ARTISAn:
Pluggable Framework for Program Understanding

Vladimir Jakobac
jakobac@usc.edu
Motivation

• A number of software clustering techniques fail to provide much rationale behind the recovered architecture
• Therefore, architectural recovery needs to be accompanied by a system understanding activity, which includes the use of semantic information before any syntactic dependencies are considered.
Hypothesis

- Insight by Perry & Wolf:
  Software Architecture
    Elements
      Processing (P)
      Data (D)
      Connecting (C)
    Form
    Rationale
The ARTISAn Framework

- **Purpose view**: high level functionality of individual elements and the structure of their composition.

- **Usage view**: regions of related elements based on their usage scenarios.
Clues and Initial Labeling

- **Clues**: design- and implementation-level constructs used to classify, label, and possibly collapse the elements of a system.

- Three clue categories:
  - domain-independent
  - domain-specific
  - application-specific

- Example of domain-independent clues and color-coding scheme that are used for labeling:

<table>
<thead>
<tr>
<th>Element type</th>
<th>Implementation and design constructs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processing</td>
<td>GUI, Threads, main() function</td>
</tr>
<tr>
<td>Data</td>
<td>No methods other than constructor(s), Serializable</td>
</tr>
<tr>
<td>Connecting</td>
<td>Socket communication</td>
</tr>
</tbody>
</table>
Clue Examples

```java
package Visualizer;
import java.net.Socket;
public class ClientHandler{
    Socket _socket;
    DataInputStream _instream;
    ...
}
```
Initial Labeling - Example
Rules and Propagation Labeling

• Propagation scenarios
  – A set of structural and interaction patterns defines possible relationships among elements.

• Propagation rules
  – Derived from propagation scenarios.
  – Used to propagate information obtained from clues from labeled to their neighboring elements.
  – Detection of potential rule conflicts.
Def-Use Analysis

- **Regions**
  - Parts of a system that are exclusively used by its other part(s) and those that are shared.
  - Comprise related system elements and serve as potential *software components*.
  - Determined by using a modified dominance analysis and notion of entry points.

- **Entry points**
  - Modules in which processing starts (e.g., a class with the `main()` method implemented).
  - Directly obtained from the initial labeling step or provided by the engineer.
Two Orthogonal Views - Revisited

Purpose view

- Each class of elements of a system with its own color.
- Labeled as P (GUI), D, C, or unlabeled in case there is not enough information.

Usage view

- The purpose of an unlabeled element might be surmised if the element belongs to the same region with labeled elements of a single type.
Tool support

- Integrated with Rational Rose.
- Imports a Java library **UML Interface**
  - Translates requests to and from Rose.
  - Two communication standards supported: Java and Rose’s native MS-COM interface.

- Input: a class diagram with all dependencies included.
- Output: class diagrams with labeled and grouped classes.
Conclusion

- The ARTISAn framework comprises replaceable components to accommodate the exact programming environment.
- The approach is twofold:
  - By determining the type of a system element, one can distinguish elements with application-specific functionality from those with application-independent functionality.
  - By providing the usage view scenario, one can continue with grouping related elements and providing the system's higher level of abstraction.
ARTISAn: Pluggable Framework for Program Understanding

Vladimir Jakobac
jakobac@usc.edu