Overview

- Problem: Software maintenance
  - Large, distributed, multi-language and developed using pre-built components
  - Difficult and expensive, especially if the source code is not available
  - Effective SC tools can greatly simplify and reduce maintenance effort
- Solution: Software Comprehension Environment
  - Profiling and analysis of distributed systems
  - Analysis of program features
  - Common data repository
  - Software views
- Evaluation: Case study
  - Mozilla: Web browser
  - Jext: Programmers’ Text Editor
  - TechReport: Publication database
Software maintenance: Relative Cost

\[\text{Corrective maintenance} \quad \text{Adaptive maintenance} \quad \text{Perfective maintenance}\]

⇒ Approximately 80% of the maintenance effort is spent on non-corrective tasks

Software maintenance: Cost Distribution

⇒ Rigorous design and thorough testing
  - Corrective maintenance

⇒ Limiting non-corrective changes
  - Enhancements
  - Adapting to hardware/software infrastructure changes
  - Not an option: System evolution depends on these changes to meet its user’s needs.
Software maintenance: Cost Reduction

- Reducing amount of effort spent on maintenance
  
  \( \frac{1}{2} \) the maintenance effort is spent on understanding the system’s logic and behavior. [Pigoski-1994]

➤ Effective software comprehension tools

Outline

- Motivation
  ➤ Software Comprehension
   - Architecture of Software Comprehension Environment
   - Software views / Case Study
   - Related Work
   - Conclusions

Software Comprehension
Outline

✓ Software Comprehension
  ➢ Architecture of Software Comprehension Environment
    • Software Views / Case Study
    • Related Work
    • Conclusions

Architecture of the Software Comprehension Environment

Data Gathering Subsystem
**Data Gathering Subsystem**

- Program Facts/Data
  - Static: Entities and Relations
  - Dynamic: Entities, Relations and Events
- Distributed Profiler
  - Local Profiler
  - Logical Time Server
- Remote Interactions
  - Network interceptor
    ➔ Endpoint

---

**Data Gathering**

- **Endpoint entity**
  - Local and Remote host
  - Local and Remote port numbers
  - Time-stamp
- **Connects Relation**

---

**Example: Sequence Diagram**

---
Data Repository

- Logical Models
  - Language definitions
  - Program data
- Implementation
  - Relational database
- Query/Manipulation Language
  - Manipulation: SQL
  - Query/retrieval: SMQL

Language Definitions

- Model(General) = \text{Graph}(E_g, R_g)
  \begin{align*}
  E_g &: \text{Set of entity types} \\
  R_g &: \text{Set of relation types}
  \end{align*}
- Java, example:
  \begin{align*}
  \text{Model(Java)} &= \text{Graph}(E_{java}, R_{java}) \\
  E_{java} &\subset E_g: \text{Entity types supported by Java} \\
  R_{java} &\subset R_g: \text{Relation types supported by Java}
  \end{align*}
  Such that the source and destination entity \( E \in E_{java} \)
- C++, defining
  \begin{align*}
  \text{Model(C++)} &= \text{Graph}(E_{c++}, R_{c++}) \\
  E_{c++} &= E_{java} \cup \{ \text{template, struct, typedef, function} \} \\
  &\setminus \{ \text{interface, package} \}
  \end{align*}

SMQL: Software Modeling Query Language

- Set-based:
  - Typed Set (SMQL Objects)
    - Entity Set
    - Relation Set
    - Event Set
    - Composite set
    - Generic Set
- Operations
  - Union, intersection, Difference
- Functions
  - SMQL Filters implement in Java
  - Closure, Composition, …, etc.
SMQL: Example

Union of relations from Mozilla and Jetty

SMQL Filters

- SMQL filters = Analyzers
  - Java implementation of IFilter interface
- Implemented filters
  - Closure (Source, Target, RelationTypes)
  - Composition (E1, E2, ...)
  - Callgraph (E1, E2, ...)
  - AnalyzeEvents (E1, E2, ...)
  - FeatureSimilarity (E1, E2, ...)
  - MethodSequences (E1, E2, ...)
  - Clone (EntitySet || RelationSet || EventSet)
  - Output_Dot (RelationSet )
    - Output_sml (EntitySet || RelationSet || EventSet)

Analysis and Visualization
RelationSet: MixedRelations

RelationSet: KwicCallgraph (Call-graph)

EventSet : KwicEvents (Sequence Diagram)
Summary of Architecture

- Support
  - Analysis of distributed
  - Analysis of program features
  - Static and dynamic analysis
  - Multiple languages
- Extensible architecture
  - Analysis - IFilter
  - Visualization - ISoftView

Software Views

Taxonomy of Software Views

[Diagram showing the taxonomy of software views]

Requires information about runtime objects.
Software Views

- Feature views
  - Impact of changing one feature on other features
  - Feature implementation and dependencies
- Remote-interaction
  - Highlights interactions between components of distributed systems
- Thread-interaction
  - Highlights potential concurrency issues
- Class usage protocol
  - Documenting usage scenarios

Terminology

- Classification of objects
  Let $\mathcal{F} = \{\text{set of features}\}$
  - Local($F_k$): objects created and used by $F_k$
  - Import($F_k$): objects used by $F_k$ and created by $F_j$
  - Export ($F_k$): objects created by $F_k$ and used by $F_j$
- Relationships:
  - $\text{depends}(F_k, F_j) = \text{Import}(F_k) \cap \text{Export}(F_j) \neq \emptyset$
  - $\text{shares}(F_k, F_j) = \text{Import}(F_k) \cap \text{Import}(F_j) \neq \emptyset$

Feature-Interaction

- Feature $\leftrightarrow$ Marked-Trace
- Construction Feature-interaction diagram
  - Identity Import ($F$) and Export($F$) sets
  - Compute $\text{depends}(F_k, F_j) = \text{Import}(F_k) \cap \text{Export}(F_j)$
  - Compute $\text{shares}(F_k, F_j) = \text{Import}(F_k) \cap \text{Import}(F_j)$
  - Draw a depends-edge if $\text{depends}(F_k, F_j) \neq \emptyset$
  - Draw a shares-edge if $\text{shares}(F_k, F_j) \neq \emptyset$
Feature-Interaction: Example

The object is created by the corresponding feature.
The object is used by the corresponding feature, but created by other feature.

Interactions within a given Feature are ignored.

July 25, 2005  Maher Salah

Feature-Interaction: Example

- Import/Export sets:
- Depends/Shares:
  - Depends (F2, F1) = E(F1) \cap I(F2) = {O2}
  - Depends (F1, F2) = E(F2) \cap I(F1) = {O1}
  - Depends (F3, F2) = E(F2) \cap I(F3) = {O4}
  - Depends (F4, F2) = E(F2) \cap I(F4) = {O4}
  - Shares (F3, F4) = I(F3) \cap I(F4) = {O4}

Feature-interaction view

Feature-1 is related to Feature-2, Feature-3, Feature-4.

Uses: Object O4

July 25, 2005  Maher Salah

Jext: Feature-interaction

- Feature interaction graph

July 25, 2005  Maher Salah
**Feature-similarity view**

- The similarity matrix is computed from the call-graphs
  \[ F_k = G_k(V_k, E_k) \]
- Using relations (edges)

\[
\text{Similarity}(F_1, F_2) = \frac{|E_1| |E_2|}{|F_1| |F_2|}
\]

**Class protocol view**

- Terminology
  - Usage scenario: A description of how a class is used by other classes
  - Method invocation sequence: The sequence of invocations against an instance of a class (object).
- Problem
  - Determine the class-usage scenarios from its invocation sequences
- Approach
  - Compute the canonical set and canonical groups of method sequences.
  - Approximate each canonical group as a regular expression (Usage scenario)

**File class**

- Symbols
  - R = Read
  - W = Write
  - O = Open
  - C = Close

**Method Sequences**

- ORC
- ORRC
- ORRRRC
- ORRRRRRC
- OWWC
- ORWWWWWC
- OWWWWWWWWWC
- ORRRRRRRRRRC

TechReport: Client-side feature similarity
Class protocol view

- Canonical set
  - ORRC, OWWRC
- Canonical groups
  - P1 = {ORC, ORRC, ORRRC, ORRRRRRRRWC}
  - P2 = {OWWWC, WWWWWWWWWWC, ORWWWWC, OWC}
- Regular expressions
  - P1 = O(R)^1,2,3,4,10 \(W\)\(^{-1}\) C \(\approx O(R)^* (W)^+ C\)
  - P2 = O(R)^1,3,5,10 \(W\)\(^{-1}\) C \(\approx O(R)^* (W)^+ C\)

Computing Canonical Set/Groups

- Method sequences
- \(P = p_1, p_2, \ldots, p_J\)
- \(K_{\text{seq}}\)
- Compute the similarity graph \(G(P)\) using the similarity function
- Compute the canonical set that satisfies:
  - \(\sum d(s_i, s_j) < K_{\min}\)
  - \(\sum d(s_i, s_j) > K_{\max}\)
- Compute the canonical groups
- Formulate the BCS problem as an integer programming problem
- Use semi-definite programming (SDP) to find an approximate solution
- Compute Canonical groups

Similarity Measure

- Levenshtein edit-distance function
  - Distance \(d(s_i, s_j)\) = Least number of operations required to transform string \(s_i\) into string \(s_j\)
  - Operations: deletion, insertion, substitution
- Adaptation of edit-distance
  - Associating a variable cost with each operation
    - Frequency of a particular method invocation
      - Cost \(\propto \text{frequency/length}\)
    - Location of the particular method invocation
      - Cost \(= 1\) if \(s[i] \neq s[j-1]\)
      - Cost \(= \text{tiny}\) if \(s[i] = s[j-1]\)
**Similarity Measure**

- Examples:
  - $\text{InsertCost}(\text{"ORRC"}, \text{"R"}, k) > \text{InsertCost}(\text{"ORRRRC"}, \text{"R"}, k)$
  - ORRC has fewer Rs than ORRRRC
  - $\text{InsertCost}(\text{"ORRWWC"}, \text{"R"}, 5) > \text{InsertCost}(\text{"ORRWWC"}, \text{"R"}, 3)$

- Similarity function
  \[ \text{Similarity}(s, s_1) = \frac{1}{\text{wd}(s, s_1)} \]

**Computing Canonical Groups**

- Let
  - $P = \{p_1, p_2, ..., p_n\}$ - Method sequences
  - $P^* = \{q_1, q_2, ..., p_m\}$ - Canonical set
- For each $p_i$, compute its similarity to all elements of $P^*$
- Place $p_i$ in the group $g_k$, where
  - $\text{similarity}(p_i, q) > \text{similarity}(p_i, q'), \forall q' \in P^*, k \neq l$
- In case of a tie in max-similarity, place $p_i$ in all corresponding groups

**Observations/Limitations**

- MS can get long and complex
- MS may contain subsequences that may not be traceable to source code
- Overlapping groups
  - At this stage, our approach cannot produce a simplified set class usage scenarios.
- Analyzing closely related classes can produce simpler MS than if each class is analyzed individually.
  - Example: Socket, SocketInputStreams, and SocketOutputSteam
Socket class example

A<init>B setImpl
C getInputStream D getOutputStream
E getLocalAddress F getLocalPort
G getPort H getInetAddress
J close I postAccept
K setSoTimeout L setSoLinger
M setTcpNoDelay N getSoTimeout
O connect P isClosed

• Systems

<table>
<thead>
<tr>
<th>System</th>
<th>Platform</th>
<th>Size (KLOC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mozilla: Web browser (v1.0.1)</td>
<td>Win32/C,C++</td>
<td>4,410</td>
</tr>
<tr>
<td>Jext: Text Editor (v3.2)</td>
<td>Java</td>
<td>98</td>
</tr>
<tr>
<td>TechReport: Publication database</td>
<td>Java/EJB</td>
<td>2.3</td>
</tr>
</tbody>
</table>
Case study

• Views

<table>
<thead>
<tr>
<th>Interaction Type</th>
<th>Mozilla</th>
<th>Java</th>
<th>Task Report</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object-interaction</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Class-interaction</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Module-interaction</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Feature-implement</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Feature-similarity</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Remote-interaction</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Thread-interaction</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Class-usage protocol</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Example of a maintenance scenario

• Change request
  – Modify send-page feature of the Mozilla web-browser

• Developer task
  – Study the change request
  – Locate source code portions related to the send-page:
    • Option-1
      – Study the source code and documentation
    • Option-2
      – Instrument the application's code
      – Exercise the feature using a profiler
      – Analyze execution trace(s)

Mozilla: Send-page feature

- Run/Profile Mozilla
- Type feature name "send-page"
- Mark start of send-page trace
- Exercise send-page feature
- Mark end of send-page trace
- Load and analyze runtime data
Simplifying views

- The user can exercise enough features to identify common modules/classes
- **Common modules** are the application’s infrastructure and **are used by most features**
- Eliminating common modules/classes will result in simpler and manageable views per feature
With all modules/classes

Without common modules/classes

Outline

• Motivation
• Software Comprehension
• Architecture of Software Comprehension Environment
• Software Views / Case Study
  ➢ Related Work
• Conclusions

Related work

• Analysis of Multi-language applications
  – Web applications [Hassan02]
  – Integrated conceptual model [Kullback98]
• Analysis of distributed systems
  – BEE++ system [Bruegge93]
  – X-Ray system [Mendoca99]
• Analysis of program features
  – Concept analysis [Eisenbarth01]
  – Abstract system dependency graph [Chen00]
Outline

- Software Comprehension
- Architecture of Software Comprehension Environment
- Software views / Case Study
- Related Work
  - Conclusions

Conclusions

- Contributions
  - Development of a software comprehension environment that supports
    - Feature-driven profiling and analysis
    - Distributed profiling
    - Multiple languages (C, C++, Java)
    - SMQL language for retrieving and analyzing data
  - Software views
    - Feature-interaction, implementation and similarity
    - Remote-interaction
    - Thread-interaction
    - Class protocol view
    - Module-interaction
    - Class-interaction

- Opportunities for future work
  - Views
    - Visualization
    - New views
  - Data collection
    - Scripting languages
    - Profiling of popular frameworks: COM+ and .NET
  - Integration with common IDEs
Publications

- Class-usage protocol
  Salah M., Denton T., Mancoridis S., Shokoufandeh A., Vokolos F., 
  Scenarigrapher: A Tool for Reverse Engineering Class Usage Scenarios from 

- Analysis of large systems: Mozilla
  Salah M., Mancoridis S., Antoniol G. and Di Penta M., Employing Use-cases and 

- Feature analysis – Feature-interaction view
  Salah M., Mancoridis S. A Hierarchy of Dynamic Software Views: from object- 

- Architecture and analysis of distributed systems
  Salah M., Mancoridis S. Toward an Environment for Comprehending Distributed 
  Component-Based Systems, IEEE WCRE, Victoria, British Columbia, Canada, 
  November 2003.

- Profiling and dynamic analysis
  Souder, T., Mancoridis, S., Salah, M., Form: A Framework for Creating Views of 

Questions

http://serg.cs.drexel.edu/~msalah/sce/