Utilizing Clone Detection For Domain Analysis of Simulation Systems

WICSA/ECSA 2012

24/08/2012 Helsinki, Finland

Merve ASTEKİN
Istanbul Technical University, TUBITAK BILGEM
Turkey

Hasan SÖZER
Ozyegin University
Turkey

Motivation

- Systematic software **reuse** at the **architectural level**
  - increase software quality
  - decrease the development time and costs
- Increasing complexity and the number of projects makes it **costly to manually analyze** commonality/variability among different systems.
  - **tool support** becomes essential

Approach

- Utilizing **clone detection** tools for supporting domain analysis

- A case study based on **four industrial simulation software systems**
  - Examination of clone size, distribution and density both within each system and **across the four systems**
  - Identification of **commonalities** and reusable components
  - Design/refine a **reference architecture**

Software Code Clones

- *"Clones are segments of code that are similar according to some definition of similarity."*

  Ira Baxter, 2002

- There can be different definitions of similarity based on:
  - Program text
  - Syntax
  - Semantics
  - Pattern

Clone Detection

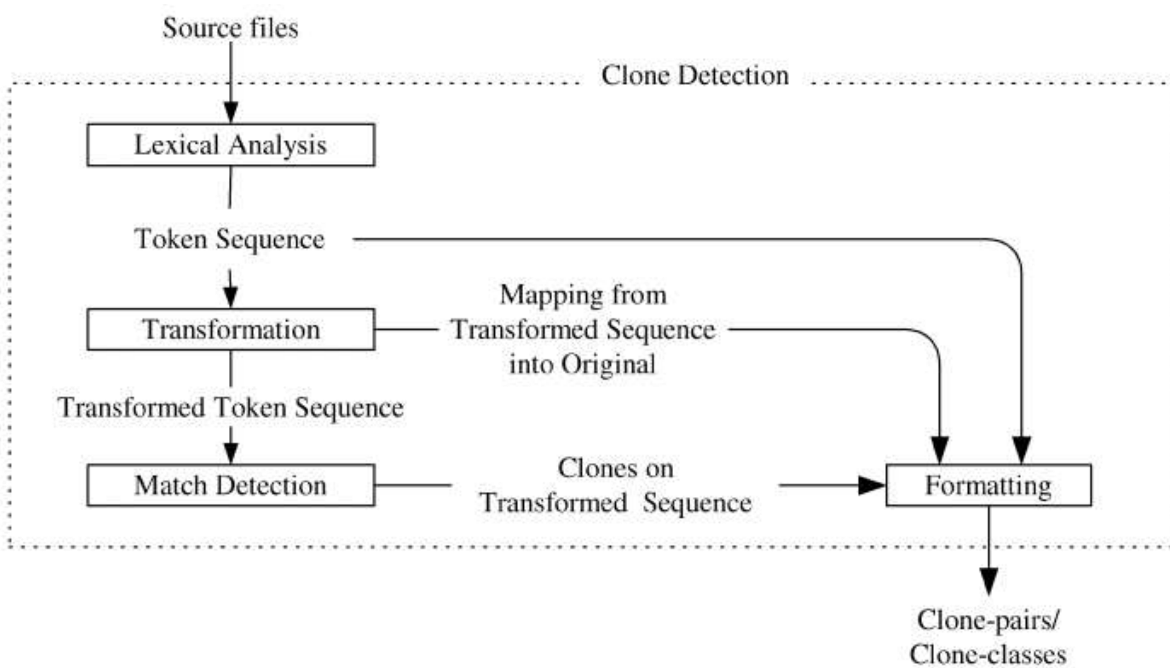
- Mainly applied for supporting reverse engineering and **refactoring**
- Usually applied to a **particular system** to identify and eliminate clones to improve maintainability
- Not utilized for detecting clones **accross systems** to identify commonalities
  - Previously proposed for supporting domain analysis and software product line development but not implemented
  - No such case studies have been reported yet

Clone Detection Techniques

- Many techniques available
  - Textual/Token/Metric Comparison
  - Abstract Syntax Tree Comparison
  - Program Dependency Graph Comparison
  - Other Hybrid Techniques
- In our study, we have used **CCFinder**
  - a token-based code clone detection tool
  - detects duplicated tokens in the source code
  - has a precision comparable to the other techniques
  - efficient and scalable

Clone Detection Process

- CCFinder – Token-based clone detection tool



Case Study: Simulation Systems

- We have studied four different simulation systems:

SIZE OF SUBJECT SYSTEMS

	Simulation Systems			
	Project X	Project Y	Project Z	Project T
Classes	1,440	1,317	10,877	2,012
Files	992	2,085	14,590	3,289
Lines	192,073	356,404	3,213,352	505,074

- Analysis on software systems
  - Different domain, architecture, size, development phase, development team, etc.

Analysis Process and Results

- Analysis in three steps:
  - I. Clone identification within each system
  - II. Clone identification accross different systems
  - III. Defining/refining a reference architecture

I. Clones within each system

1. Examination of the density of code clones.
2. Identification of the files that include most of the detected clones.
3. Examination of the distribution of code clones
4. Identification of code clones that scatter throughout most of the source files.
5. Manual identification and analysis of modules with high clone density to pinpoint.

Results: Clones within each system

- Scattered clones mostly related to initialization messages sent to the simulation engine infrastructure
  - modularized in an architectural layer in some projects
- In two HLA-based projects, the mostly scattered clones are related to the interaction creation and registration to federation sections of modules.
  - Federates shall interact with the runtime infrastructure for exchanging events among simulation entities, in compliance with HLA
- GUI component implementation structure leads to significant scattering of clones in all subject systems.

II. Clones accross different systems

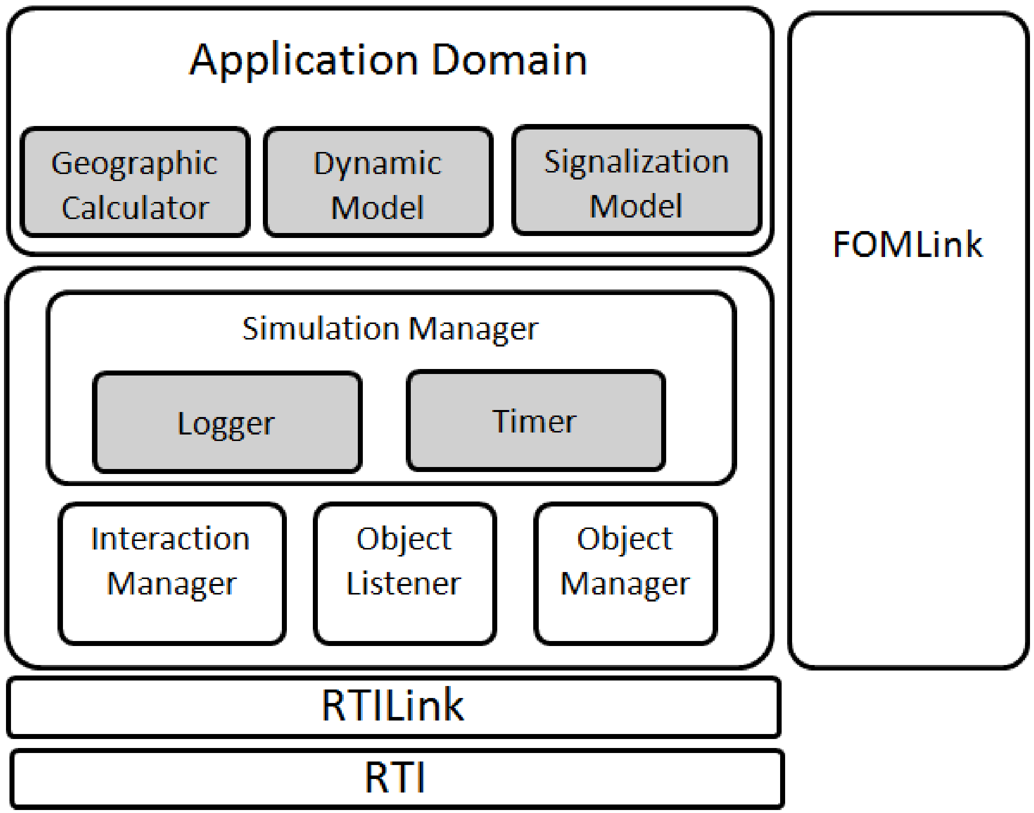
1. Analysis of the four subject systems in pairs
2. Intersecting clones are detected between the two subject systems
3. Manual analysis of module-based clone distribution to identify reusable component candidates across simulation systems
4. Definition/Refinement of the system architecture based on the analysis results

Results: Clones accross different systems

- **Duplicate implementation** of a domain-specific algorithm
- The highest cloning rate for HLA-based projects
- Clones **accumulated at only a small number of modules**
  - The dynamic model of the simulated system/environment
  - The signalization model functions of the domain components.
- Other common clones related to encoding/decoding rules
  - Rules necessary for the operation of the Federation Object Model (FOM) as part of HLA

III. Defining/Refining a Reference Architecture

- Reference Architecture with Identified Reusable Components



Defining/Refining a Reference Architecture

- The analysis of intersecting clones revealed various functionality that are reused routinely in simulation systems.
  - Timing and logging mechanisms in Simulation Manager layer;
  - Geographic calculator, dynamic model and signalization model algorithms of domain platforms in Application Domain layer.
- Refinement by identifying additional reusable elements.
- The validity and relevance of the results
  - confirmed by the **domain experts** and **software architects** who have worked on the subject systems.
  - confirmed by the **consistency** with existing architectural components reused e.g., *Simulation Manager, FOMLink, Object Manager, Object Listener and Interaction Manager*.

Summary and Conclusions

- A **case study** on utilizing **clone detection** for domain analysis of **simulation systems**
- Analysis of four industrial software systems
- Identification of a set of **domain concepts** and reusable components
- A **reference architecture** defined based on HLA
- Utilization of **clone detection** can be a viable approach for supporting **domain analysis** and definition/refinement of a reference architecture.

Future Work

- The effectiveness of other clone detection techniques
  - especially those that focus on program logic/architectural similarity will be investigated.
- Experimentation with more/different subject systems

THANK YOU!