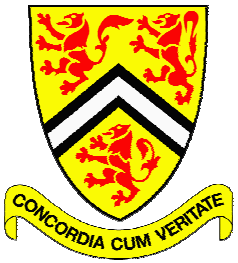
Connecting Architecture Reconstruction Frameworks



Ivan Bowman, Michael Godfrey, Ric Holt
Software Architecture Group
University of Waterloo

CoSET '99
May 17, 1999

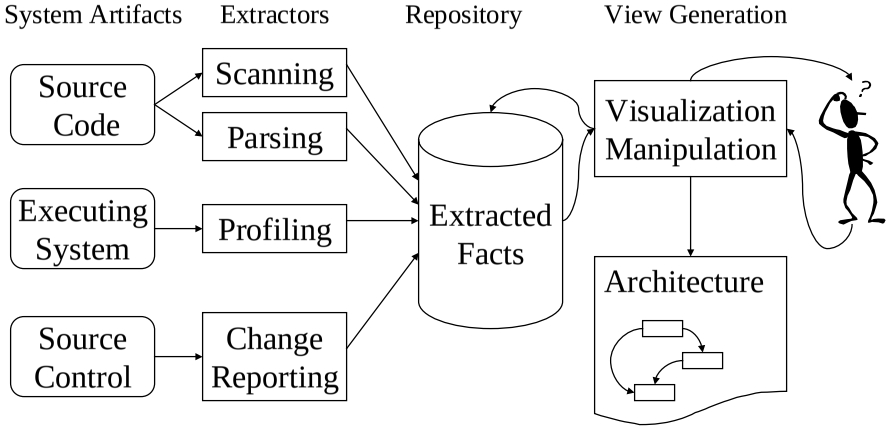
Introduction

- There is a wealth of re-engineering tools
- Often, we would like to combine these
- Differences in tool storage formats and semantics makes reuse difficult
- We defined an exchange format that can be used to combine several tools:
  - CIAO, Dali, Datrix, PBS, and Rigi

Background

- Recent work within CSER has identified opportunities for re-use between tools
- We identified two levels of tools:
  - *Code-Level* tools provide detailed support
  - *Architecture-Level* tools identify high-level abstractions
- We describe a format for connecting architecture-level tools

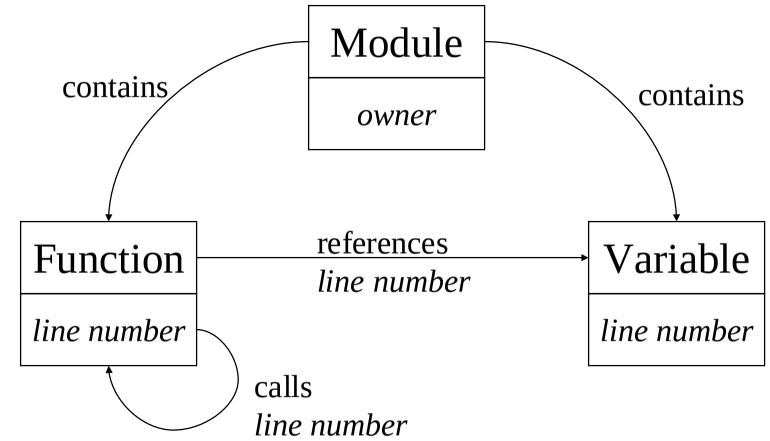
Architectural Reconstruction



Entity-Relational Model

- *Entities* represent source code elements such as functions, variables, or types
- *Relations* represent associations in source code such as calls or inheritance
- *Attributes* such as line-number record additional information
- A *Schema* defines the allowed entities, relations, and attributes

Example Schema



Connection Format Requirements

- Support multiple source languages
- Scale to large systems (e.g. 10 MLOC)
- Provide mapping to source code
- Support static and dynamic dependencies
- Incremental approach
- Must be extensible, allowing new schemes to be defined as needed

The Naming Problem

- Each entity must have unique ID
- Source languages may allow two code elements to have the same name
  - `typedef int T;`
  - `struct T { ... };`
- To combine facts, we need a common naming scheme
- We have defined a scheme for Java, and we are discussing possible solutions for C,C++

The Line-Number Problem

- We require a mechanism to get from an entity back to source code
- An obvious solution is to store file + line #
  - We want same file name on different machines
  - Some entities are defined on a range of lines, or non-contiguous ranges of lines (for example, namespaces)

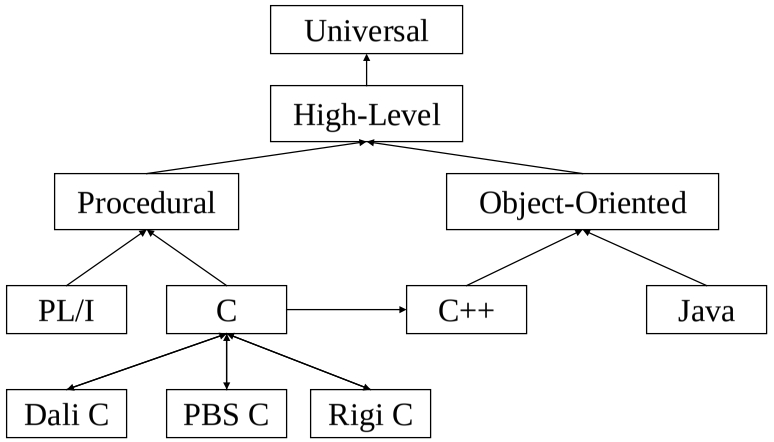
The Resolution Problem

- For each reference in source code, we can determine the reference *target*
- Several resolution strategies are used:
  - No resolution - each reference is an entity
  - Resolved to declaration (in a header file)
  - Resolved to static definition (entity body)
  - Resolved to dynamic definition (virtual functions, pointers)

TAXForm

- Idea: provide converters between tool-specific formats and a common format
- There are two parts to an exchange format:
  - Syntax of data (representation in files)
  - Semantic structure (schemas)
- We chose TA syntax (others are attractive)
- We allow tool developers to define their own schemas as needed

Transforming Between Schemas



Implementing TAXForm

- We evaluated TAXForm by implementing converters for several existing re-engineering tools
- The syntactic transformation was trivial
- We used Tarski relational algebra to specify transformations, and executed them with *grok*, a relational calculator

Evaluation of TAXForm

- Writing schema transformations required careful study of semantics used by each tool
- Tools used different terminology with different underlying assumptions
- By defining transforms between models, we formally document all of these assumptions and provide a dictionary for terminology

Systems Modeled

System	Size	Language	Tool
Jikes	77 KLOC	C++	CIAO
Linux	800 KLOC	C	Dali, PBS
Mozilla	904 KLOC	C	Rigi
Nachos	10 KLOC	C++	CIAO

Summary and Future Work

- We defined TAXForm as an exchange format between architecture-level tools
- We validated our format by implementing converters from existing tool formats
- We need to further describe the semantics of each model