Transforming The Code: More Than Meets The Eye

Doni Pracner

Department of Mathematics and Informatics
Faculty of Sciences
University of Novi Sad

12th Workshop
“Software Engineering, Education & Reverse Engineering”
2 – 9 September 2012, Opatija, Croatia

Presentation Organization

Presentation Organisation

> “Change in all things is sweet.”
>
> — Aristotle

Therefore:

- Aristotle was not a software maintainer
- Software maintainers have a high diabetes risk

> “Change in all things is sweet.”
>
> — Aristotle

Therefore:

- Aristotle was not a software maintainer
- Software maintainers have a high diabetes risk

> “Change in all things is sweet.”
>
> — Aristotle

Therefore:

- Aristotle was not a software maintainer
- Software maintainers have a high diabetes risk

Software Evolution

- Software does not degrade with time on its own, the environment changes
- A need for constant maintainance and enhancement
- Software Evolution is (largely) repeated reengineering.

- Previously our group built tools with aim to make old, low level, assembly code easier to understand, and hopefully restructure it.
- Currently we are working on doing similar things with Java Bytecode.

WSL – *Wide Spectrum Language*

- The tools use WSL
- Developed by Martin Ward (since 1989)
- Strong mathematical core
- Formal transformations
- Wide spectrum: from abstract specifications to low level program code
- MetaWSL – operations on WSL code
- Successfully used in migrating legacy assembly code to maintainable C/COBOL code
- Implemented as FermaT program transformation system

Presentation Organisation

Our transformation process

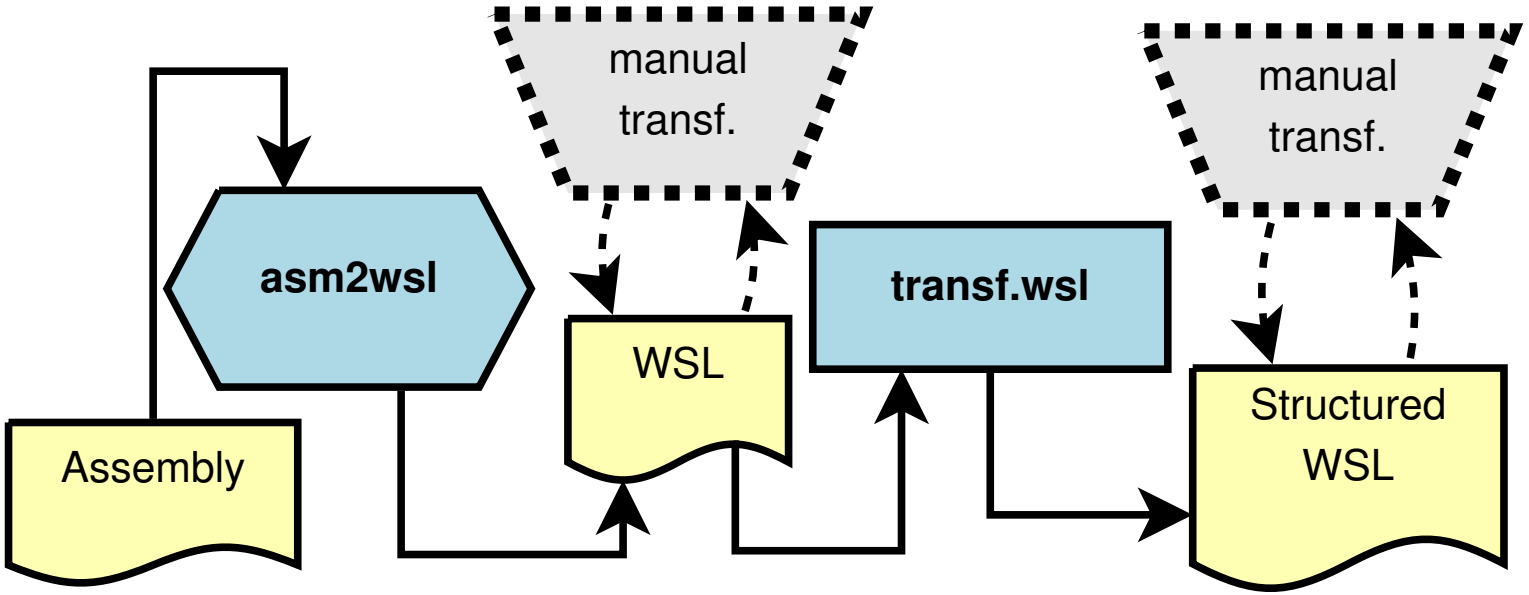

Figure: *Work-flow diagram*

- asm2wsl and transf.wsl were built “around” WSL
- Main goal is to get a high level version of the original program.

Asm2wsl application

- Translates a subset of x86 assembly to WSL
  - Mostly presumes 80286 for simplicity
- Implemented in Java
- Basically a line by line translator. Focus is on translating all aspects, not optimization (at this stage)
- Uses *Action systems* built into WSL for handling unstructured code
- We work with a “virtual” processor, simulating:
  - Processor registers (with Low and High parts)
  - Flags, overflow
  - Stack – a list
  - Labels – Action system names
  - Some special macros are recognized and translated directly
  - Procedures – nested Action systems

Automatic transformations – transf.wsl

- A small script to call the existing transformations
- Main transformations:
  - Collapse Action Systems
  - Transform DO . . . OD loops
  - Constant propagation
  - Remove Redundant
  - Flag removal

- Translating assembly programs to WSL so we can:
  - Generate call diagrams for easier understanding of original code;
  - Automatically transform the code to much simpler versions;
  - Optionally to manually tweak the results with more transformations

Problems

- Questions of feasibility of an all-high-level translation
- Problematic standards – order of operands, macros, architectures, various “hacks”, input/output
- Lack of a good assembly code base to experiment on
- Little experience with coding assembly – lack of “feel”
- Has been done in other ways (with auxiliary files)

- Plan: use mainly in Software Evolution courses for examples and adapt if needed

Presentation Organisation

Java Bytecode

- Similar to “classic” assembly in many ways
- Has a standard virtual machine
- Widely used, even by other language compilers
- A lot of code available, as well as experience in working with it.

- Plan: build translators to and from WSL
- Useful for formal verification or transformation, as well as code compiled from non-Java languages

MicroJava Bytecode

- First step: use MicroJava – proof of concept
- Developed by Hanspeter Moessenboeck, for use in Compiler Construction courses; not the same as “Java ME”
- Concepts are similar to “full” Java Bytecode, but simplified
- Less instructions, only int and char primitive types, arrays and basic classes
- MJ Bytecode does not encode types
- New tool *mjc2wsl* (mjc – MicroJava Compiled)
- Similar in many ways to asm2wsl – local variables, stack etc.

Generalized transformation process

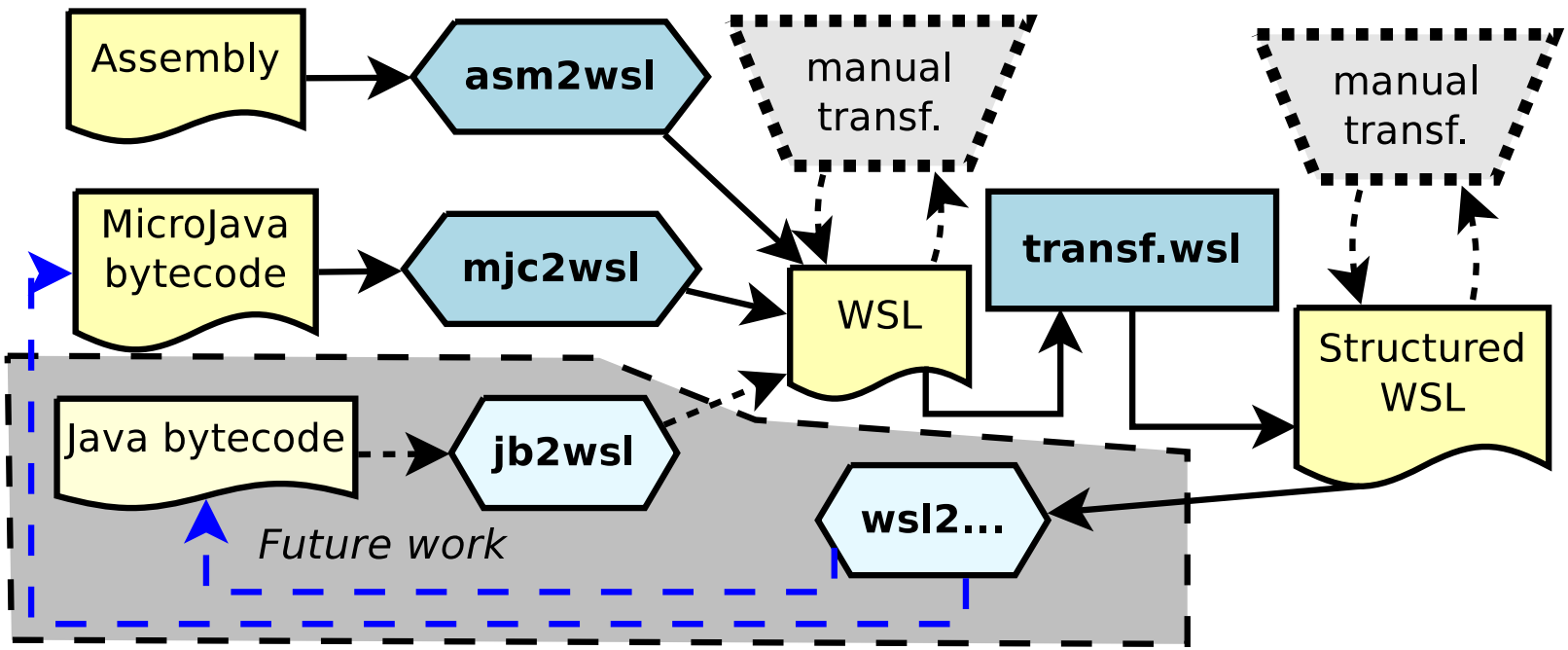

Figure: *Generalized work-flow diagram*

Type system

- WSL has no type system
- Therefore transformations can't check type consistency, possible source of errors.
- This is necessary for "full" Java Bytecode
- A *Wide Spectrum Type System* was developed by Matthias Ladkau in his PhD thesis
- Not yet fully integrated into FermaT
- It could be used to improve many of the transformations – one of the goals of this project

Presentation Organisation

Summary

- Previously: tools for some basic assembly conversion and transformation
- New tools for translating MicroJava Bytecode (simplified Java)
- Future work
  - Further work on the MJ Bytecode translator and transformations
  - Development of WSL to MJ Bytecode translators
  - Integration of the Wide Spectrum Type System into FermaT
  - Development of Java Bytecode translators to and from WSL

Thank you for your attention

Questions?