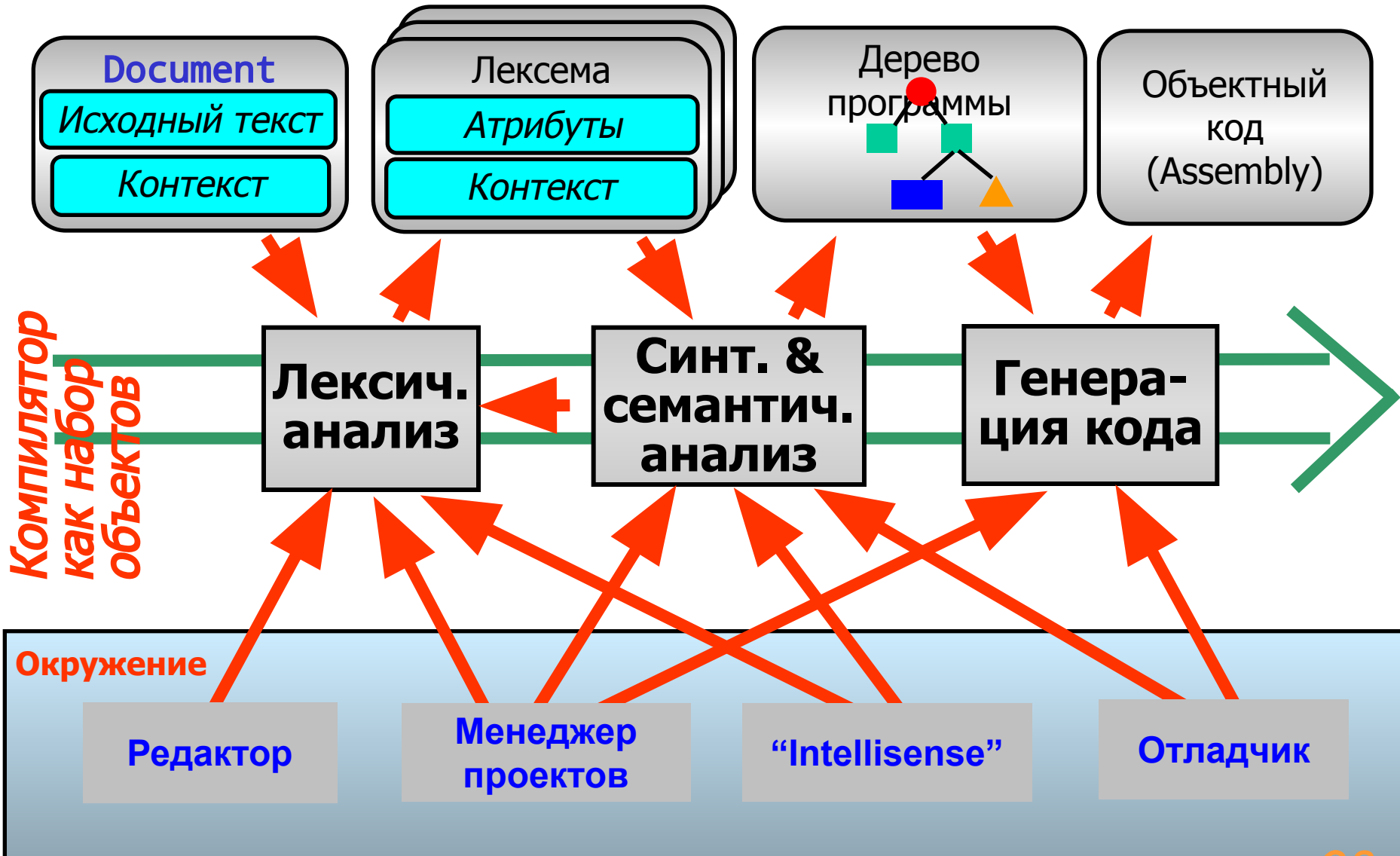
The bottom of the IDE shows the **Output** and **General** tabs.

Архитектура ССИ-компилятора (1)



*Компилятор как
коллекция ресурсов*

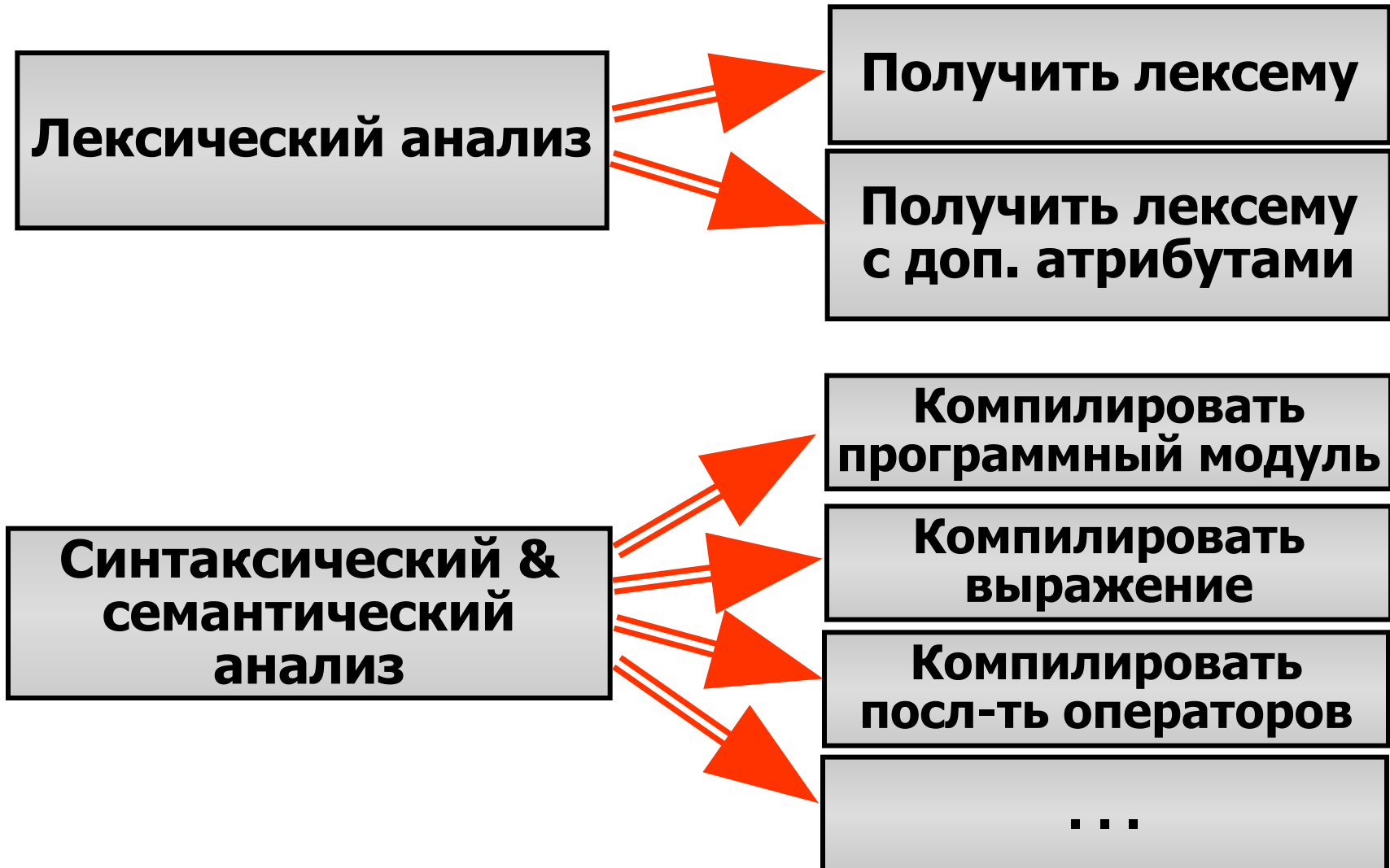
Архитектура ССИ-компилятора (2)



Архитектура ССИ-компилятора (3)

Фаза компляции

Прогр. модули

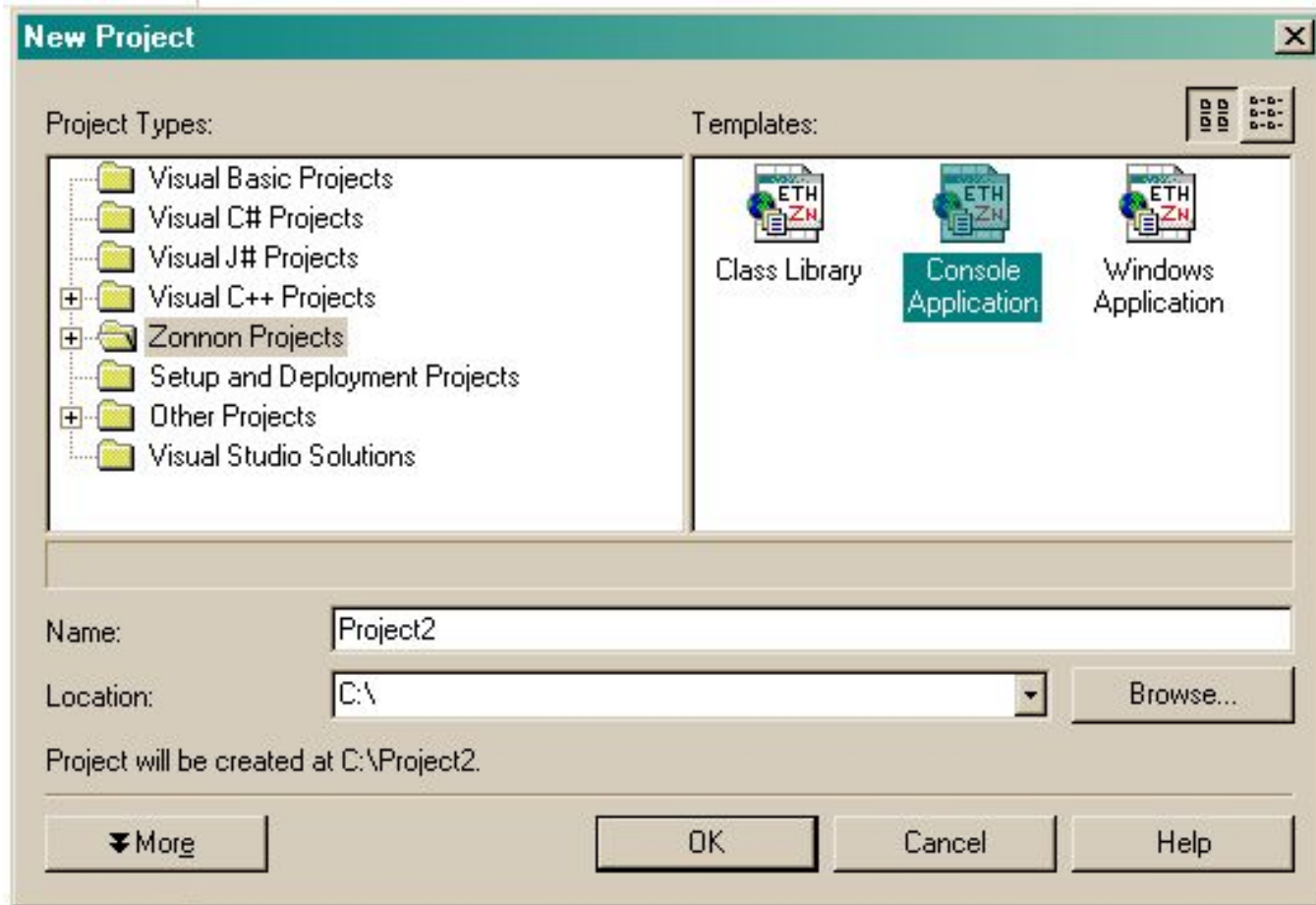


CCI: текущее состояние и статус

- 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.

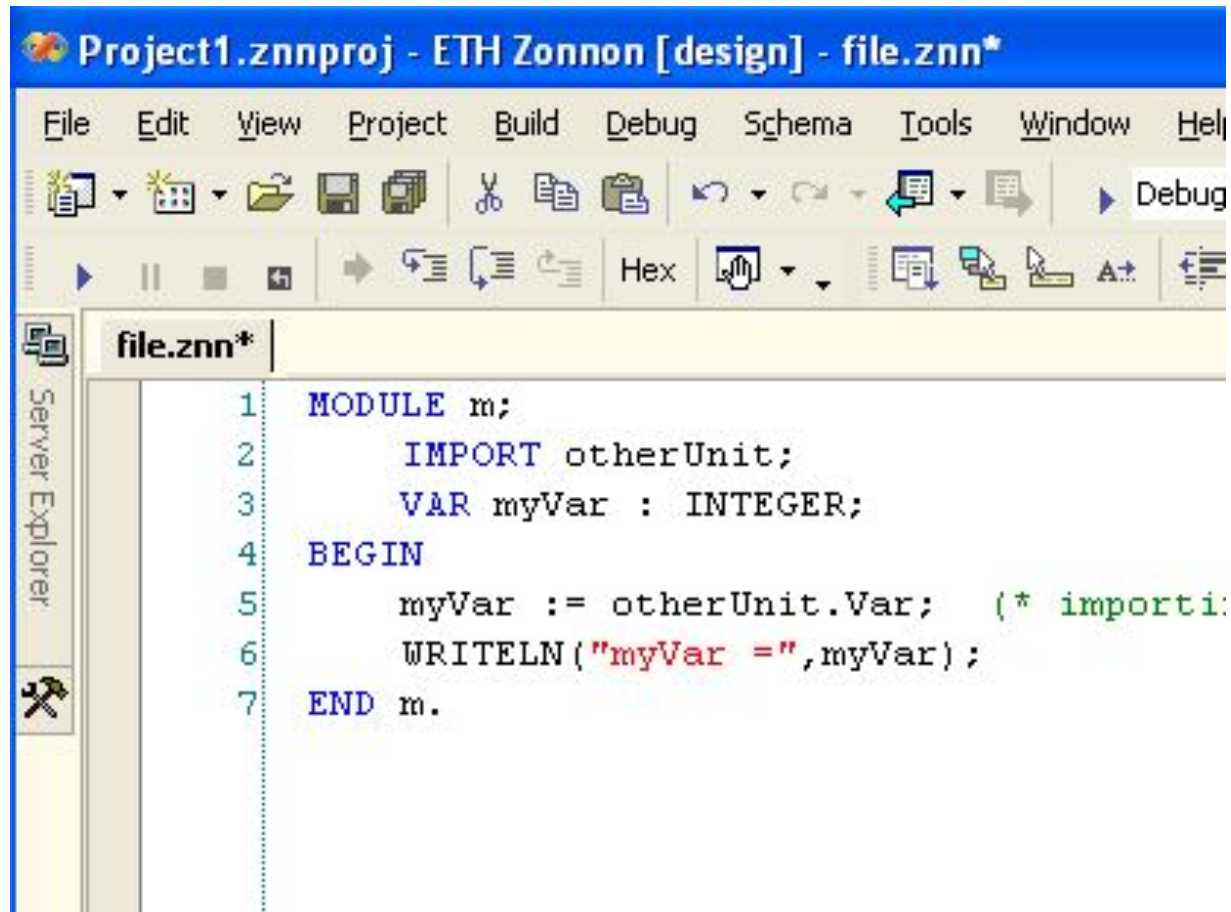
Интеграция Zоннон-компилятора в VS (1)

(Вместо демонстрации 😊)



Интеграция Zonnop-компилятора в VS (2)

(Вместо демонстрации 😊)

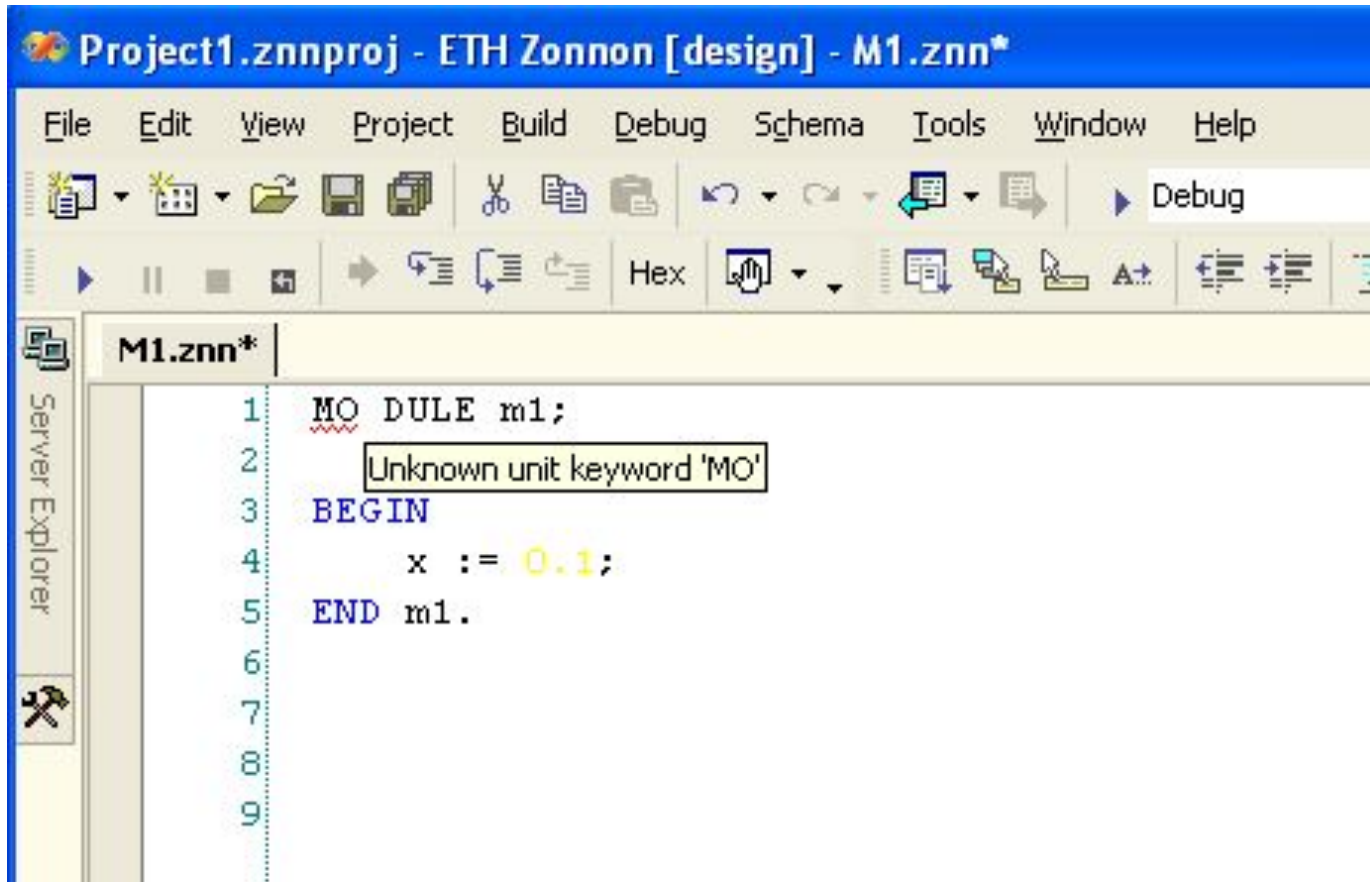


The screenshot shows the Zonnop IDE interface. The title bar reads "Project1.znnproj - ETH Zonnop [design] - file.znn*". The menu bar includes File, Edit, View, Project, Build, Debug, Schema, Tools, Window, and Help. The toolbar contains various icons for file operations, editing, and debugging. The main editor window displays the following Pascal code:

```
1  MODULE m;  
2      IMPORT otherUnit;  
3      VAR myVar : INTEGER;  
4  BEGIN  
5      myVar := otherUnit.Var; (* import:  
6      WRITELN("myVar =", myVar);  
7  END m.
```


Интеграция Zonnop-компилятора в VS (3)

(Вместо демонстрации 😊)



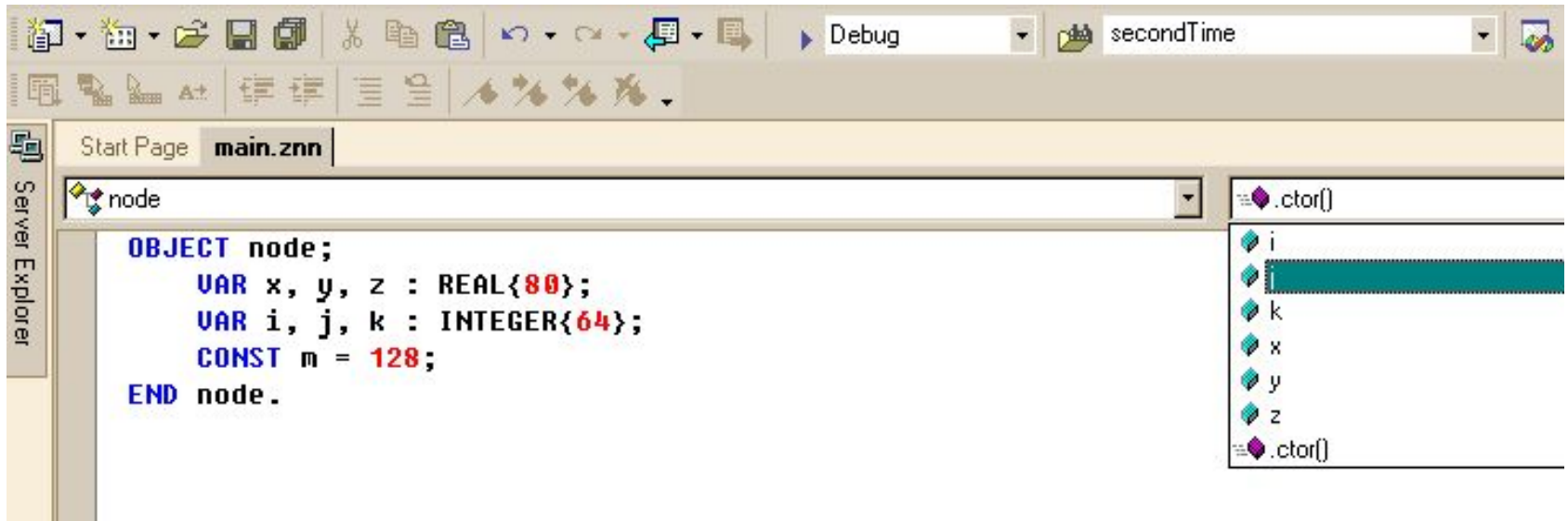
The screenshot shows the Visual Studio IDE with a project named 'Project1.znnproj - ETH Zonnop [design] - M1.znn*'. The menu bar includes File, Edit, View, Project, Build, Debug, Schema, Tools, Window, and Help. The toolbar contains various icons for file operations and debugging. The main editor window displays the code for 'M1.znn*':

```
1  MO DULE m1;  
2    
3  BEGIN  
4      x := 0.1;  
5  END m1.  
6  
7  
8  
9
```

A red squiggly line under the word 'MO' on line 1 indicates an error. A tooltip box points to this error with the text 'Unknown unit keyword 'MO''. The left sidebar shows the 'Server Explorer' pane.

Интеграция Zonnon-компилятора в VS (4)

(Вместо демонстрации 😊)



The screenshot shows the Visual Studio IDE with a Zonnon project. The main editor displays the following code in `main.znn`:

```
OBJECT node;  
  VAR x, y, z : REAL{80};  
  VAR i, j, k : INTEGER{64};  
  CONST m = 128;  
END node.
```

The Server Explorer on the left shows a project named `node`. The right-hand pane displays the class hierarchy for `node`, listing the following members:

- `.ctor()`
- `i`
- `j`
- `k`
- `x`
- `y`
- `z`
- `.ctor()`