Spec #: An Overview

Mohamed Aly, BSc(Eng)

Outline

- Motivation
- Overview
- Usage
- Language
- Examples
- References

Outline

- **Motivation**
- Overview
- Usage
- Language
- Examples
- References

Motivation - History

- Software development is costly and error prone.
- Software engineering aims for the development of correct and maintainable software.
- Various attempts (1960s – 70s).
- Correctness of programs are to be ensured via specification and verification.

Motivation - Problem

- Specifications are usually informal in the form of natural language documentation / standardized library interface descriptions.
- Programmers assumptions are left unspecified which complicates program maintenance.
- No guarantee for making sure that the program works under the assumptions the programmer has in mind or that the programmer might have overlooked some assumptions

Motivation – Why Spec#

- A programming language is being adopted widely due to its support, infrastructure, easiness, editing capabilities etc.
- Spec# is an superset of the existing programming language C#
- Based on the Microsoft .NET Framework.

Motivation – Current Applications

- The Microsoft Singularity project.
- Windows Server 2003 helped in discovering 10 - 13% of bugs in the source code and saved a million of dollars.
- Microsoft is still hiding it !

Outline

- Introduction
- Motivation

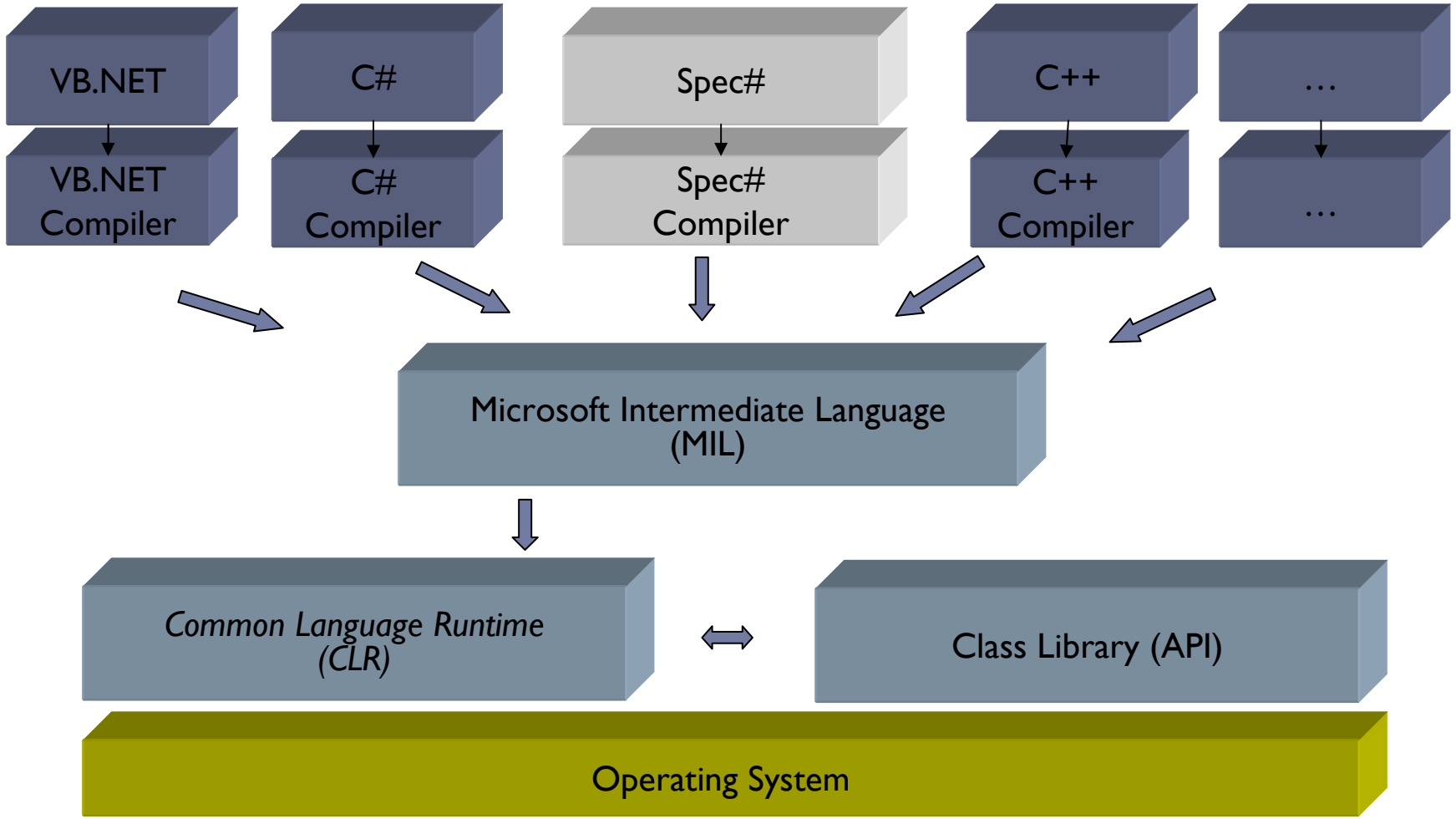
Overview

- Usage and Architecture
- Language
- Examples
- References

Overview: .NET Programs

- Source files (.ssc)
- Several source files collected into projects (.sscproject)
- Projects are collected into solutions (.sln)
- Compiler compiles projects (.exe .dll)
- Each project can use its own language and compiler.

Overview: .NET Framework



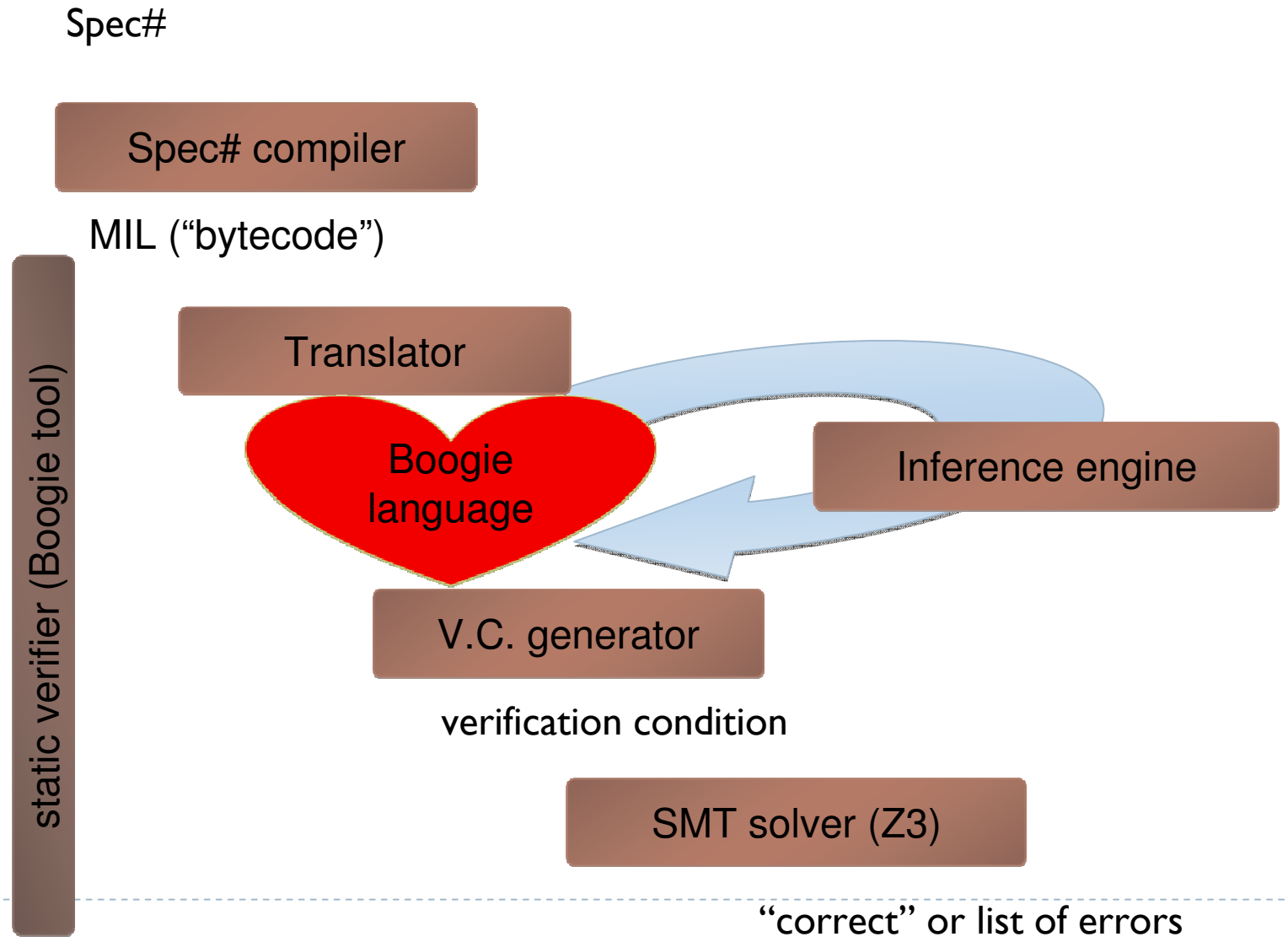
Overview: What is Spec# ?

- Programming Language: extension of C# with non-null types, checked exceptions and throws clauses, method contracts and object invariants.

- Compiler: statically enforces non-null types, emits run-time checks for method contracts and invariants, and records the contracts as metadata for consumption by downstream tools.

- Static Program Verifier: generates logical verification conditions from a Spec# program. Internally, it uses an automatic theorem prover that analyzes the verification conditions to prove the correctness of the program or find errors in it. (Boogie)

Overview: Verifier



Overview: Main Contributions of Spec#

- Extension of a popular language.
- Specification and reasoning about object invariants even in call backs.
- Dynamic checking and automatic verification.
- Smooth adoption paths to aid programmers profit from the benefits of specification.

Outline

- Introduction
- Motivation
- Overview

- **Usage**

- Language
- Examples
- References

Usage: Acquiring Spec#

- Spec# project is hosted at http://research.microsoft.com/en-us/projects/specsharp
- Lack of documentation
- Channel 9 of the MSDN has a wiki with some samples (badly maintained as well)
- Latest versions: v1.0.21125 for Visual Studio 2008 (release notes – deadlink)

Usage: Main guidelines

- Write each class with methods and specification in the same Spec# source file.
- Invariants may also be included.
- Compile
- Run the verifier

Outline

- Introduction
- Motivation
- Overview
- Usage
- **Language**
- Examples
- References

Language: Non-null

- Many errors occur are in the form of null-dereference.
- Spec# attempts to avoid all such errors.
- A type **X** is possibly null.
- A type **X!** cannot be null.

Language: Contracts

- What do we expect? (preconditions)
- What do we guarantee? (postconditions)
- What do we maintain? (invariants)

Language: Contracts, Preconditions

- Precondition is considered to be part of the signature of a method.
- To use the precondition, we use the **`requires`** keyword.
- A custom exception can be thrown on the failure of the precondition by using the keyword **`otherwise`**.

Language: Contracts, Postconditions

- Postcondition is considered to be part of the signature of a method.
- To use a postcondition, we use the **`ensures`** keyword.
- The **`result`** keyword may be used when referring to the result of the operation. The **`old`** keyword can be used when referring the value of the parameter at the beginning of the method.

Language: Contracts, Invariants

- Spec# adds a boolean field called **`inv`** to classes which tells the runtime whether the invariant currently holds.
- If it does hold, then the object is said to be in a consistent state.
- While the **`inv`** field is true, the fields within the object cannot be modified due to the possibility of breaking any of the invariants.

Language: Contracts, Invariants

- To declare an invariant we use the **`invariant`** keyword.
- If you need to update any of the invariant fields, Spec# makes you expose the object inside of an **`expose`** block.

Language: Contracts, Loop Invariants

- This invariant specifies conditions that must hold during the execution of a loop.
- Checked before the loop conditions are checked.
- The loop condition itself cannot be an invariant due to the last iteration of the loop must return false and that would break an invariant.

Language: Aggregates and Quantifiers

- **`sum`**
- **`count`**
- **`product`**
- **`min`**
- **`max`**

- **`forall`**
- **`exists`**
- **`exists unique`**

Language: Intervals

- `in (0 : n )` half-open interval 0 <= i < n
- `in (0 .. n)` closed interval 0<= i <= n

Language: Assumptions

- Assertions are checked at the runtime.
- Assumptions are meant for the Boogie verifier.
- Must be careful when using it. Incorrect assumption may prevent the static verifier from doing its job correctly.
- We use the keyword `assume` to declare an assumption.

Language: Frame Conditions

- Restrict which pieces of the state that a particular method is allowed to modify.
- To declare frame conditions we use the keyword **`modifies`**.

Outline

- Motivation
- Overview
- Usage
- Language
- **Examples**
- References

Outline

- Motivation
- Overview
- Usage
- Language
- Examples
- **References**

References

- The Spec# programming system: An overview, Mike Barnett, K. Rustan M. Leino, and Wolfram Schulte. In CASSIS 2004, LNCS vol. 3362, Springer, 2004.

- Program Verification Using the Spec# Programming System. (Tutorial presented at ETAPS 208 by Rustan Leino and Rosemary Monahan.)

- .NET 3.5, Design by contract and Spec#, Matthew Podwysocki,http://geekswithblogs.net/Podwysocki/archive/2007/12/10/117542.aspx , retrieved on 1/1/2009.

- Expert to Expert: Contract Oriented Programming and Spec#, Interview of some Spec# development team leaders, Channel 9 MSDN, Video, http://beta.channel9.msdn.com/shows/Going+Deep/Expert-to-Expert-Contract-Oriented-Programming-and-Spec/?Page=4 , retrieved on 1/1/2009 .

Questions ?

Thank You !