High-Level Schemas:
A Journey through the Bush

Presented by Michael W. Godfrey

Software Architecture Group (SWAG)

Dept of Comp Sci, Univ of Waterloo

This presentation is available from
`http://plg.uwaterloo.ca/~migod/papers/`

What is a High-Level Schema?

My answer:

Any schema above the statement level

I see two distinct levels of abstraction:

1. Programming language entity level
   - Entities are fcns, (non-local) vars, types, classes, …
2. Architectural level
   - Entities are modules, subsystems, classes, interfaces, …

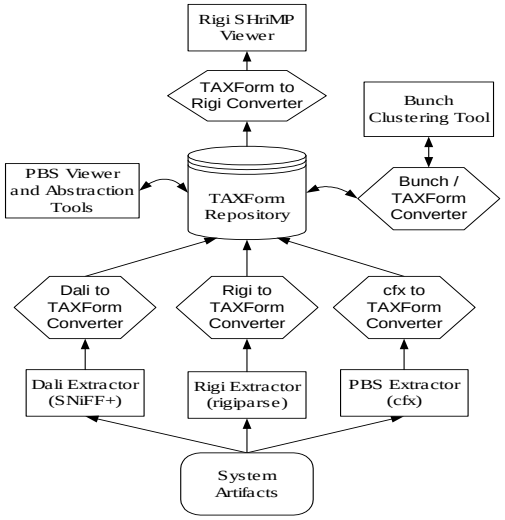
Previous Work

- Lots of
  - motivational work
  - *ad hoc* extractor snarfing
  - experimental translation mechanisms
- Examples (many others exist)
  - CORUM I and II
  - GRAX
  - TAXForm (TA eXchange FORMat) using Acacia, Rigiparse
  - Rigi using VA
  - Dali using Sniff+

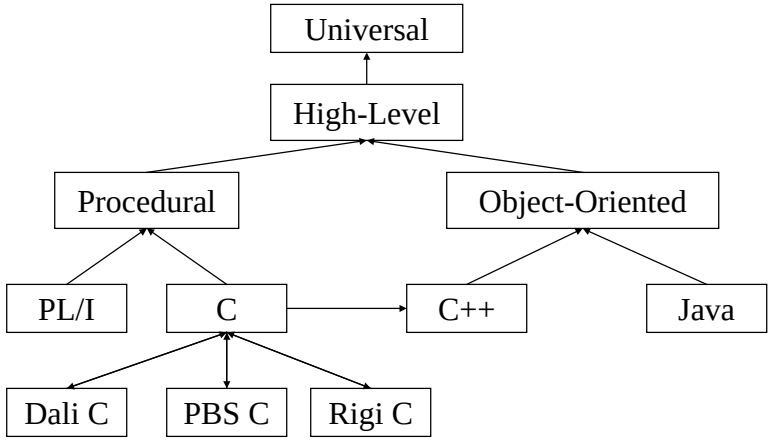
My (selfish) goals

- I would like to be able to use other extractors …
  - Want to perform architectural analyses of systems written in languages other than C
  - Want to implement *BEAGLE*
    (a tool for exploring software evolution)
- … but extractors differ in languages modelled, level of detail, robustness, bugs, data format, …
  - I want to be able to convert data between tools.
  - Need agreement (awareness) from tool creators

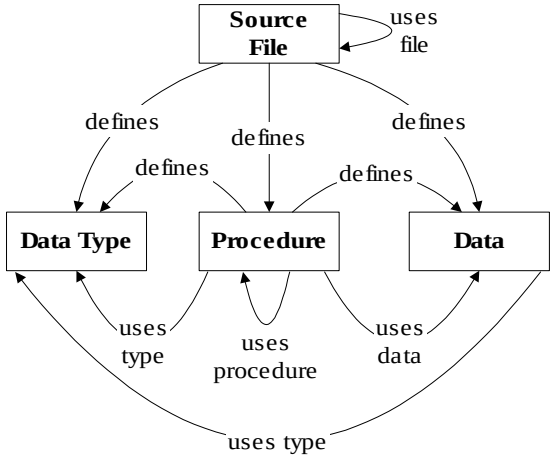
TAXForm Utopia



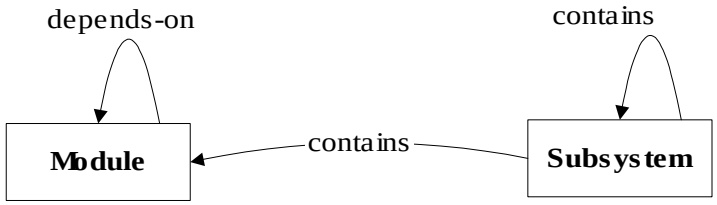
Transforming Between Schemas



TAXForm — Procedural schema



TAXForm — High level schema



Back to my (selfish) goals

- Would like to concentrate on procedural and OO languages.
  - Others are interested in COBOL, JCL *etc*.
- I am interested in *high-level* info `(f calls g)`
  - but not in ASGs, code-level metrics
- Need to agree on
  - Syntax
  - Level of granularity and detail
  - What to do in case of X *e.g.*, X = "missing files"

My schema wish list

[influenced by Acacia's C and C++ data models]

Top-level programming language entities:

- functions, variables, constants, type definitions (procedural languages)
- methods, class member data, static methods and member data (object-oriented languages)

Entity containers:

- files, modules, classes, packages

My schema wish list

Entity attributes:

- Name, unique identifier (UID -- see next section)
- UID of container, UID of containing file (if container is not a file)
- Signature/data type
- Line number information (see below)
- Declared scope/visibility, static or not, final or not
- Definition or declaration (see below)

Entity container attributes:

- name, UID
- relative path (if a file)
- version identifier (if provided)
- UID of container (if not a file), UID of containing file (if not a file)

My schema wish list

Relationships:

- Function calls, variable uses
- Line number information (see below)
- Container use/inclusion (by other containers)
- Inheritance (various kinds)
- "Friendship", various template relationships

Relationship attributes:

- Line number information (see below)
- Scope/permission of inheritance

Problems

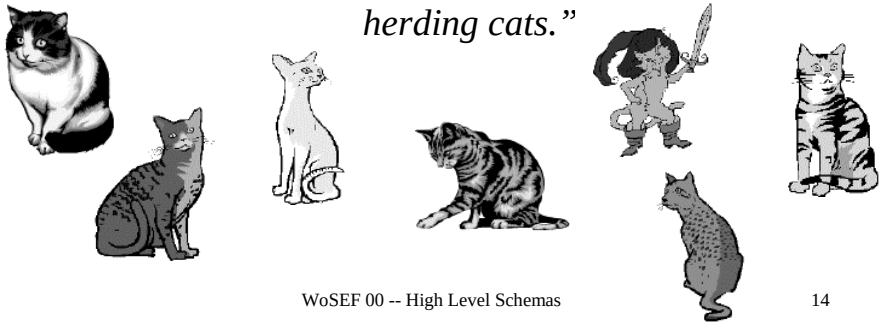
Some technical problems:

- UID generation? (name-mangling?)
- Line numbering (ranges)?
- Incomplete information?
  - ill-formed code, `gcc`/K&R-isms
  - missing header files
  - resolving entity use to dfn/dcl
    (esp. with polymorphism, overloading)
- Pre or post preprocessing?

Problems

We've had these conversations before …

"Getting academics to agree on anything is like herding cats."



Example Schemas

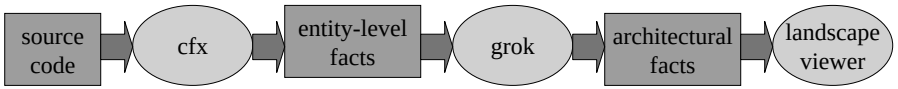
- PBS [UWloo]
- Acacia [AT&T]
- `cxref, ctags`
- TA++ [UOttawa]
- Rigi [UVictoria]
- SPOOL [UMontréal]
- BAUHAUS [UStuttgart]
- GUPRO [UKoblenz]
- SHORE [SD&M]
- Neuhold [UVienna]

Dimensions of Variation

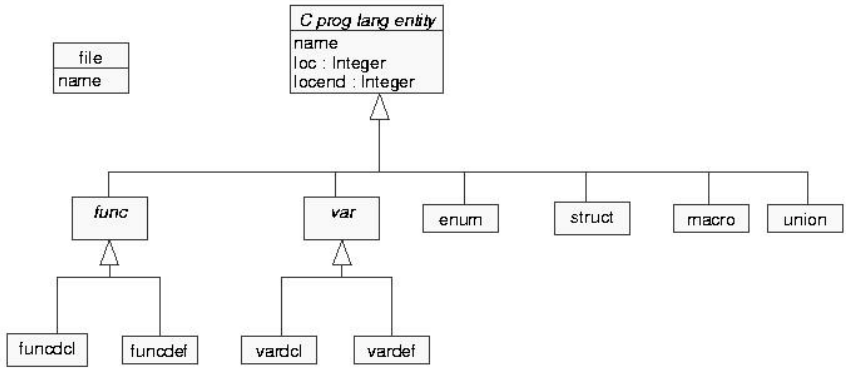
- Intended use
  - Level of schema (entity level vs. architectural)
  - Amount of detail
- Languages modelled
  - Multi-lingual
  - Common super schemas
  - Model "cross-overs" (*e.g.*, JCL, embedded SQL)
- Hidden assumptions
  - Known limitations
- Notation/approach to store factbase
  - Support for translations and transformations
- What's particularly novel and noteworthy

PBS [Holt *et al.* @ UWaterloo]

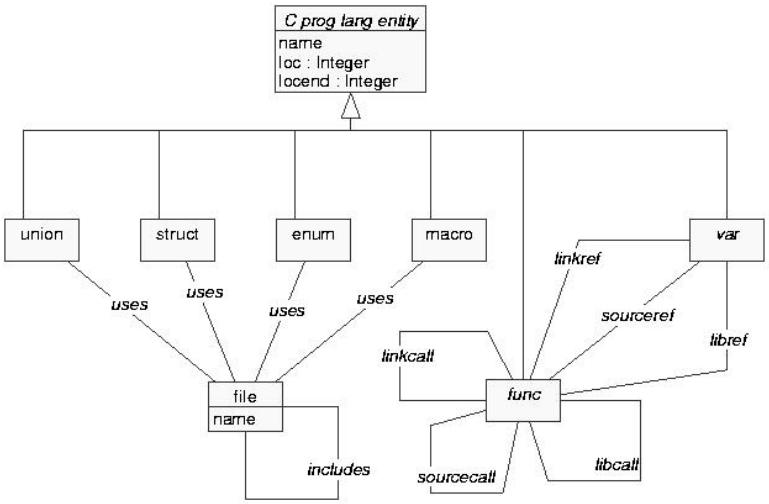
- Portable Bookshelf is a reverse engineering tool for creating software architecture models of large systems:
  - Guinea pigs: Mozilla, Linux, Apache, VIM, Mitel, TOBEY
- Consists of fact extractor, fact manipulation engine ("`grok`"), and visualization tool ("landscape")



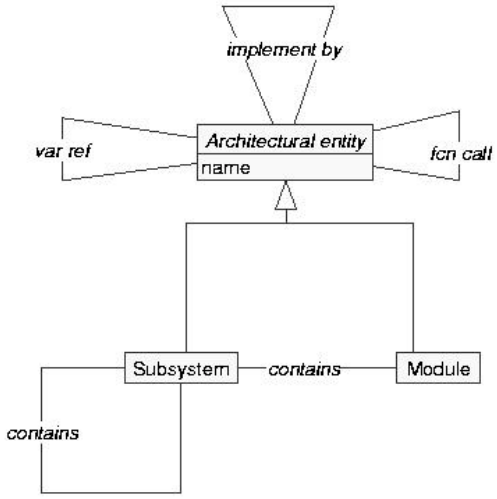
PBS C Language Entities



PBS C Language E/R View



PBS Architectural Schema



Acacia [Chen, Gansner *et al.* @ AT&T]

- History:
  - CIA → CIAO → Acacia
- Consists of
  - C and C++ extractors
  - SQL-like query engine
  - visualization with auto-layout

Acacia C++/C Schemas

- Entity attributes:
  - Hex UID, name, kind (file, function, type, var, macro), filename, datatype (string), typeclass (`enum`, `struct`, *etc.*), linenum info for def/dec, def/dec/undef, param list, template info, scope, storage spec (`static`, `const`, `inline`, `inline virtual`, *etc.*), signature
- Relationship attributes:
  - Linenum info, rel. kind (refers, contains, inherits, instantiates, typedef, *etc.*), relationship scope

Acacia Queries

- SQL-like queries for entities and relationships produces ";" delimited textual output:

```
% ksh cdef -u fu closeTagFile
26f53ece;closeTagFile;function;entry.h;void;regular;83;0;83;
   dec;00000000;(const boolean);;extern;;;;
76e7ae31;closeTagFile;function;entry.c;void;regular;551;553;
   563;def;00000000;(const boolean);;extern;;;;

% ksh cref –u - - - m file2='osdeps.h'
<all entity1 attrs> ; <all entity2 attrs > ; <rel attrs>
```

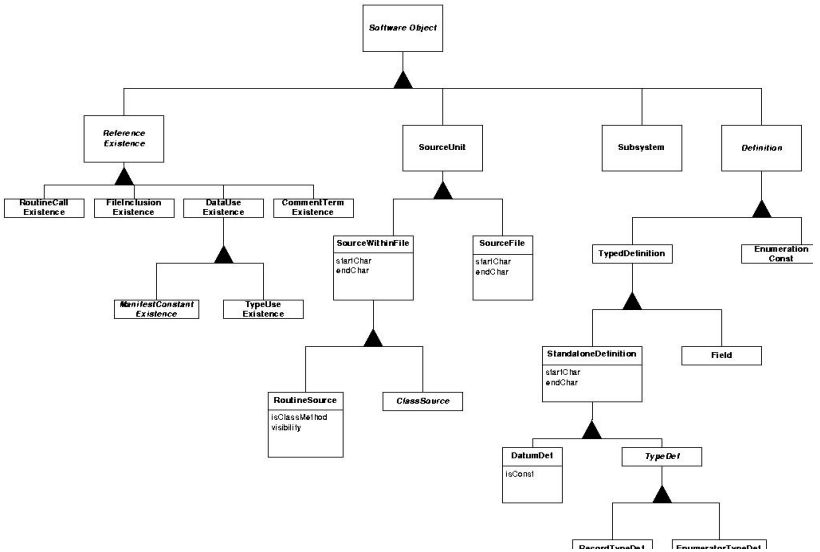

`ctags, cxref, cscope`

- These are "open source" Unix tools that perform extractions:
  - `ctags` extracts only entity info
    - *e.g.*, file, name, line num, kind, *etc*
    - works with C, C++, Eiffel, Fortran, and Java.
    - Used for fast context switching while editing source code with `vim/emacs`
  - `cxref` generates cross-reference table for C systems.
    - Often used for webifying source code (*e.g.*, Linux, Mozilla).
  - `cscope` used for program comprehension of C systems (*e.g.*, *who calls* `f`, *who uses* `v`)
    - Older commercial Unix tool, recently open sourced.

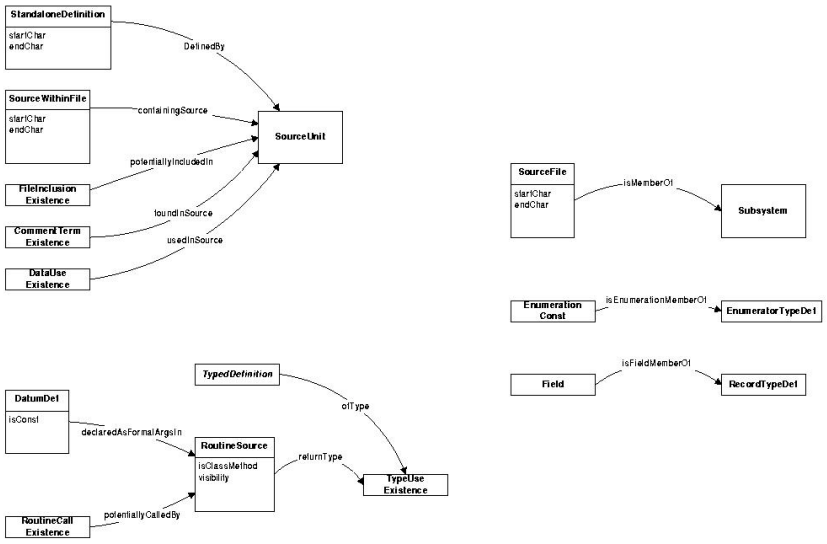
TA++ [Lethbridge *et al.* @ UOttawa]

- TKSee aids *programming comprehension*
  - *i.e.*, what programmers do all day
  - TA++ is the data modelling language
- Want "full story" from the source code:
  - Want pre-preprocessing view of code for all platforms and environments *(text editor's view)*
  - … but most extractors use a compiler front end and preprocess toward a particular target and option set
    - Some extractors keep some macro info

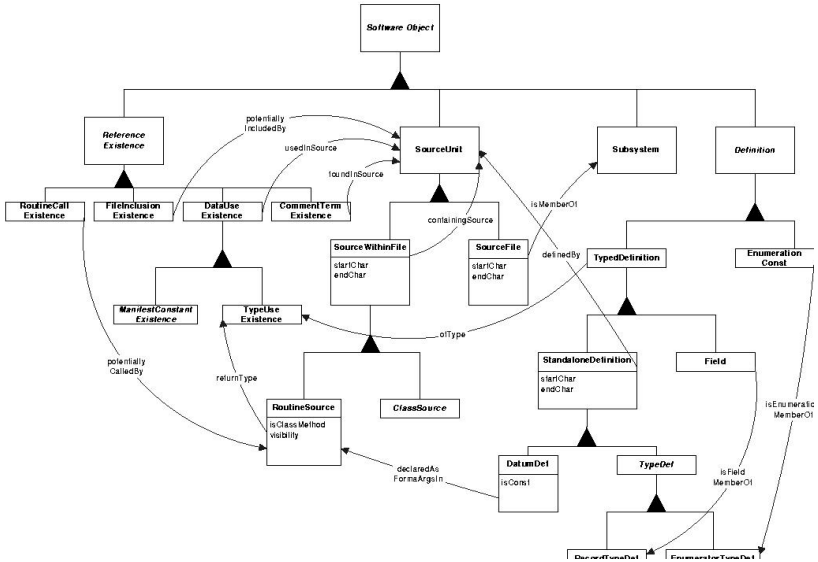
TA++ Entities



TA++ Relationships



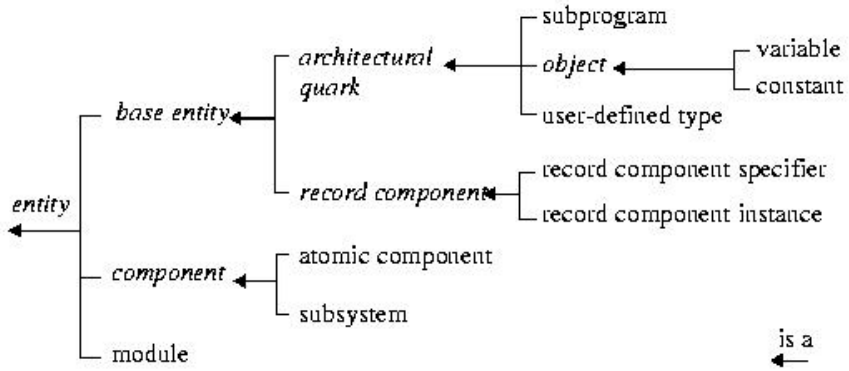
TA++ Combined E/R Model



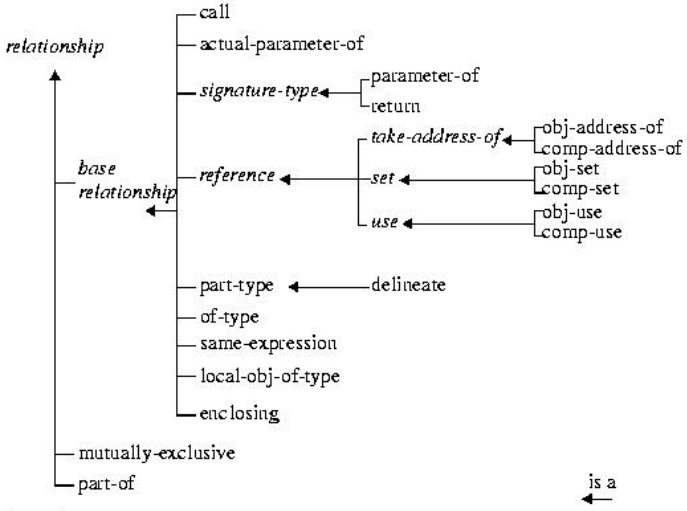
BAUHAUS [Koschke *et al.* @ UStuttgart]

- Software architecture recovery system
  - Parse code, look for hidden/decayed abstractions, then redesign
  - Uses various heuristics to perform “clustering”
  - Works both at entity level and subsystem level
- Built from many tools …
  - … including Rigi viewer and a customized C parser/extractor that (optionally) dumps RSF
- Example WoSEF problem:
  - Cannot derive full `includes` hierarchy from Bauhaus extracted facts; this was a design decision, as the researchers were not interested in this information

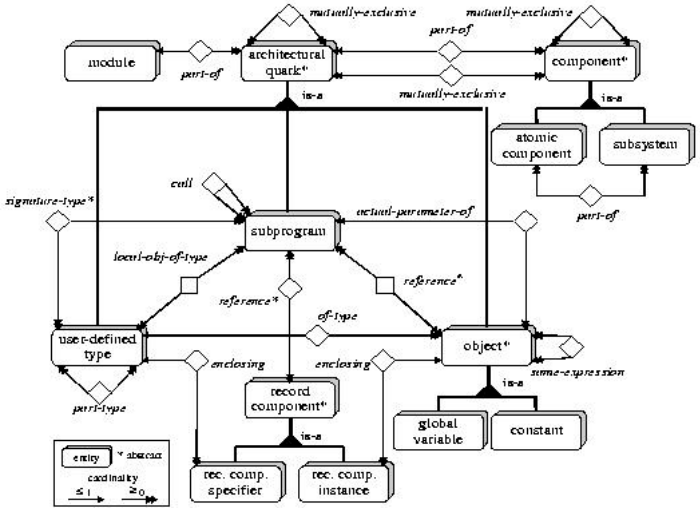
BAUHAUS Entities



BAUHAUS Relationships



BAUHAUS Combined E/R



GUPRO [Ebert, Kullbach, Winter *et al.* @ UKoblenz]

- GUPRO supports simultaneous modelling of inter-related systems written in different programming languages
  - In particular, concerned with the COBOL/MVS/JCL mainframe world
- GUPRO is notable because:
  - Simultaneously multilingual
  - Explicitly models “boundary crossings” (!)
  - Looks at (very real) problems of the mainframe world
    - COBOL, JCL, database migration

GUPRO

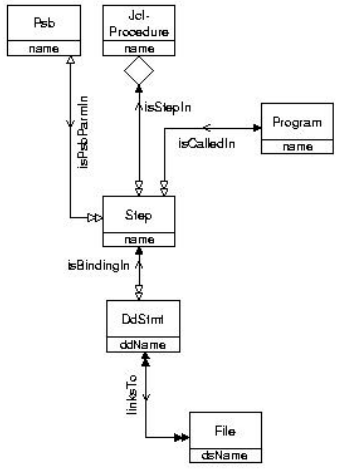
- Candidate system is modelled in an object-based repository using a graph-based approach:

  EER (modelling language)

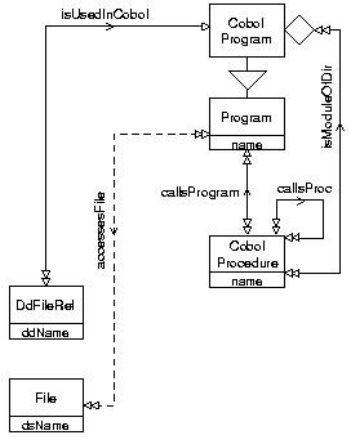
  \+

  GRAL (constraint language)
- GReQL mechanism supports structured queries on the repository via restricted first-order logic

GUPRO

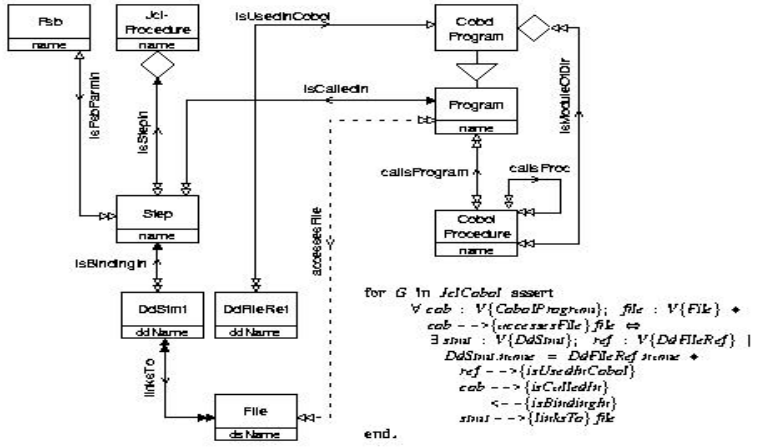


JCL schema



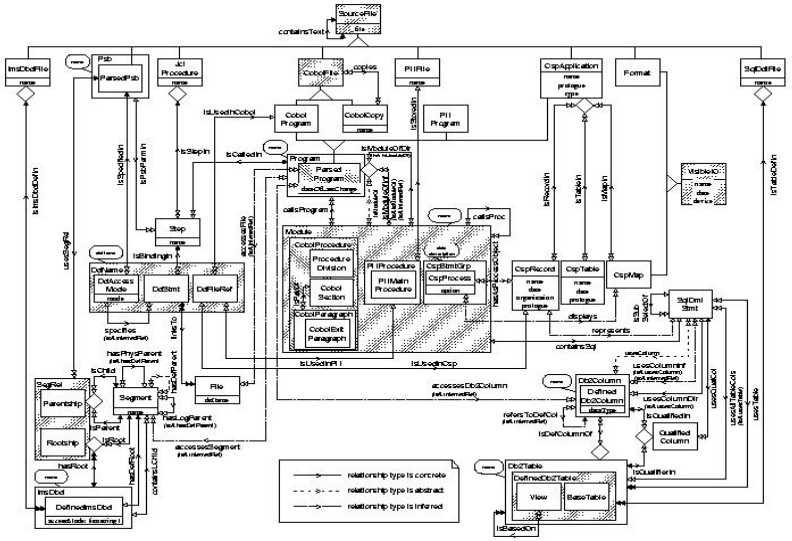
COBOL schema

GUPRO



Integrated schemas for JCL and COBOL

GUPRO Multi-Language Model



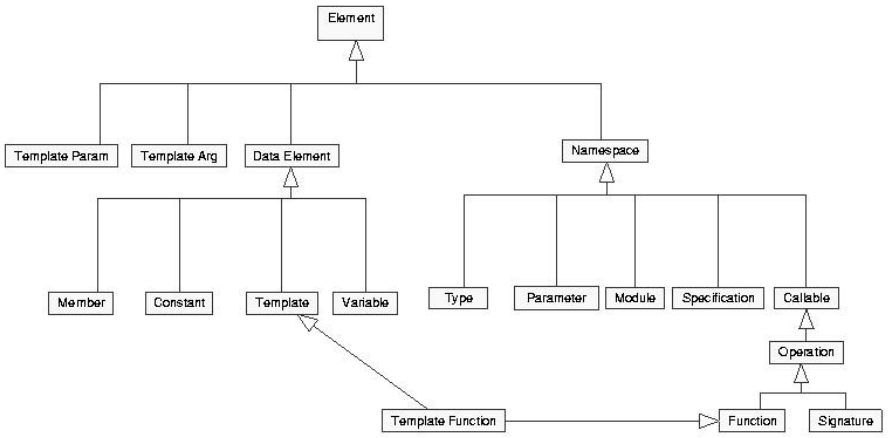
SHORE [Hess *et al.* @ SD&M]

- SHORE is a web-based repository that stores information extracted from structured documents
  e.g., XML-ified source code, reqs spec
- Uses "layered meta model" to integrate different programming languages
  - Has language independent meta model plus specializations for Java and COBOL models
  - Has parsers (XML-ifiers) for Java, COBOL

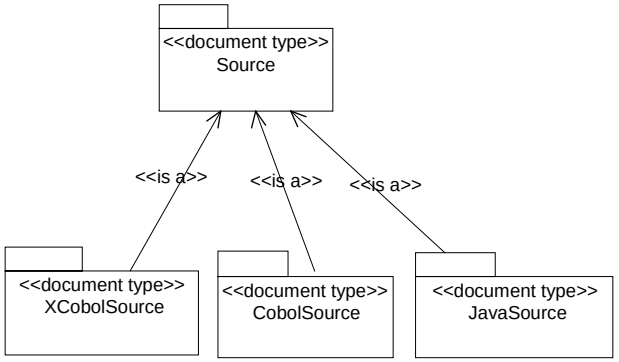
SHORE

- Their current schemas are "high-level", but they propose that a future exchange format should model:
  - all AST-level (structural) info
  - all semantic analysis info
- Not clear (to me) how entity resolution is done (name-based?):
  - seems to assume a tree-based definitional/structural view of the code

SHORE Entities



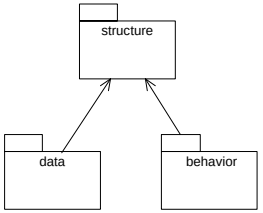
SHORE Prog. Lang. Metamodel



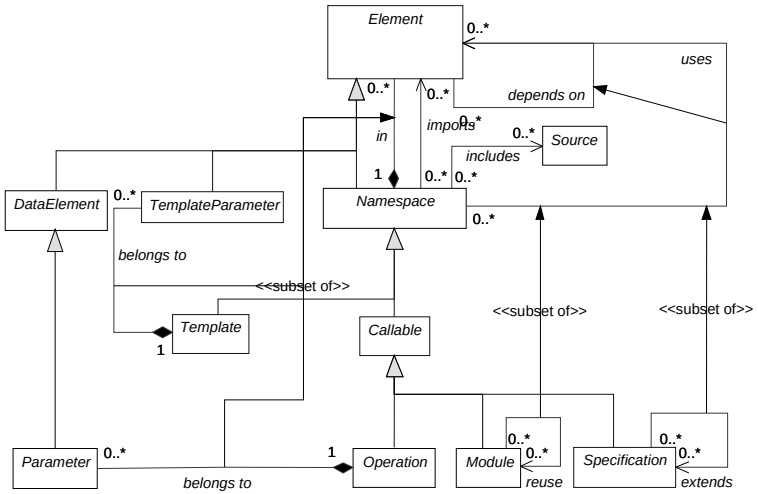
SHORE

Common Meta Model for Programming Languages

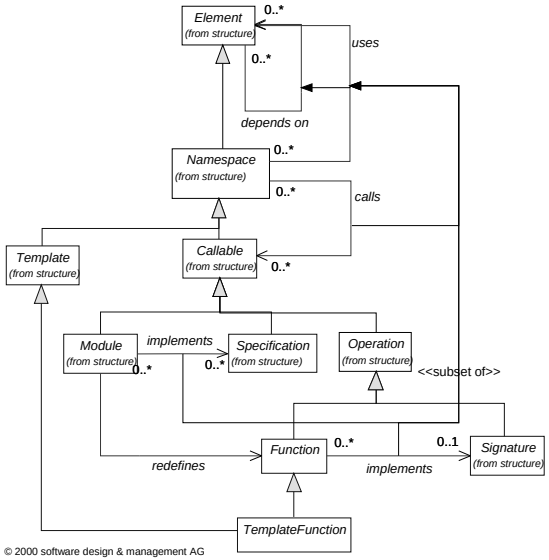
Procedural Programming Languages



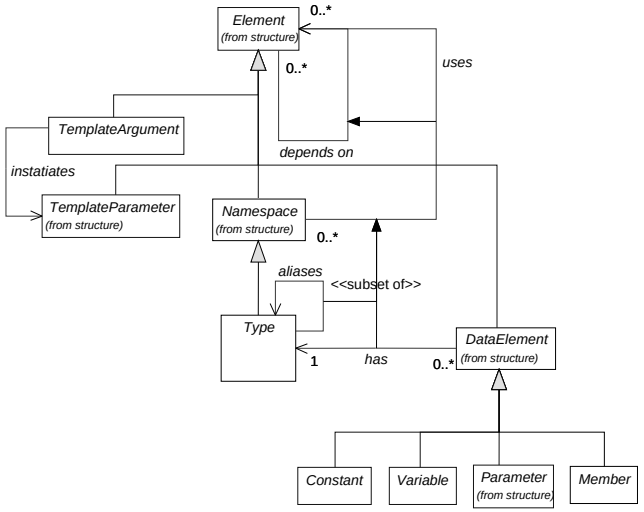
SHORE Entity Structural View



SHORE Static Behavioural View

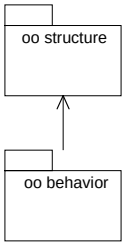


SHORE Data View

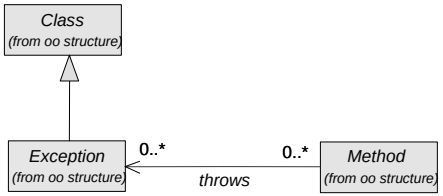


SHORE OO Metamodel

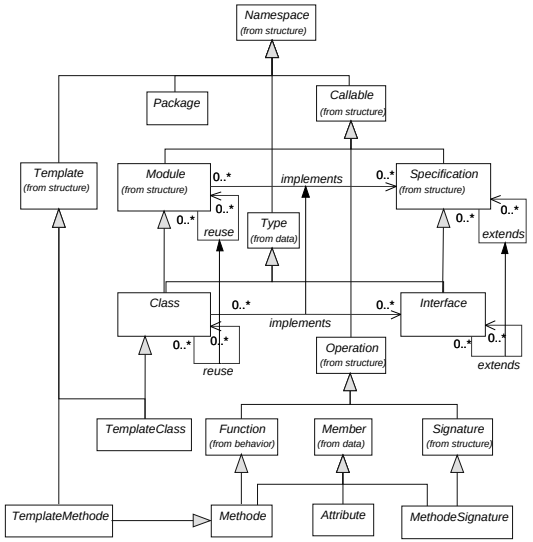
Common Meta Model for OO Languages



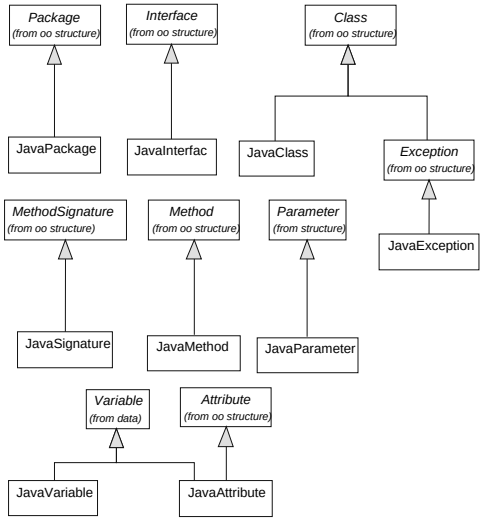
Behaviour of OO Languages



SHORE OO Structural View



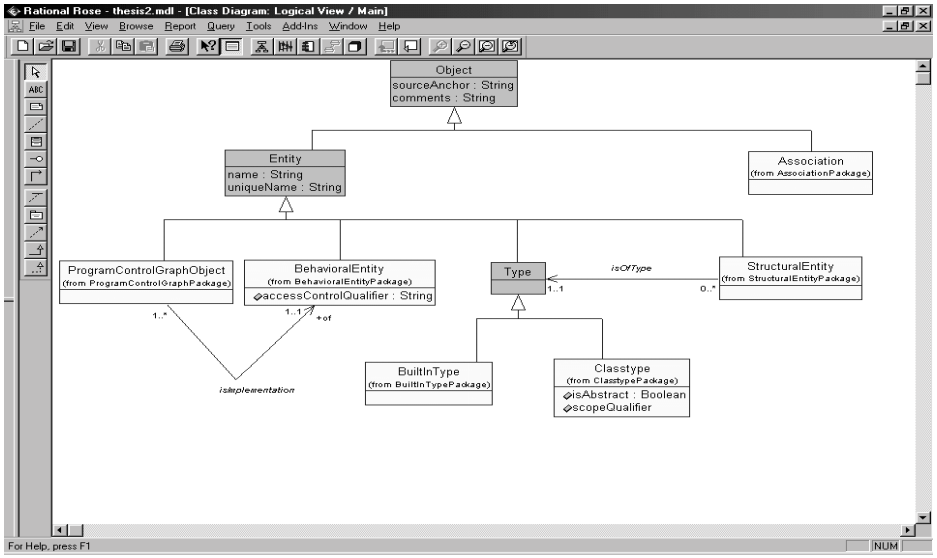
SHORE Java Schema



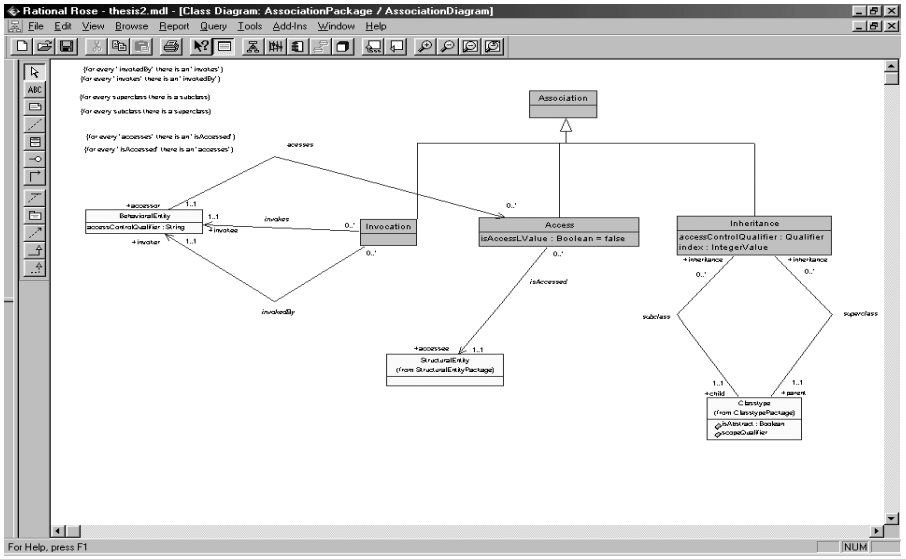
Neuhold [Karin Neuhold @ UWien]

- Consists of parsers + repository + metrics engine
- Interested in applying OO metrics to code
  - Concerned with “statement level” of detail, but some “flattening” was performed.
- Have parsers for several OO languages …
  - C++, Java, Delphi, Smalltalk
- … but wanted a single meta-model (repository schema) that would be as language independent as possible.
  - Some language-specific specialization allowed in repository

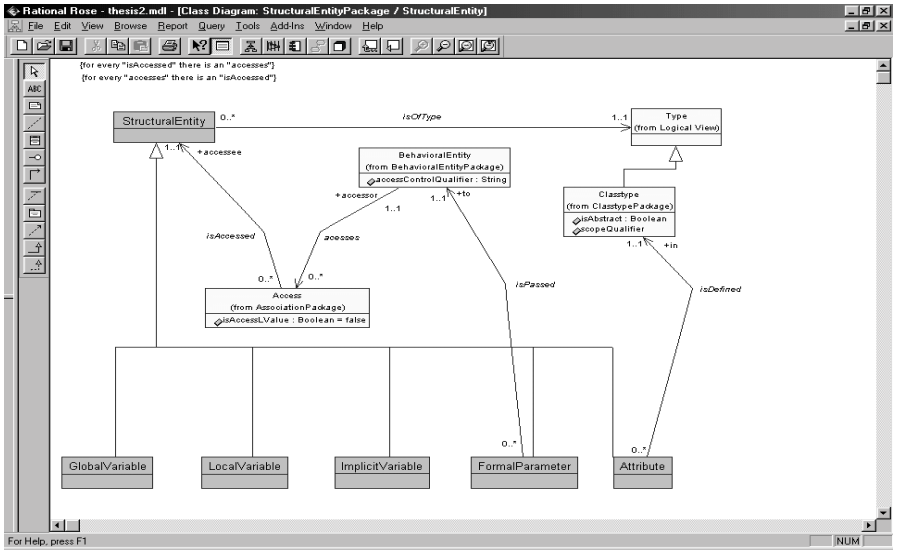
Neuhold



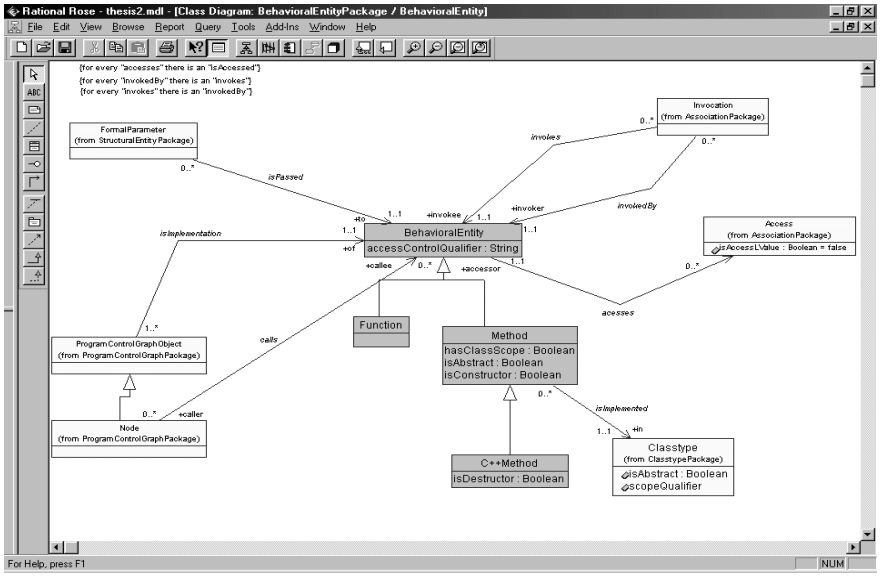
Neuhold



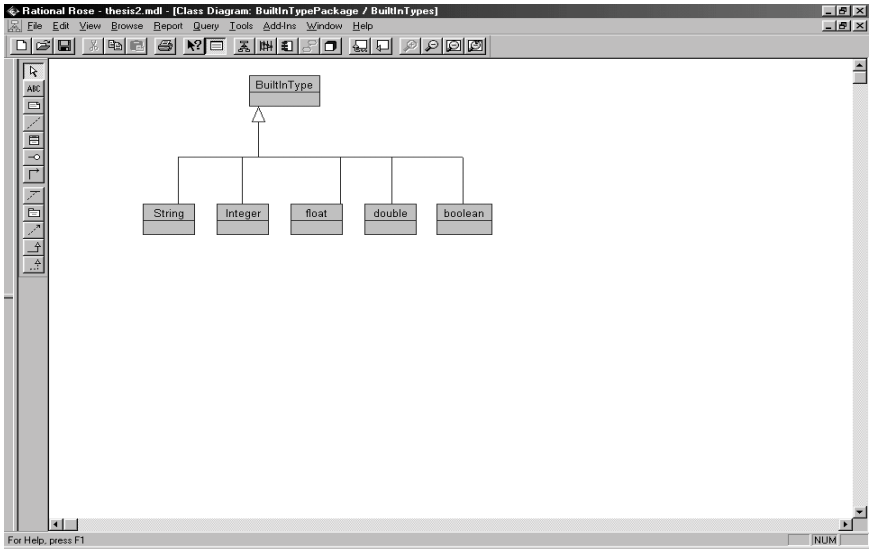
Neuhold



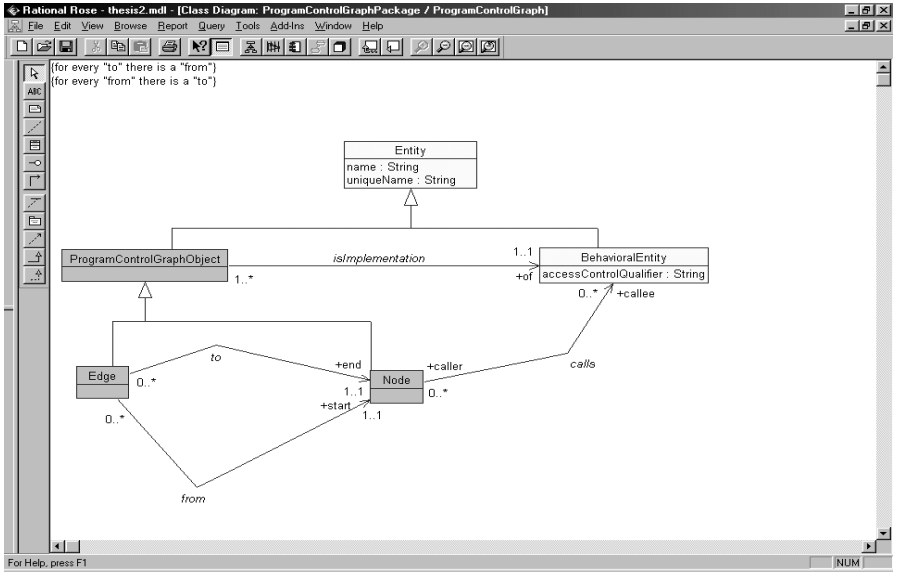
Neuhold



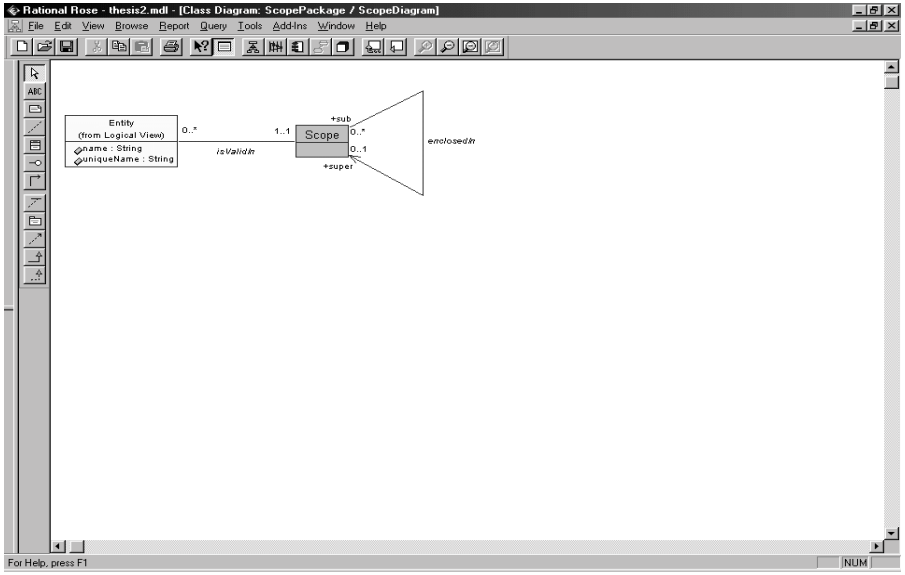
Neuhold



Neuhold



Neuhold



Summary — High-Level Schemas

- Lots of sticky issues at the prog. lang. level:
  - To pre- or not to pre-process
  - Entity resolution often not done
  - What is a function: def, dec, polymorphism, overloading, templates, …
  - How to deal with missing libraries, incremental extractions, versioned extractions, non-ANSI-isms, …
- Conceptual gaps:
  - COBOL/JCL world very different from C/C++/Java world
  - *"I didn't know you wanted full* `includes` *info…"*

Summary — Good News

- Many of us seem to be doing similar kinds of extractions. It seems like that:
  - Many extractors *can* be used within other tools
  - Some form of common interchange format *is* feasible
- Challenges:
  - May want to use multiple tools together
    - I am working on a standalone `cxref`-based hack to add full `includes` information to a BAUHAUS converter
  - Can we take advantage of the web to set up some sort of distributed fact extraction/conversion factory?

Q: Are you game?