The Dagstuhl Middle Model: An Overview

Timothy C. Lethbridge
SITE, University. of Ottawa
tcl@site.uottawa.ca

Motivation

Information interchange is necessary among reverse engineering tools

- So researchers/companies ca
  - Use each others' tools
  - Interpret each other's data

Therefore we need one or more metamodels

- (better if we can integrate and have just one metamodel)

We will use the term metamodel and schema interchangeably

Various metamodels that have been used

- CDIF (obsolete now)
- UML metamodel and the MOF
- TA / RSF schemas
  - In the Canadian research community
- Famix
- Others

Typical problems

- Too proprietary
- Too complex
- Don’t meet all reverse-engineering requirements

Requirements not met by the UML metamodel

Effective handling of source-code level information such as:

- Full procedure call hierarchy
- Full details of variables and access information
- Macros and other source code constructs

The UML metamodel focuses on things representable by UML

- i.e. designed for forward-engineering

Main types of metamodels for reverse engineering

Low-level (Abstract Syntax Graph)

- Specific to a programming language
- Can represent full details of code, allowing almost unlimited types of analysis

Mid-level

- Relationships among program entities
- More compact than low-level

Architectural

- E.g. components, pipes and filters

Database

The Dagstuhl Middle Model (DMM)

Probably should from now on be called the Dagstuhl Middle *Metamodel*

Initiated at the Dagstuhl Seminar on reverse engineering tool interoperability

- Held Jan 22-26, 2001
- Involved about 45 people
- Ratified GXL 1.0
  - A syntactic representation for schemas
- Started work on the four types of schemas

DMM background continued

- Represents program and source elements and their relationships
  - In a language-independent way
  - Suitable for typical OO and procedural languages
    - C, C++, Java, Fortran, Cobol, etc.
  - Extensions needed for languages with special features
    - E.g. Logic languages, scripting languages
- Key initial contributors and inputs
  - University of Ottawa (TA++)
  - Sander Tichelaar; Berne (Famix)
  - Erhard Plödereder; Stuttgart (Bauhaus)
  - others
- Web site: http://titan.cnds.unibe.ch:8080/Exchange/2

Design issues

On the next few slides we will look at some of the issues considered when designing DMM

Most of these issues were resolved in the direction of providing

- Simpler models
- That capture adequate information
- Enabling the reverse engineer to navigate objects and relationships in the system
  - Once they have found information of interest they may look at code or more detailed models

Design issue 1

Should you be able to regenerate the original code given a middle model?

Decision: No

- That would yield models of too great complexity for our target audience
- But this does not preclude a parser that generates a low level model, coupled to a tool that ‘lifts’ the data to a middle model

Design issue 2

Should a model represent original code or pre-processed code?

- Options:
  - Original cod
    - More useful for browsing tools
    - What the reverse engineer really thinks about and needs to modify
    - Can model pre-processor directives + macros
  - Pre-processed
    - Much easier to parse

Decision: Represent original code

Design issue 3

How language-independent should the metamodel be?

Decision:

- It should be able to abstractly represent all common relationships among elements
- Even though in different languages …
  - … the elements may have different names
  - … the elements may have subtly different semantics

Examples of language-independence-promoting abstractions

- The notion of type of a variable is preserved
  - But different types of pointers and storage can be abstracted out
- Procedures, functions and methods can all be treated similarly
- Classes, records and structures can be treated similarly
- Etc.

Design issue 4

Should all references be resolved?

E.g. exactly what procedure is called by what procedure call

Resolved references

- Not possible in the general case due to function pointers, dynamic binding
- Even in the non-OO case, requires full understanding of language-dependent linkage rules
- Often dependent on makefile

In practice, software engineers can understand most aspects of code without resolved references

Design issue 5

What level of detail needs to be captured?

- Assignments, control constructs?
  - No – that is for low-level ASG metamodels
- Local variables?
  - Not normally at the middle level
- Exact position of all references? (e.g. calls)
  - Existence of references in some object may be all that is needed
- Enough detail for full data flow analysis?
  - No; only enough for limited DFA

Design issue 6

Do we separate representation of source entities from representation of conceptual entities?

Source elements are chunks of source code

- Needed for tools to point reverse engineers to where conceptual entities are mapped to code
- Also useful for representing source constructs such as macros, conditional compilation, etc.

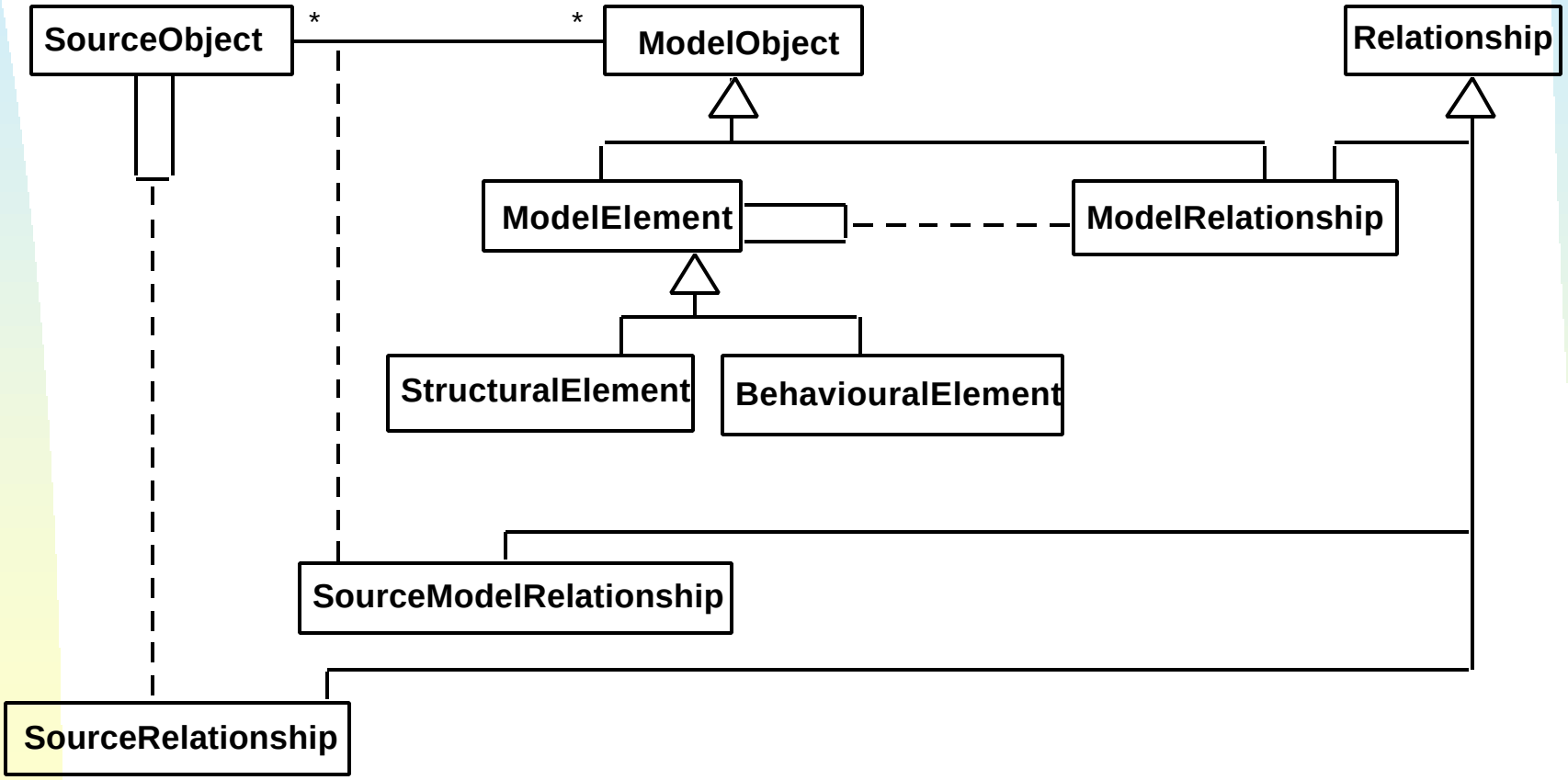
Such a separation has proved useful, but may not always be needed

How to model DMM?

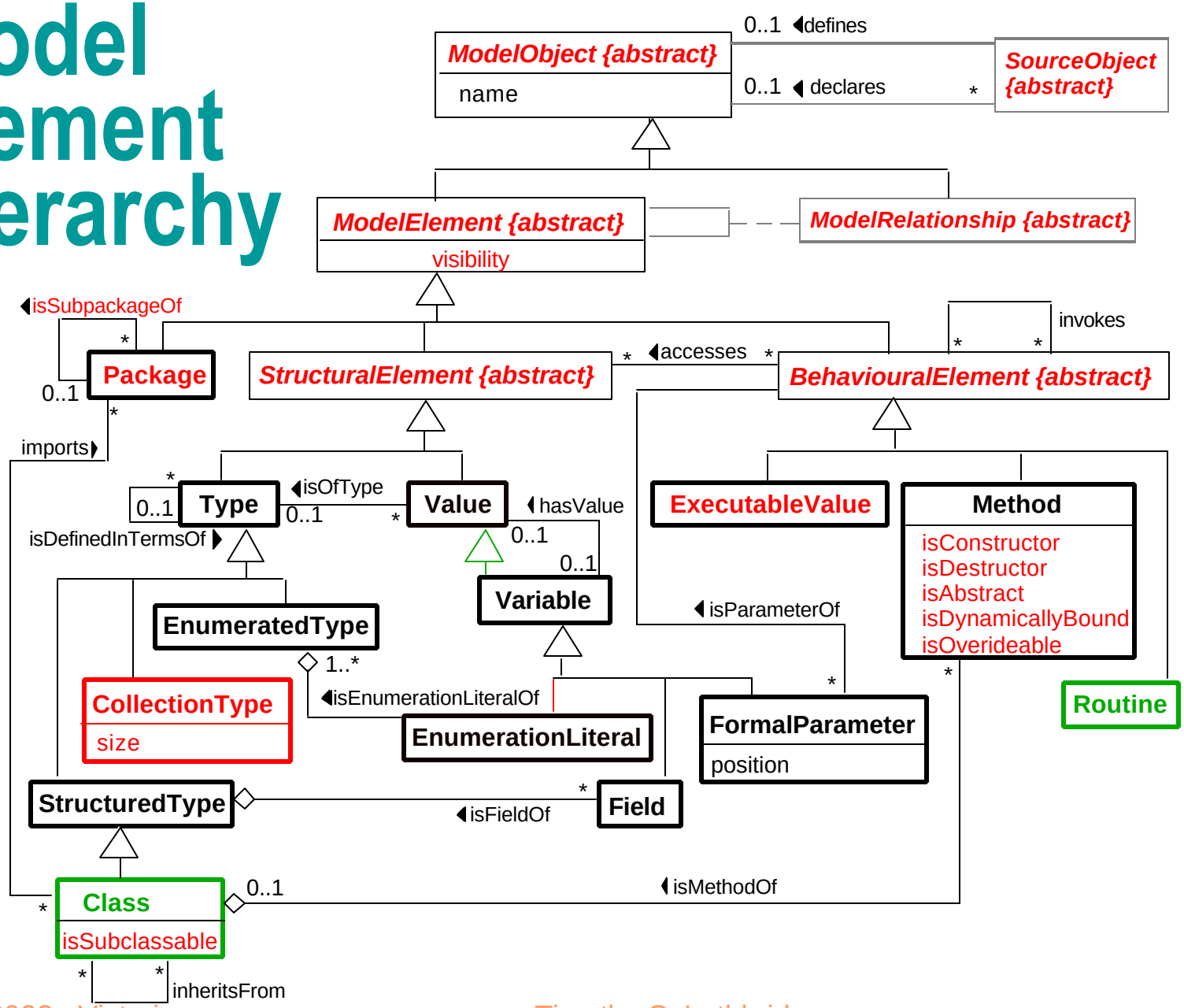
We follow the OMG convention of modeling DMM using a class diagram

Actually we use the subset of class diagrams called MOF (Meta Object Format)

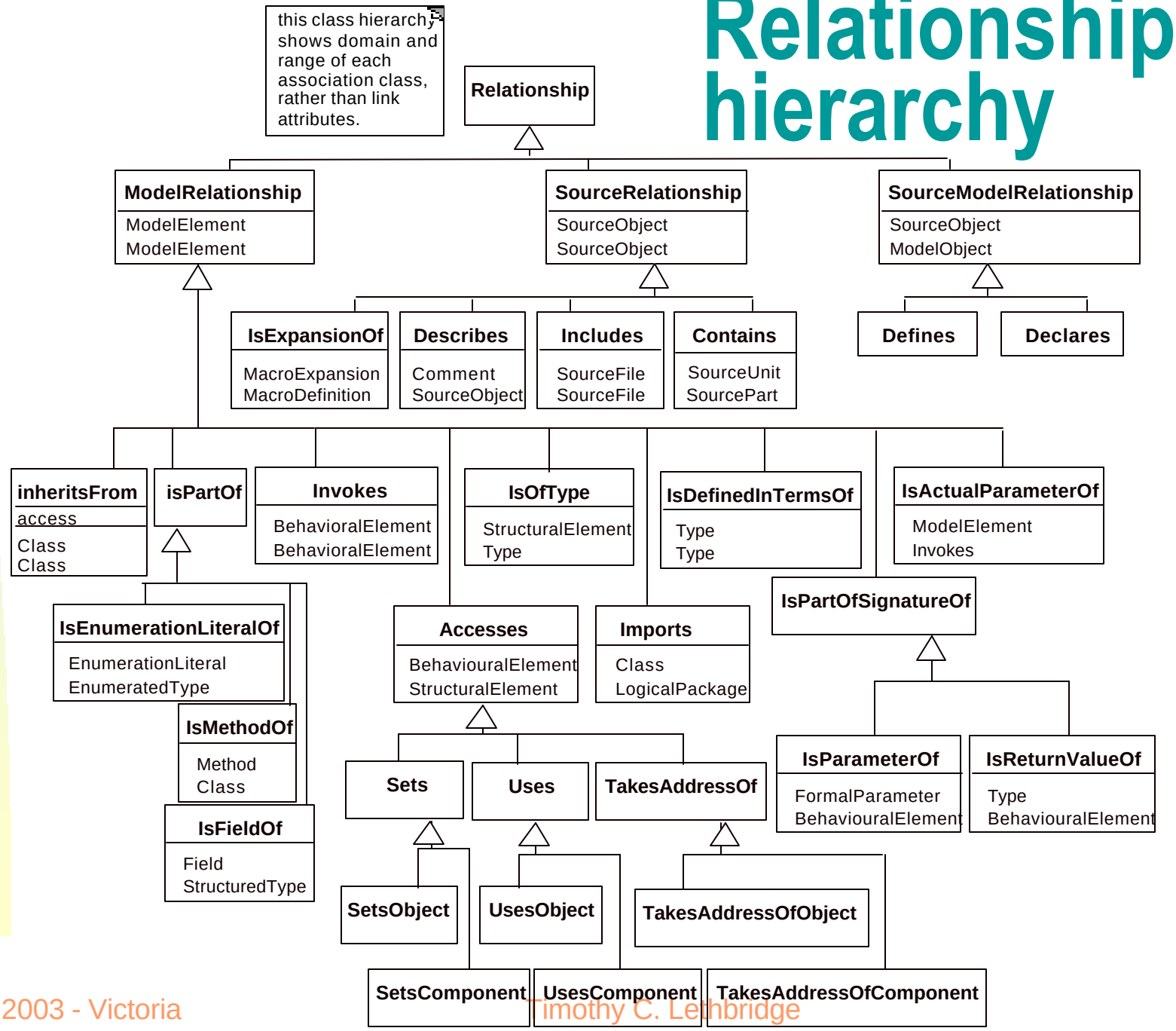
Top level of the hierarchy



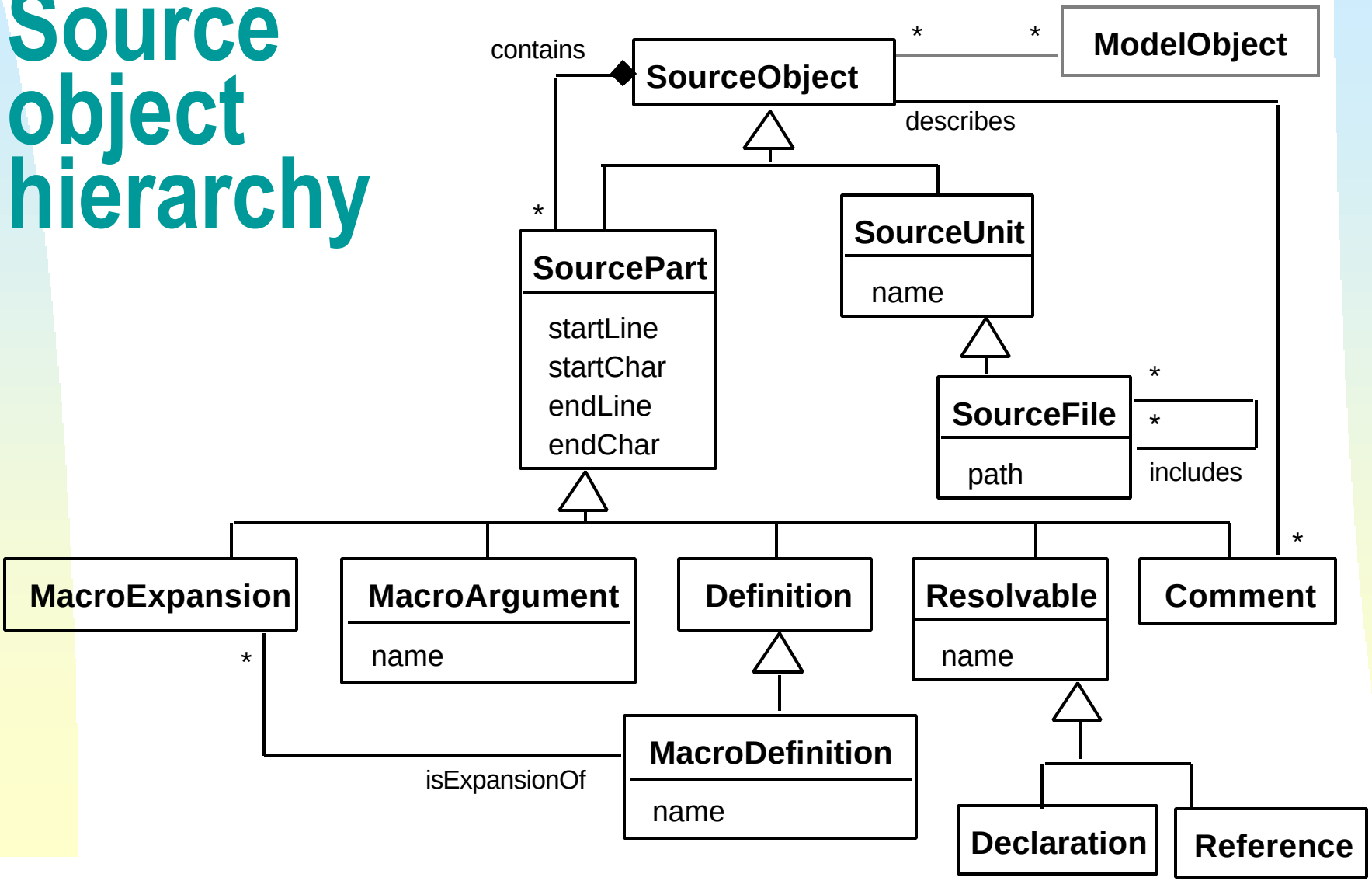
Model element hierarchy



Relationship hierarchy



Source object hierarchy



Issues not handled by DMM that may concern reverse engineers

- Multi-part references to members
  - E.g. in the expression a.b().c
    - How
- Function pointers
  - Reverse engineers we have worked for really have wanted to understand what can call what via function pointers
- Computed and aliased references
- Templates, à la C++

Syntactic carrier neutrality

DMM can be represented using:

- GXL: See http://www.gupro.de/GXL/
- TA
- XMI
- RDF
- Any other format capable of representing instances of classes and their associations

A tool can use whichever it likes, provided there are appropriate syntax translators

DMM Development Status

General website

- http://scgwiki.iam.unibe.ch:8080/Exchange/2.

University of Ottawa

- Have C++ parser that generates DMM in GXL
- URL: http://www.site.uottawa.ca:4322/dmm

Others are also experimenting with model

Future work

- Many details to be decided based on initial experiences

Topics for discussion

How useful is DMM as it stands? Is anything clearly inadequate? What changes, variants or extensions are needed?

What other middle level metamodels are there and can we merge the ideas from these to create a common standard

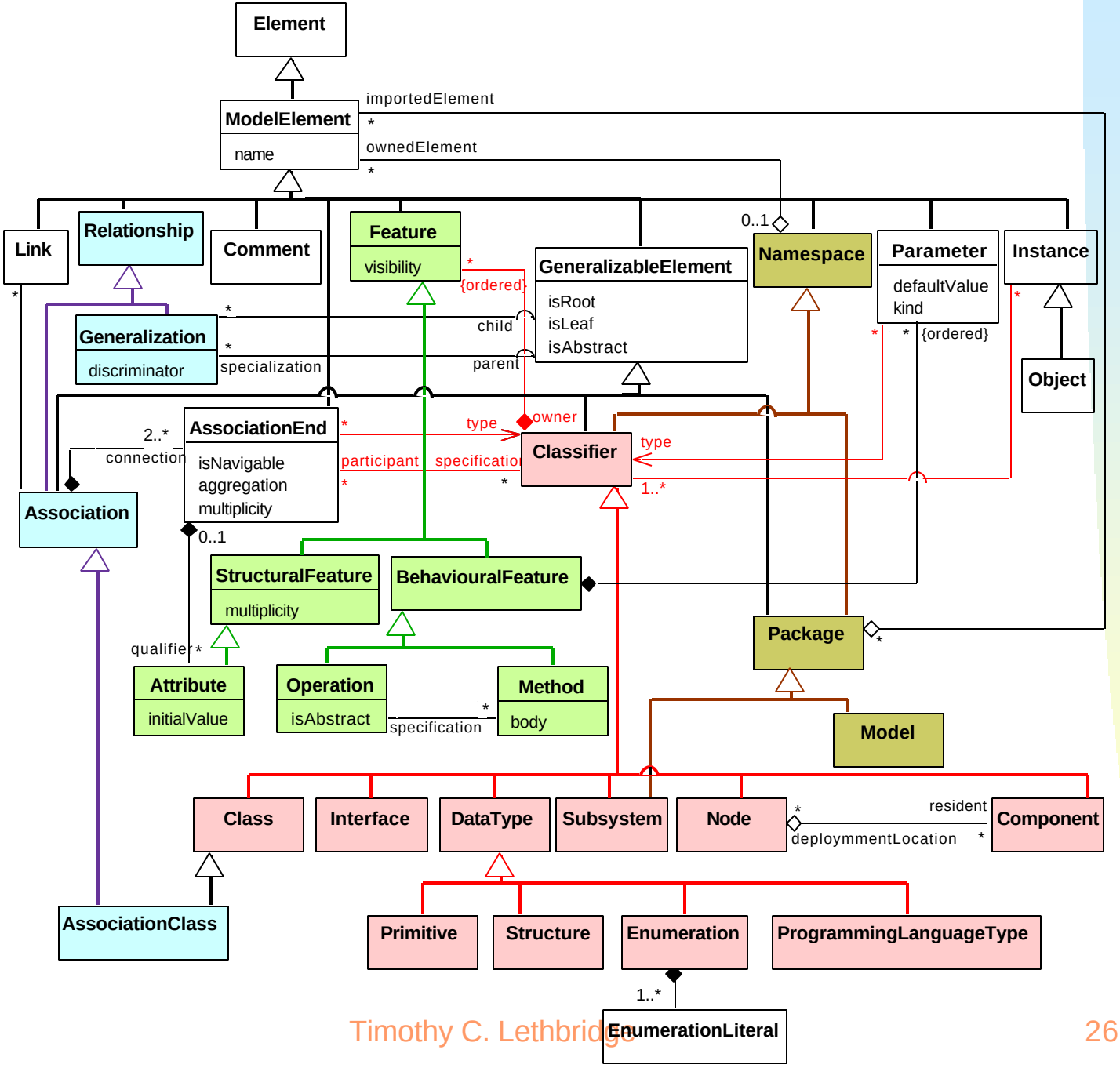
Will this standard be adopted

What extensions might be useful?

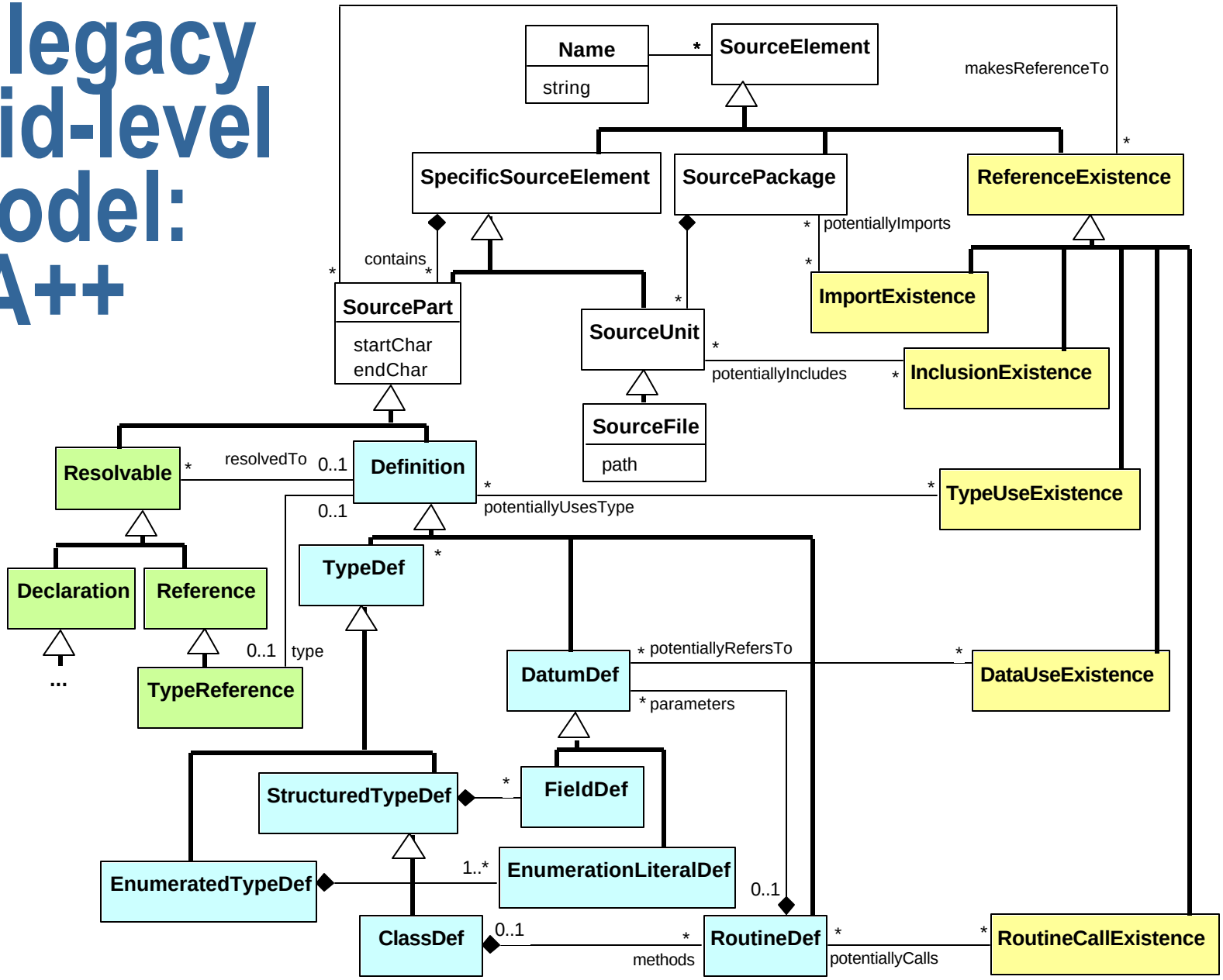
Can DMM be reasonably integrated with other *types* of metamodels? (E.g. database, architectural, and ASG metamodels, with common classes at the intersection)

(End)

The UML Meta-model



A legacy mid-level model: TA++



Another input model: part of Famix