Analysis and Transformation of Source Code
ASF+SDF Meta-Environment

Paul Klint & Jurgen Vinju

IBM Nederland

30 Mei 2006
Today’s subjects

- Applications of ASF+SDF
  - Overview
  - Case 1: COBOL analysis
  - Case 2: COBOL transformation
  - Case 3: C analysis

- Points of interest
  - Language parametric ............. independent of the language
  - Modular .................... language dialects, embedded languages
  - “Everything is a transformation”
  - Layout and source code comments are important
Source code representations

A domain specific language for all transformations
Example applications of ASF+SDF
Languages everywhere

Source code representations
A domain specific language for all transformations
Example applications of ASF+SDF

Programmers, Architects, ...
People
Source Code
C, COBOL, Java, Assembler, ...
Facts
SQL, abstract syntax trees, CSV, ...
Pictures
UML, SVG, DOT, JPG, HTML, ...

Languages everywhere
“Everything is a transformation”
A domain specific language for all transformations

ASF+SDF
Example applications of ASF+SDF

- Compiler construction
- Documentation generation
- Model driven engineering (MDE)
- Code generators
- Source-to-source transformation
- Domain specific languages
- Formal verification
- Code formatting
- Coding convention checkers
- Security analysis
- ...

...
COBOL is a nice language for business applications
COBOL is old
COBOL has many, many versions and dialects
Many COBOL dialects do not have structured statements
  - No if-then-else
  - No while loop
  - No switch statement
  - No procedures/subroutines/functions/methods
  - But...the all powerfull goto was there
COBOL programs are very long: millions LOC
The resulting chaos is called: control flow spaghetti
Don’t touch the terrible control flow spaghetti monster!
Or else...
Case:

- Here is a COBOL legacy system from the 70’s
- Please fix this bug/add this feature
- You need to understand the code for this
- Spaghetti code is notoriously hard to understand

Questions:

- is this code spaghetti or not?
- what is the control flow of this code?

Answer:

- Here’s a picture, see for yourself!

A transformation from COBOL to a control flow picture
Demonstration

- Release 1.5.3 of the ASF+SDF Meta-Environment
- COBOL Control Flow Visualization
- Vrije Universiteit
- Niels Veerman, Ernst-Jan Verhoeven & Steven Klusener
COBOL transformation

- Case:
  - Please add a web interface to this system
  - You need to understand and change almost everything
  - Spaghetti code is notoriously hard to test
- Questions:
  - Can we improve the quality of this code?
  - Can we move from spaghetti to structured code?
  - Can we move from old COBOL to new COBOL?
- Answer:
  - Yes
  - By combining many simple rewriting steps
- A transformation from COBOL to COBOL
Demonstration

- Release 1.5.3 of the ASF+SDF Meta-Environment
- COBOL Control Flow Restructuring
  - Adding scope delimiters (END-IF, ...)
  - Finding goto idioms
  - Introducing structured statements
  - Introducing subroutines
- ABN-AMRO, Pink/Getronics, Vrije Universiteit
- Niels Veerman, Ernst-Jan Verhoeven & Steven Klusener
Introduction

- C is a low-level general purpose programming language
- C does not scale extremely well to extremely large systems
- C does not have:
  - Garbage collection
  - Exception handling
- Resulting workarounds involve much low-level programming
  - Checking for NULL pointers
  - 80% error handling code
- All kinds of scattered aspects of systems programming
- Coding conventions help system understanding
C Analysis

Case:
- Check the correctness/consistency of millions LOC C
- Source code conventions
- Comment conventions

Questions:
- Which are the actual input/output parameters of each function?
- Are all functions commented with input/output parameters?
- Are the comments correct?

Answers:
- Analyze code properties
- Analyze comments
- Correlate
int functionName(char *arg1, int arg2, MyType **arg3)

/* Input(s) :  arg1
 * my first argument
 * arg2
 * this is an integer input
 * Output(s) :  arg3
 * should be an output parameter */

{
    /* Body of function */
}
Details

- Release candidate 2.0 of the ASF+SDF Meta-Environment
- C input/output function parameters analysis
  - Control flow and data flow analysis (using CodeSurfer)
  - Detailed analysis of layout and comment conventions (using ASF)
  - Extensive correlation phase
  - CodeSurfer functionality also possible in ASF
- Anonymous company, CWI
- Magiel Bruntink, Tom Tourwe & Jurgen Vinju
Analysis of extracted facts...
Extracted Facts

- Basic facts: coded parameters versus commented parameters
- Raw inconsistencies: the difference between the sets
- Refinement: identify the causes
Extracted Facts

- Only coded as output parameters
- Correctly commented output parameters
- Only commented as output parameters
- Outputs commented as inputs

Etc.
## Extracted Facts

<table>
<thead>
<tr>
<th>Basic facts</th>
<th>Comm</th>
<th>Coded</th>
<th>Ratio (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lines of code</td>
<td>20994</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>Functions</td>
<td>92</td>
<td>173</td>
<td>53%</td>
</tr>
<tr>
<td>void functions</td>
<td>18</td>
<td>21</td>
<td>85%</td>
</tr>
<tr>
<td>difference</td>
<td>74</td>
<td>152</td>
<td>49%</td>
</tr>
<tr>
<td>Input params</td>
<td>124</td>
<td>304</td>
<td>41%</td>
</tr>
<tr>
<td>Output params</td>
<td>95</td>
<td>190</td>
<td>50%</td>
</tr>
<tr>
<td>All params</td>
<td>219</td>
<td>494</td>
<td>45%</td>
</tr>
</tbody>
</table>
## Extracted Facts

<table>
<thead>
<tr>
<th>Raw inconsistences</th>
<th>Number</th>
<th>Ratio (%)</th>
<th>of basic facts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not coded inputs</td>
<td>21</td>
<td>17%</td>
<td>comm inputs</td>
</tr>
<tr>
<td>Not comm inputs</td>
<td>5</td>
<td>2%</td>
<td>coded inputs</td>
</tr>
<tr>
<td>Not coded outputs</td>
<td>6</td>
<td>6%</td>
<td>comm outputs</td>
</tr>
<tr>
<td>Not comm outputs</td>
<td>25</td>
<td>12%</td>
<td>coded outputs</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Raw summary</th>
<th>Number</th>
<th>Ratio (%)</th>
<th>of basic facts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inconsistencies</td>
<td>57</td>
<td>25%</td>
<td>comm params</td>
</tr>
<tr>
<td>Inconsistent params</td>
<td>33</td>
<td>14%</td>
<td>comm params</td>
</tr>
<tr>
<td>Inconsistent functions</td>
<td>24</td>
<td>32%</td>
<td>comm funcs</td>
</tr>
</tbody>
</table>
### Extracted Facts

<table>
<thead>
<tr>
<th>Refined inconsistencies</th>
<th>Number</th>
<th>Ratio (%)</th>
<th>of</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comm inp as outp</td>
<td>3</td>
<td>60%</td>
<td>not com inp</td>
</tr>
<tr>
<td>Code has added inp</td>
<td>2</td>
<td>40%</td>
<td>not com inp</td>
</tr>
<tr>
<td>Comm outp as inp</td>
<td>21</td>
<td>84%</td>
<td>not com outp</td>
</tr>
<tr>
<td>Code has added outp</td>
<td>4</td>
<td>16%</td>
<td>not com outp</td>
</tr>
<tr>
<td>Comm inp as outp</td>
<td>3</td>
<td>50%</td>
<td>not cod outp</td>
</tr>
<tr>
<td>Code has less outp</td>
<td>3</td>
<td>50%</td>
<td>not cod outp</td>
</tr>
<tr>
<td>Com. outp as inp</td>
<td>21</td>
<td>100%</td>
<td>not cod inp</td>
</tr>
<tr>
<td>Code has less inp</td>
<td>0</td>
<td>0%</td>
<td>not cod inp</td>
</tr>
</tbody>
</table>
### Extracted Facts

<table>
<thead>
<tr>
<th>Refined summary</th>
<th>Number</th>
<th>Ratio (%)</th>
<th>of</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inp versus outp params</td>
<td>24</td>
<td>73%</td>
<td>incons. params</td>
</tr>
<tr>
<td>............ per function</td>
<td>20</td>
<td>83%</td>
<td>incons. funcs</td>
</tr>
<tr>
<td>Added or rem params</td>
<td>9</td>
<td>27%</td>
<td>incons. params</td>
</tr>
<tr>
<td>............ per function</td>
<td>3</td>
<td>15%</td>
<td>incons. funcs</td>
</tr>
<tr>
<td>Funcs with both issues</td>
<td>1</td>
<td>2%</td>
<td>incons. funcs</td>
</tr>
</tbody>
</table>
• The extraction and analysis can contain errors
• We did the experiment 5 times before fixing all bugs
• Manually inspected all 24 faulty functions (out of 124)
• The only assumption: when analysis and comments agree, they are correct.
ASF+SDF is a tool for any kind of analysis or transformation

What makes it different:

- High level
- Language parametric independent of the language
- Modular language dialects, embedded languages
- Layout and source code comments are important
More information

- **Link**
  - [http://www.meta-environment.org](http://www.meta-environment.org)

- **Email:**
  - meta-users-list@cwi.nl
  - jurgen.vinju@cwi.nl
  - paul.klint@cwi.nl

- Questions are welcome!
C Transformation

Case:
- Please remove all parameter checking code
- Please remove all error handling code
- And then use AOP to weave it back in

Questions:
- Can we identify the aspect code correctly?
- Can we remove it without damaging the other code?
- Can we be semantics preserving?

Answer:
- Work in progress
- Extensive static analysis
- Not many simple transformations, but a few very complex ones
- How to test?
Examples

[Cobol source] to [CFG] to [dot picture]

72-PT-MS SECTION.
  7201.
    MOVE M-PT IN TRA-NUM TO M-ACC.
    MOVE 0 TO PT-M-TABLE-R.
  7203.
    IF M-ACC < 01
      ADD PERIOD-ACC TO M-ACC
      MOVE 1 TO M-TABLE ( M-ACC )
      GO TO 7205.
    IF M-ACC < PERIOD-ACC
      MOVE 1 TO M-TABLE ( M-ACC )
      GO TO 7205.
    IF M-ACC = PERIOD-ACC
      MOVE 1 TO M-TABLE ( M-ACC )
      GO TO 7205.
    SUBTRACT PERIOD-ACC FROM M-ACC.
    GO TO 7203.
  7205.
    ADD PERIOD-ACC TO M-ACC.
    IF M-ACC > 72
      GO TO 7207.
    MOVE 1 TO M-TABLE ( M-ACC )
    GO TO 7205.
  7207.
    MOVE PT-M-TABLE-R TO PT-MS.
    MOVE 0 TO DEP-DATA IN TRA-NUM.
    IF P-TYPE
[Cobol source] to [CFG] to [dot picture]

```
72-PT-MS SECTION.
  7201.
    MOVE M-PT IN TRA-NUM TO M-ACC.
    MOVE 0 TO PT-M-TABLE-R.
  7203.
    IF M-ACC < 01
      ADD PERIOD-ACC TO M-ACC
      MOVE 1 TO M-TABLE (M-ACC)
      GO TO 7205.
    IF M-ACC < PERIOD-ACC
      MOVE 1 TO M-TABLE (M-ACC)
      GO TO 7205.
    IF M-ACC = PERIOD-ACC
      MOVE 1 TO M-TABLE (M-ACC)
      GO TO 7205.
    SUBTRACT PERIOD-ACC FROM M-ACC.
    GO TO 7203.
  7205.
    ADD PERIOD-ACC TO M-ACC.
    IF M-ACC > 72
      GO TO 7207.
      MOVE 1 TO M-TABLE (M-ACC).
      GO TO 7205.
    7207.
      MOVE PT-M-TABLE-R TO PT-MS.
      MOVE 0 TO DEP-DATA IN TRA-NUM.
```

digraph "A-PROGRAM"
  "72-PT-MS" [shape = box]
  "7201 IN 72-PT-MS" [label = "7201"]
  "7203 IN 72-PT-MS" [label = "7203"]
  "7205 IN 72-PT-MS" [label = "7205"]
  "7207 IN 72-PT-MS" [label = "7207"]
  "7209 IN 72-PT-MS" [label = "7209"]
  "7299 IN 72-PT-MS" [label = "7299"]
  subgraph "cluster_72-PT-MS"
    "72-PT-MS" ;
    "7201 IN 72-PT-MS" ;
    "7203 IN 72-PT-MS" ;
    "7205 IN 72-PT-MS" ;
    "7207 IN 72-PT-MS" ;
    "7209 IN 72-PT-MS" ;
    "7299 IN 72-PT-MS" ;
  "72-PT-MS" -> "7201 IN 72-PT-MS" [weight = 100] ;
  "7201 IN 72-PT-MS" -> "7203 IN 72-PT-MS" [style = invis, weight =
    "7203 IN 72-PT-MS" -> "7205 IN 72-PT-MS" [style = invis, weight =
    "7205 IN 72-PT-MS" -> "7207 IN 72-PT-MS" [style = invis, weight =
    "7207 IN 72-PT-MS" -> "7209 IN 72-PT-MS" [style = invis, weight =
    "7209 IN 72-PT-MS" -> "7299 IN 72-PT-MS" [style = invis, weight =
    "7299 IN 72-PT-MS" -> "7201 IN 72-PT-MS" [color = red, label = "4"
    "7201 IN 72-PT-MS" -> "7203 IN 72-PT-MS" [color = red, label = "4"
    "7203 IN 72-PT-MS" -> "7205 IN 72-PT-MS" [color = red, label = "4"
    "7205 IN 72-PT-MS" -> "7207 IN 72-PT-MS" [color = red, label = "4"
    "7207 IN 72-PT-MS" -> "7209 IN 72-PT-MS" [color = red, label = "4"
    "7209 IN 72-PT-MS" -> "7299 IN 72-PT-MS" [color = red, label = "4"
```
72-PT-MS SECTION.
7201.
MOVE M-PT IN TRA-NUM TO M-ACC.
MOVE 0 TO PT-M-TABLE-R.
7203.
IF M-ACC < 01
   ADD PERIOD-ACC TO M-ACC
   MOVE 1 TO M-TABLE ( M-ACC )
   GO TO 7205.
IF M-ACC < PERIOD-ACC
   MOVE 1 TO M-TABLE ( M-ACC )
   GO TO 7205.
IF M-ACC = PERIOD-ACC
   MOVE 1 TO M-TABLE ( M-ACC )
   GO TO 7205.
SUBTRACT PERIOD-ACC FROM M-ACC.
GO TO 7203.
7205.
ADD PERIOD-ACC TO M-ACC.
IF M-ACC > 72
   GO TO 7207.
   MOVE 1 TO M-TABLE ( M-ACC ).
   GO TO 7205.
7207.
MOVE PT-M-TABLE-R TO PT-MS.
MOVE 0 TO DEP-DATA IN TRA-NUM.
IF P-TYPE
   ADD PERIOD-ACC TO M-ACC
   MOVE 1 TO M-TABLE ( M-ACC )
   GO TO 7205.
Examples

[Cobol source] to [Cobol source]

72-PT-MS SECTION.

7201.
  MOVE M-PT IN TRA-NUM TO M-ACC.
  MOVE 0 TO PT-M-TABLE-R.

7203.
  IF M-ACC < 01
    ADD PERIOD-ACC TO M-ACC
    MOVE 1 TO M-TABLE ( M-ACC )
    GO TO 7205.
  IF M-ACC < PERIOD-ACC
    MOVE 1 TO M-TABLE ( M-ACC )
    GO TO 7205.
  IF M-ACC = PERIOD-ACC
    MOVE 1 TO M-TABLE ( M-ACC )
    GO TO 7205.
  SUBTRACT PERIOD-ACC FROM M-ACC.
  GO TO 7203.

7205.
  ADD PERIOD-ACC TO M-ACC.
  IF M-ACC > 72
    GO TO 7207.
  MOVE 1 TO M-TABLE ( M-ACC ).
  GO TO 7205.

7207.
  MOVE PT-M-TABLE-R TO PT-MS.
  MOVE 0 TO DEP-DATA IN TRA-NUM.
**Examples**

[Cobol source] to [Cobol source]

72-PT-MS SECTION.
7201.
   MOVE M-PT IN TRA-NUM TO M-ACC.
   MOVE 0 TO PT-M-TABLE-R.
7203.
   IF M-ACC < 01
      ADD PERIOD-ACC TO M-ACC
   MOVE 1 TO M-TABLE ( M-ACC )
   GO TO 7205.
   IF M-ACC < PERIOD-ACC
      MOVE 1 TO M-TABLE ( M-ACC )
   GO TO 7205.
   IF M-ACC = PERIOD-ACC
      MOVE 1 TO M-TABLE ( M-ACC )
   GO TO 7205.
   SUBTRACT PERIOD-ACC FROM M-ACC.
   GO TO 7203.
7205.
   ADD PERIOD-ACC TO M-ACC.
   IF M-ACC > 72
      GO TO 7207.
   MOVE 1 TO M-TABLE ( M-ACC ).
   GO TO 7205.
7207.
   MOVE PT-M-TABLE-R TO PT-MS.
   MOVE 0 TO DEP-DATA IN TRA-NUM.
7299.
   EXIT.

72-PT-MS SECTION.
   RESTRUCTURE-PAR.
   PERFORM 7201
   PERFORM 7203
   PERFORM 7205
   PERFORM 7207.
7299.
   EXIT.

BAR SECTION.
   BAR-PARAGRAPH.
   GOBACK.

72-PT-MS-SUBROUTINES SECTION.
7201.
   MOVE M-PT IN TRA-NUM TO M-ACC
   MOVE 0 TO PT-M-TABLE-R.
7203.
   PERFORM TEST BEFORE UNTIL
      (M-ACC < 01) OR ((M-ACC < PERIOD-ACC) OR (M-ACC = PERIOD-ACC))
   SUBTRACT PERIOD-ACC FROM M-ACC END-PERFORM
   IF M-ACC < 01
      ADD PERIOD-ACC TO M-ACC
   END-IF
   MOVE 1 TO M-TABLE ( M-ACC ).
   GO TO 7205.
7207.
   ADD PERIOD-ACC TO M-ACC
Example: comparing abstractions

[old CFG] with [new CFG]
Motivation for SDF

- Any existing programming language:
  - C, COBOL, FORTRAN, PL/I, C++
Any existing programming language:
  - C, COBOL, FORTRAN, PL/I, C++

Precision and correctness implies no heuristics:
  - Scanners suck: their heuristics are often meaningless
  - Implicit disambiguation sucks: same story
Motivation for SDF

- Any existing programming language:
  - C, COBOL, FORTRAN, PL/I, C++

- Precision and correctness implies no heuristics:
  - Scanners suck: their heuristics are often meaningless
  - Implicit disambiguation sucks: same story

- Our tower of Babel:
  - Programming languages have many dialects
  - Programming languages come with embedded languages
  - Every language has a particular ‘disambiguation semantics’
Motivation for SDF

- Any existing programming language:
  - C, COBOL, FORTRAN, PL/I, C++
- Precision and correctness implies no heuristics:
  - Scanners suck: their heuristics are often meaningless
  - Implicit disambiguation sucks: same story
- Our tower of Babel:
  - Programming languages have many dialects
  - Programming languages come with embedded languages
  - Every language has a particular ‘disambiguation semantics’
- For any meta program, so no abstraction:
  - Every character of the input may be important
  - Human readable/maintainable input and output source code
Motivation for SDF

- Any existing programming language:
  - C, COBOL, FORTRAN, PL/I, C++
- Precision and correctness implies no heuristics:
  - Scanners suck: their heuristics are often meaningless
  - Implicit disambiguation sucks: same story
- Our tower of Babel:
  - Programming languages have many dialects
  - Programming languages come with embedded languages
  - Every language has a particular ‘disambiguation semantics’
- For any meta program, so no abstraction:
  - Every character of the input may be important
  - Human readable/maintainable input and output source code
- Grammars as syntax definitions first!
  - No factoring whatsoever
  - Hidden recursion, cycles and ambiguities, all allowed
**SDF** — parser generation from syntax definitions

- integrated lexical and context-free syntax ........ No scanner
- accepts all context-free languages ................. Modular
- declarative disambiguations ...................... No heuristics
- fully informative parse trees ................... No abstraction
Design and implementation

- SGLR implements SDF:
  - **Scannerless** parsing (no more meaningless tokenizations)
  - **Generalized** LR (unbounded lookahead without backtracking)
  - **Filtering** (precise removal of ambiguous derivations)

- GLR efficiency:
  - Complexity $\in O((\text{input length})^{(\text{max rule length})})$
  - Average case comes close to LR (for almost LR languages)

- Scannerless efficiency
  - Factor 5–8 penalty
  - More languages
  - Filters help a lot!
SGLR

- Tomita’s Generalized LR, fixed by Farshi for hidden recursion
  - GSS: A stack-like graph (DAG) encodes parallel parsers
  - Some actions lead to stack merging
  - Multiple derivations lead to local ambiguity nodes
- Extended by Visser
  - Generation of character-level grammars
  - Scanner heuristics replaced by filters on parse table and stack
- ATerms for parse forests
  - Maximal subterm sharing: memory efficiency
  - Automatic garbage collection
  - Efficient exchange format
- Used for parsing many (ambiguous) programming languages
A filter is the implementation of a disambiguation

Filters are defined as functions on parse forests

... but might be implemented on parse tables or stacks

The earlier a filter is applied, the faster the parsing

CFG+Filter may lead to a context-sensitive language
Disambiguation by filtering

- follow restrictions (longest match) ........ parseable & SGLR
- reject productions (keyword reservation) ............ SGLR
- priorities/associativity (expression operators) .... parseable
- preferences (horizontal priority) ................. SGLR
- anything else ........................................ post parse filtering
Maximal sharing

"(true or true) or true" versus "true or (true or true)"
21 nodes
Maximal sharing

C Transformation
Examples
Motivation for SDF
Design and implementation
Examples

```
Maximal sharing

 Bool "|" Bool -> Bool
  /\                /
 Bool "|" Bool -> Bool  Bool "|" Bool -> Bool
  /\                /
 Bool "|" Bool -> Bool  Bool "|" Bool -> Bool
  /\                /
 "true" -> Bool  "|"  "false" -> Bool
    /\                /
   "true"  "|"  "false"
```
Example (Longest match and associativity)

**lexical syntax**

```
[A-Za-z][A-Za-z0-9_]* -> Identifier
```

**lexical restrictions**

Identifier 

`-/- [A-Za-z0-9_]`

**context-free syntax**

```
Expression Expression -> Expression {right}
Identifier -> Expression
```
Example (Reserved keywords)

**lexical syntax**

[A-Za-z][A-Za-z0-9\_]* -> Identifier

**context-free syntax**

Expression Expression -> Expression {right}

Identifier -> Expression

"begin" Expression "end" -> Expression

"begin" | "end" -> Identifier {reject}
Example (Reserved keywords)

lexical syntax

\[[A-Za-z][A-Za-z0-9\_]*\] -> Identifier

context-free syntax

Expression Expression -> Expression \{right\}  
Identifier -> Expression  
"begin" Expression "end" -> Expression  
"begin" | "end" -> Identifier \{reject\}

lexical syntax \%

[b][e][g][i][n] -> "begin"  
[e][n][d] -> "end"
Examples

Example (Priorities)

<table>
<thead>
<tr>
<th>context-free priorities</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;~&quot; Expression -&gt; Expression &gt;</td>
</tr>
<tr>
<td>Expression &quot;[&quot; Expression &quot;]&quot; -&gt; Expression &lt;0&gt; &gt;</td>
</tr>
<tr>
<td>Expression Expression -&gt; Expression {right}</td>
</tr>
</tbody>
</table>

- Priorities declare a **vertical** ordering
- Priority relation is transatively closed
- Associativity is a degenerate case of priority
Example (Preferences (as opposed to reservation))

context-free syntax

Expression Expression -> Expression {right}
"begin" Expression "end" -> Expression {prefer}

- Preferences declare a horizontal ordering
- Dual of prefer is avoid